X.25
Call Service API Guide

DC 900-1392E

Protogate, Inc.
12225 World Trade Drive, Suite R
San Diego, CA 92128
October 2004

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

X.25 Call Service API Guide

Printed in the United States of America

Contents

List of Figures

List of Tables

Preface

Purpose of Document

This document describes how to develop wide-area network (WAN) communications applications using Protogate's X.25 protocol service on Protogate's Freeway communications hardware. It also describes a high-level call service application program interface (CS API) to the X.25 service.

Intended Audience

This document should be read by applications programmers. You must be familiar with the C programming language and have a working knowledge of permanent virtual circuit (PVC) and switched virtual circuit (SVC) concepts and operations. Experience in applying state machine (*finite-state automata*) concepts to program design might also be helpful.

Organization of Document

Chapter 1 discusses application program interface concepts and procedures in general.

Chapter 2 builds on the basic concepts introduced in Chapter 1 by providing "how-to" style instructions for writing X.25 and HDLC client software that uses the CS API functions described in Chapter 5.

Chapter 3 describes the files used by the CS API at run-time.

Chapter 4 describes the CS API operational modes.

Chapter 5 is a detailed reference for the CS API functions.

Appendix A shows a copy of the cs_api.h, cs_dfine.h, cs_struc.h, cs_errno.h, and cs_proto.h include files, which give the actual values defined for the symbolically named parameter values described in Chapter 5.

Appendix B shows a pair of sample programs, pasv.c and actv.c, that connect and transfer messages back and forth.

Appendix C shows the meaning assigned to various X.25 diagnostic codes associated with the quality of service item HF_DIAG.

Appendix D is a cross reference between the short packet type names contained in the cs_x25.h file and the DLI names as used in the *Freeway X.25 Low-Level Interface* document.

The Glossary lists Freeway terminology and acronyms.

Protogate References

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

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 1100/1150 Hardware Installation Guide* DC 900-1370

- *Freeway 1200/1300 Hardware Installation Guide* DC 900-1537
- *Freeway 2000/4000 Hardware Installation Guide* DC 900-1331
- *Freeway 8800 Hardware Installation Guide* DC 900-1553
- *Freeway ICP6000R/ICP6000X Hardware Description* DC 900-1020
- *ICP6000(X)/ICP9000(X) Hardware Description and Theory of Operation* DC 900-0408
- *ICP2424 Hardware Description and Theory of Operation* DC 900-1328
- *ICP2432 Hardware Description and Theory of Operation* DC 900-1501
- *ICP2432 Hardware Installation Guide* DC 900-1502

Freeway Software Installation Support

- *Freeway Release Addendum: Client Platforms* DC 900-1555
- *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 Solaris STREAMS* DC 900-1512
- *ICP2432 User's Guide for Windows NT* DC 900-1510
- *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
- *QIO/SQIO API Reference Guide* DC 900-1355

Socket Interface Programming Support

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

Toolkit Programming Support

- *Freeway Server-Resident Application and Server Toolkit Programmer's Guide* DC 900-1325

- *OS/Impact Programmer's Guide* DC 900-1030
- *Protocol Software Toolkit Programmer's Guide* DC 900-1338

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
- *Bit-Stream Protocol Programmer's Guide* DC 900-1574
- *BSC Programmer's Guide* DC 900-1340
- *BSCDEMO User's Guide* DC 900-1349
- *BSCTRAN Programmer's Guide* DC 900-1406
- *DDCMP Programmer's Guide* DC 900-1343
- *FMP Programmer's Guide* DC 900-1339
- *Military/Government Protocols Programmer's Guide* DC 900-1602
- *N/SP-STD-1200B 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

Protogate's CS API for Freeway X.25 operates on a variety of client computer systems. In this document, bits within a byte, word, or longword are identified by the binary logarithm of their value. That is, bit *n* is valued as 2 to the *n*th power (bit 0 is 1, bit 1 is 2, bit 2 is 4, bit 3 is 8, and so on).

The term "Freeway" refers to any of the Freeway server models (for example, Freeway 500/3100/3200/3400 PCI-bus servers, Freeway 1000 ISA-bus servers, or Freeway 2000/4000/8800 VME-bus servers). References to "Freeway" also may apply to an

embedded ICP product using DLITE (for example, the embedded ICP2432 using DLITE on a Windows NT system).

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

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

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

X.25 packet types (e.g. IDATA, HOPEN_SESSION, etc.) listed in this document use the short form as contained in the cs_x25.h file. Appendix D is a cross reference between the short packet type names and the DLI names as used in the *X.25 Low-Level Interface* document.

Revision History

The revision history of the *X.25 Call Service API Guide*, Protogate document DC 900-1392E, is recorded below:

Revision	Release Date	Description
DC 900-1322A	June 1994	Original release.
DC 900-1322B	March 1995	Minor clarifications.
DC 900-1322C	April 1995	Update file names.
DC 900-1322D	May 1996	Minor corrections. Modify cs_unbind (Section 5.5.3 on page 115). Add cs_gen_event function (Section 5.6.9 on page 127), ICP reset detection (Section 5.4.2 on page 102), and dead socket detection (Table 4–1 on page 55 and Table 4–2 on page 59). Add Appendix D, "*X.25 Packet Types Cross Reference*".
DC 900-1322E	September 1996	Add new error codes in Chapter 4.

Revision	Release Date	Description
DC 900-1392A (2.5) Special version for Freeway Server 2.5 Release	February 1997	Add browser interface note (page 19). Add Section 2.2 on page 41, *HDLC Applications.* Add HDLC_RAW protocol (page 50). Update Table 4–1 on page 55, Table 4–2 on page 59 and Table 5–2 on page 72. Add cs_listen information (page 91).
DC 900-1392B Special version for Freeway Server 2.*x* Release	July 1997	Minor corrections. Add caution about exhausting memory pool (page 69). Add blocking I/O information to Chapter 4 (functions: cs_attach, cs_bind, cs_register, cs_bind, cs_accept, cs_redirect, cs_refuse, cs_deregister, cs_disconnect, cs_unbind, cs_detach). Add new cs_bufsize function (Section 5.6.10 on page 128). Change control.h file name to cs_x25.h (Appendix D).
DC 900-1392C	November 1997	Minor terminology changes for Freeway embedded product. Modify Section 1.3 on page 19. Add cs_select example (page 106).
DC 900-1392D	June 1999	Add HF_ICF_CALLBUSY qos parameter (page 136 and Table 5–3 on page 130). Update include files in Appendix A (add cs_x25.h file).
DC 900-1392E	October 2004	Update contact information for Protogate. Add writing and reading of UI-frame data (Section 2.2.2.2 on page 45). Add CS_DF_UI flag to cs_write (Section 5.4.4 on page 108). Add HUNDATA, IUNDATA, HTEST, and ITEST packet definitions (Appendix A.2, Appendix A.6, Appendix D).

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 Overview

Protogate's Freeway WAN communications product provides serial communications services access to client computers. The Freeway server product uses the transmission-control protocol/internet protocol (TCP/IP) over an Ethernet local area network. The Freeway embedded product connects your PCIbus computer directly to the WAN (for example, via an embedded ICP2432 board).

The CS API provides connection-oriented data services to the client application. The CS API provides both X.25 permanent virtual circuit (PVC), X.25 switched virtual circuit (SVC), and HDLC services through a subroutine library. The subroutine library implements functions that allow the application to use X.25 or HDLC services to access or establish virtual circuits and to transfer data.

1.1 Getting Started

Before you can use the CS API, you must make sure all the following prerequisites have been met.

Install the Freeway Hardware See the applicable hardware installation guide for your Freeway system (refer to the *Protogate References* on page 12).

Download the X.25/HDLC Software Freeway software must be download after any power-up or hardware reset. Protogate provides software that supports both the X.25 and HDLC protocols. Read the *Freeway User's Guide* for instructions on how to install the software on a Freeway server system. For a Freeway embedded system, refer to the user's guide for your particular operating system.

Configure the X.25 Service Read the *X.25/HDLC Configuration Guide*. You must configure the X.25 before you can verify the installation.

You can also choose to verify the Freeway installation and configuration by running the x25_svc test program and making sure that Freeway actually performs X.25 switched virtual circuit (SVC) operations.

1.2 Configuring the TSI and DLI

The CS API provides applications with the ability to access Freeway through transport subsystem interface (TSI) and data link interface (DLI) configuration files. Refer to the *Freeway Transport Subsystem Interface Reference Guide* and the *Freeway Data Link Interface Reference Guide* for configuration requirements.

The CS API requires the following configuration parameters to be defined as listed:

Note

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

```
Protocol      =   "Raw"
AsyncIO       =   "Yes"        // non-blocking I/O
AlwaysQIO     =   "Yes"
ReUsetrans    =   "No"
LocalAck      =   "Yes"
SessPerConn   =   1
Mode          =   "Mgr"        // for a manager session
Mode          =   "User"       // for a user session
```

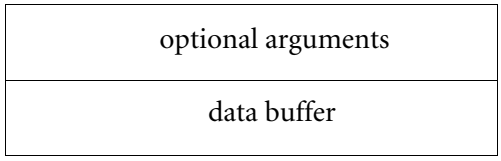
Note

After initial Freeway installation, the configuration process can also be accomplished using Protogate's browser interface which simplifies the maintenance of the Freeway, DLI, and TSI configurations. Refer to the *Freeway User's Guide* for details.

1.3 Service Access Point

Service access point (SAP) refers to a specific protocol service on Freeway. To access the X.25 protocol service on Freeway, the application uses the cs_bind request to associate SAP_X25 or SAP_SLP with the client_id previously returned by the cs_attach request. The cs_bind request creates a logical binding between the client_id and the specified SAP for the current application session, and is required before the application can access services for the specified service access point.

If the connection is configured as a *Manager* session, the application must fill in the optional arguments as specified in the *X.25/HDLC Configuration Guide*. The buffer passed to CS API in the cs_write call must be formatted as follows:



All return packets to a *Manager* session are of the CS_READ_COMPLETE event type (see Section 4.1 on page 54 and Section 4.2 on page 59 for CS API event types).

1.4 Permanent Virtual Circuit

A permanent virtual circuit (PVC) provides a non-switched virtual circuit between the application and the remote DTE. The data transfer state is active immediately after a successful cs_bind request. The circuit can be reset, but it cannot be disconnected in the

normal sense. Instead, the application terminates use of a PVC by issuing a cs_unbind request.

The circuit can be reset either by the application or by the remote DTE. When initiated by the remote DTE, a circuit reset is indicated by the status code returned by the various requests issued by the application to Freeway.

1.5 Switched Virtual Circuit

A switched virtual circuit (SVC) provides a dynamically set up virtual circuit between the application and a specified remote DTE. The method of communication requires that the connection be explicitly established by the application before data transfer occurs. The circuit can be reset or disconnected either by the application or by the remote DTE. When initiated by the remote DTE, a circuit reset or disconnect is indicated by the status code returned by the various requests issued by the application to Freeway.

1.5.1 Outgoing SVC Calls

The CS API provides applications with the ability to initiate a switched virtual circuit (SVC) connection to a remote DTE. Both blocking and non-blocking connection requests are supported.

For blocking connection requests, the application issues a cs_connect request with the block_time parameter set to non-zero. The function returns a value of zero if the connection is established. It returns an error code if the request fails or times out.

For non-blocking connection requests, the application issues a cs_connect request with the block_time parameter set to zero. The function returns immediately to allow the application to issue additional non-blocking requests or to do other work. The application must then issue a cs_connect_nb_remote request to check on the status of the original cs_connect request. Refer to Chapter 4 for non-blocking system requirements.

1.5.2 Incoming SVC Calls

The CS API provides applications with the ability to receive and handle incoming SVC connection indications (calls) from remote DTEs. This ability is provided by allowing the application to register itself with the X.25 protocol service as an incoming connection handler; the application then listens for incoming connection indications from that X.25 protocol service.

An application indicates its desire to receive incoming connection indications by using the cs_register function. This function specifies the application's bound attachment to the X.25 protocol service and optional filtering information describing the characteristics of incoming connections to be compared against. The X.25 protocol service inserts the application's registration into an internal list of incoming connection handlers.

If the registered filtering information matches the characteristics of an incoming connection indication when the X.25 protocol service receives one, the service sends the call information back to the associated application.

If no optional filtering information was provided when the application registered an incoming connection handler, the filter matches all incoming connection indications and the service sends the call information back to the associated application.

The application issues a cs_listen request to receive incoming call information and examines the returned information to determine how it wants to handle the incoming connection indication. If the call information indicates that the X.25 protocol has already accepted the call on behalf of the application, the application may use the virtual circuit or disconnect it.

If the call information indicates that the X.25 protocol is simply notifying the application of an incoming connection indication, the application has the following options available for handling an incoming connection indication, which it selects based on the call information returned by the cs_listen request:

1. Accept the connection. The application has determined that it wants to communicate with the remote DTE, so it issues the cs_accept request to establish the connection.

2. Refuse the connection. The application has determined that the connection should not be established and issues the cs_refuse request.

3. Redirect the connection. The application has determined that it does not want to accept the connection, but it will allow other applications to accept the connection if they desire. It issues the cs_redirect request that causes the X.25 protocol service to continue scanning the internal connection handler list from the current location for other applications whose filters match the characteristics of the incoming connection.

The application also has the option of passing the incoming connection information to another application, which can perform one of the three handling functions itself. This is done using the call token value, which is a unique identifier reported with the incoming connection indication. The token value is used in the cs_accept, cs_refuse, and cs_redirect functions to identify the call that is being handled; any application that has the correct token value can perform one of these functions.

The CS API does not apply a timeout to the connection indications. The application can hold that indication without taking action on it for as long as is desired. It should be noted, however, that many X.25 networks impose a timeout on connection indications; when the application finally responds to the connection indication, it should be prepared to receive an error return indicating that the X.25 network has cancelled the incoming connection indication already.

Once the application establishes an actual connection for a specific client_id, that client_id is marked as busy. No more connection indications for that client_id will be sent to the application until the application's connection for that client_id is terminated, at which

point the service will again mark that client_id as available to handle incoming connections.

While the application is busy with one connection, the X.25 protocol service places any additional connection indications for that application on hold, unless specifically configured to redirect such calls. This is useful to an application designed to intercept each incoming connection indication and validate it further before passing its call token value to another application authorized to handle the call.

Any application that does not want additional incoming connection indications to be placed on hold may issue the cs_deregister request to cancel its previous cs_register request to the X.25 protocol service. Later, after the connection is disconnected, the application might again issue the cs_register and cs_listen requests to indicate its readiness to handle another incoming connection indication. However, if the application uses the HF_ICF_CALLBUSY quality of service parameter to configure the incoming call filter to redirect calls when busy, then the application need not call cs_deregister to avoid placing calls on hold.

1.6 Quality of Service

The quality of service (QOS) is a specification of the characteristics of the X.25 connection. QOS specifications can include X.25 call facilities, virtual circuit local priority, X.25 cause and diagnostic codes for circuit clear or reset events, or incoming connection (X.25 call indication) filter specifications.

The available QOS characteristics allow an application to determine or negotiate the characteristics of transmission needed to communicate with the remote DTE. The application can specify a qos parameter in each of the following CS API requests: cs_accept, cs_connect, cs_disconnect, cs_redirect, cs_refuse, cs_register, and cs_reset. In addition, the application can supply a ret_qos parameter in each of the following CS API requests to receive the QOS values proposed or negotiated by Freeway: cs_listen, cs_connect, and cs_connect_nb_remote.

For more information on the use of the QOS specification, see individual reference sections for the specific CS API request functions described in Chapter 5.

Chapter

2 Writing CS API User-Client Software

This chapter shows how to use the CS API to establish X.25 permanent or switched virtual circuits, send and receive data, handle error conditions, and terminate PVC or SVC use. You should be familiar with the material in Chapter 5 to fully understand the procedures presented in Section 2.1 below. Table 5–1 on page 71 summarizes the available CS API request functions.

The CS API also supports the HDLC layer. HDLC applications are discussed in Section 2.2 on page 41.

For the sake of simplicity, it is assumed that Freeway has already been downloaded and configured, and that the appropriate data links have been enabled. For more information about how to download Freeway, refer to the *Freeway User's Guide*. For more information about how to configure Freeway and enable data links, refer to the *X.25/HDLC Configuration Guide*.

2.1 X.25 Applications

2.1.1 X.25 Connection Establishment

X.25 supports two types of connections: permanent virtual circuits (PVCs) and switched virtual circuits (SVCs). Before an application can transfer data on either a PVC or SVC, the application must first establish a connection.

The method for establishing a PVC connection differs from that for establishing an SVC connection. Each method is described in the sections that follow.

2.1.1.1 PVC Connections

An X.25 PVC is permanent in the same sense that a leased phone line is permanent. Both are by definition point-to-point connections between two fixed pieces of data terminal equipment. In each case, a DTE can connect only to the DTE to which it is permanently connected.

To extend the analogy, although a leased phone line permanently connects two DTEs, no data transfer can occur until both DTEs are simultaneously online. A CS API application wanting to use an X.25 PVC must perform an operation analogous to picking up the phone before sending data.

Before data transfer can take place, a CS API application must perform the following actions in the sequence given to establish a PVC connection. Figure 2–1 illustrates these actions and their associated application state transitions.

1. Issue a cs_attach request to attach the application to the desired X.25 PVC.
2. Issue a cs_bind request to bind the application to the X.25 protocol service, and thereby connect the PVC.

2.1.1.2 SVC Connections

An X.25 SVC, like a dialed phone connection, is both temporary and switched. In each case, the point-to-point connection between the two pieces of data terminal equipment requires one DTE to call and the other to answer to set up the connection; furthermore, the connection is temporary in that either DTE can hang up (or disconnect) the call at any time.

Many phone systems allow the call recipient to answer, refuse, or forward an incoming call. In a similar manner, the CS API allows the application to answer, refuse, or redirect an incoming call.

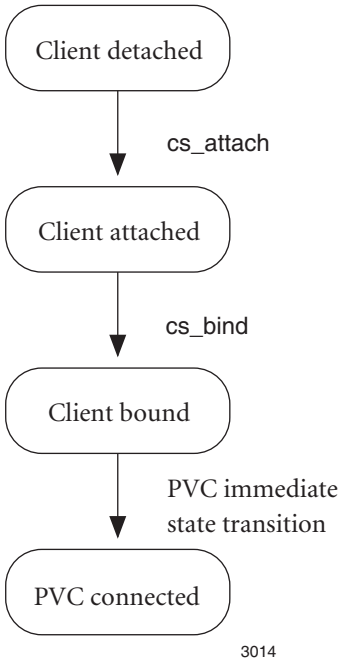


Figure 2–1: X.25 PVC Connection Establishment

SVC Call Placement

A CS API application wanting to place an X.25 SVC call must perform the following actions in the sequence given to establish an SVC connection before data transfer can take place. Figure 2–2 illustrates these actions and their associated application state transitions.

1. Issue a cs_attach request to attach the application to the desired X.25 network.
2. Issue a cs_bind request to bind the application to the X.25 protocol service.
3. Issue a blocking or non-blocking cs_connect request to initiate X.25 call placement.
4. If the block_time parameter in the cs_connect request was set to zero and a user interrupt has been registered with the cs_init call, the application can continue other processing while Freeway tries to establish the connection; in this case, the application issues a cs_connect_nb_remote request to determine whether or not the connection has been established.
5. Look for the blocking cs_connect request or the cs_connect_nb_remote request to indicate whether or not the connection has been established. Once the connection is established, data transfer is allowed.

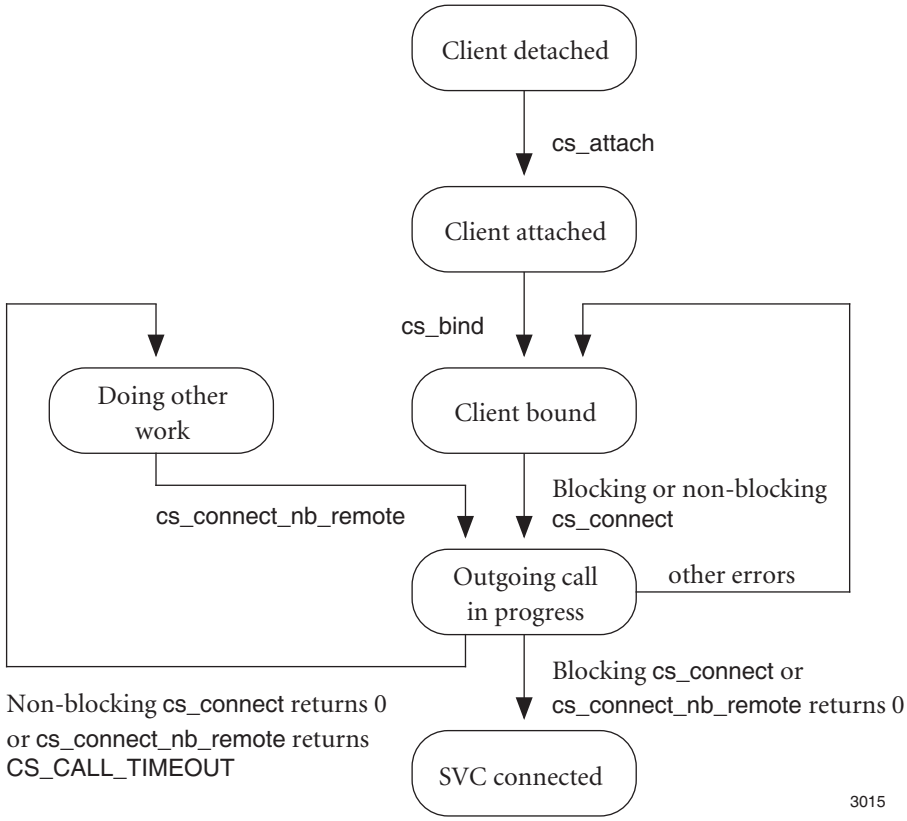


Figure 2–2: X.25 SVC Call Placement

SVC Call Reception

A CS API application wanting to receive an X.25 SVC call must perform the following actions in the sequence given to establish an SVC connection before data transfer can take place. Figure 2–3 illustrates these actions and their associated application state transitions.

1. Issue a cs_attach request to attach the application to the desired X.25 network.
2. Issue a cs_bind request to bind the application to the X.25 protocol service.
3. Issue a cs_register request to register an incoming call handler profile for the expected incoming call. The profile parameters determine the selectivity with which incoming calls are screened for a match.
4. Issue a cs_listen request to check for the expected incoming call.
5. If cs_listen returns a status of CS_NO_ERROR and the incoming call state is CS_AUTO_CONNECT, Freeway has completed the incoming connection and data transfer is immediately allowed.
6. If cs_listen returns a status of CS_NO_ERROR and the incoming call state is CS_INC_CALL, the application can take one of three actions.
    - Issue a cs_accept request to complete the incoming call connection and allow data transfer.
    - Issue a cs_redirect request to allow Freeway to present the incoming call to another application, then issue a cs_listen request to wait for the next incoming call.

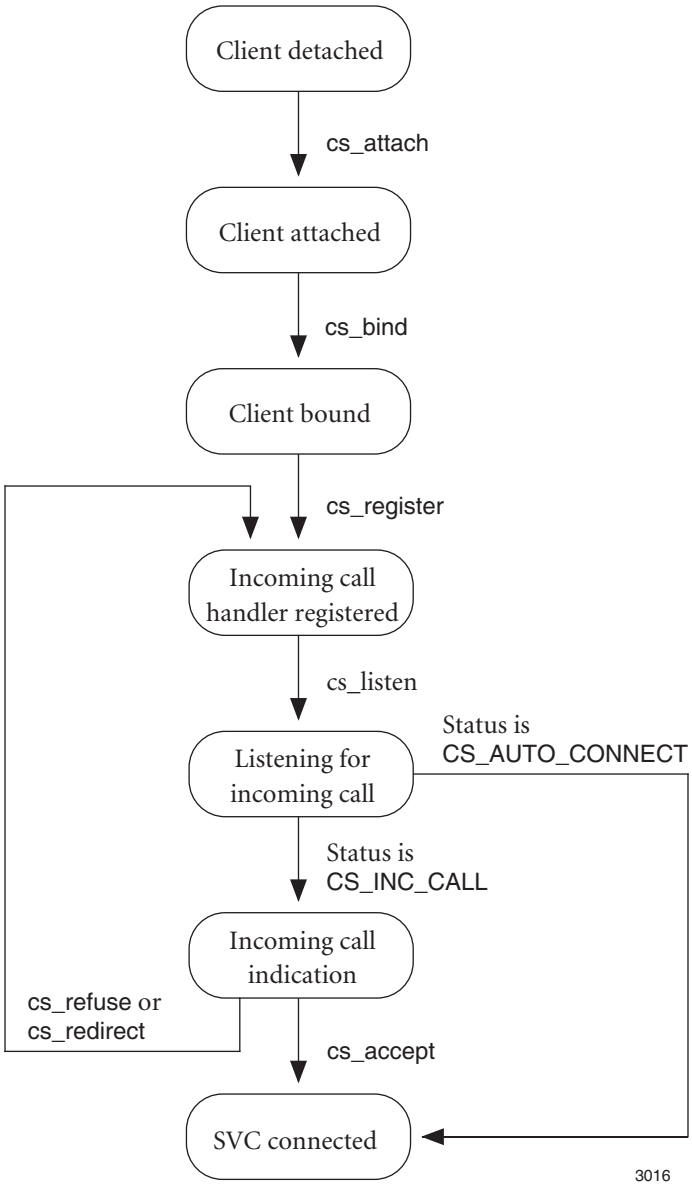


Figure 2–3: X.25 SVC Call Reception

- Issue a cs_refuse request to clear the call and prevent Freeway from presenting the incoming call to another application, then issue a cs_listen request to wait for the next incoming call.

After a CS API application accepts an incoming call, Freeway places all additional incoming calls for that client_id on hold until the current call is cleared or until the network DCE cancels the incoming call. Calls can be placed on hold even if they match the registered incoming call handler profile of other CS API applications. After the currently active virtual circuit is disconnected, the CS API application simply issues a cs_listen request to get the next incoming call.

If the CS API application does not want additional calls to be placed on hold after accepting an incoming call, the application may issue a cs_deregister request to delete its incoming call handler profile from the X.25 protocol. When the currently active virtual circuit is disconnected, the application would then issue a cs_register request before issuing a cs_listen request. However, if the application uses the HF_ICF_CALLBUSY quality of service parameter to configure the incoming call filter to redirect calls when busy, then the application need not call cs_deregister to avoid placing calls on hold.

Fast Select Call Transactions

Transaction processing applications for which the data transfer requirement is very small (128 bytes or less) can decrease the total X.25 overhead by using X.25 SVC fast-select facilities. Use of these facilities allows the X.25 call request to include up to 128 bytes of user data, such as a data base query request. The call recipient also returns up to 128 bytes of user data in an X.25 clear request.

The value of the fast-select operation is that the transfer of the 128 byte query and response data does not require the acceptance of the call. Contrast this with the overhead required to establish an SVC call, exchange the query and response data after the call is accepted, then terminate the SVC connection. The result is that fast-select proce-

dures can be used to reduce a simple query and response exchange from six X.25 packets to only two X.25 packets.

The procedures for using X.25 fast select are the same as those for handling normal SVC calls, except that the call recipient usually issues a cs_refuse request rather than a cs_accept request. The qos parameter in the cs_connect request must specify both the X.25 fast-select facility and the user query data. The qos parameter in the cs_refuse request specifies the user response data.

2.1.2 X.25 Connection Operation

After the application's PVC or SVC is connected, the CS API allows the application considerable freedom in the actual operation of the virtual circuit. Various CS API functions are used in combination to achieve the operations listed below. Figure 2–4 illustrates these operational choices of action.

- Initiate any of the following write operations:
    - Write normal data with full control over the X.25 D-bit, M-bit, and Q-bit using a cs_write request
    - Write an X.25 interrupt with 1 to 32 bytes of data using a cs_write request, and read subsequent acknowledgment
    - Write an X.25 reset with qos data using a cs_reset request, and read subsequent acknowledgment
    - Write an X.25 clear request with qos data using a cs_disconnect request
- Initiate any of the following read operations:
    - Read normal data with full access to the X.25 D-bit, M-bit, and Q-bit using a cs_read request

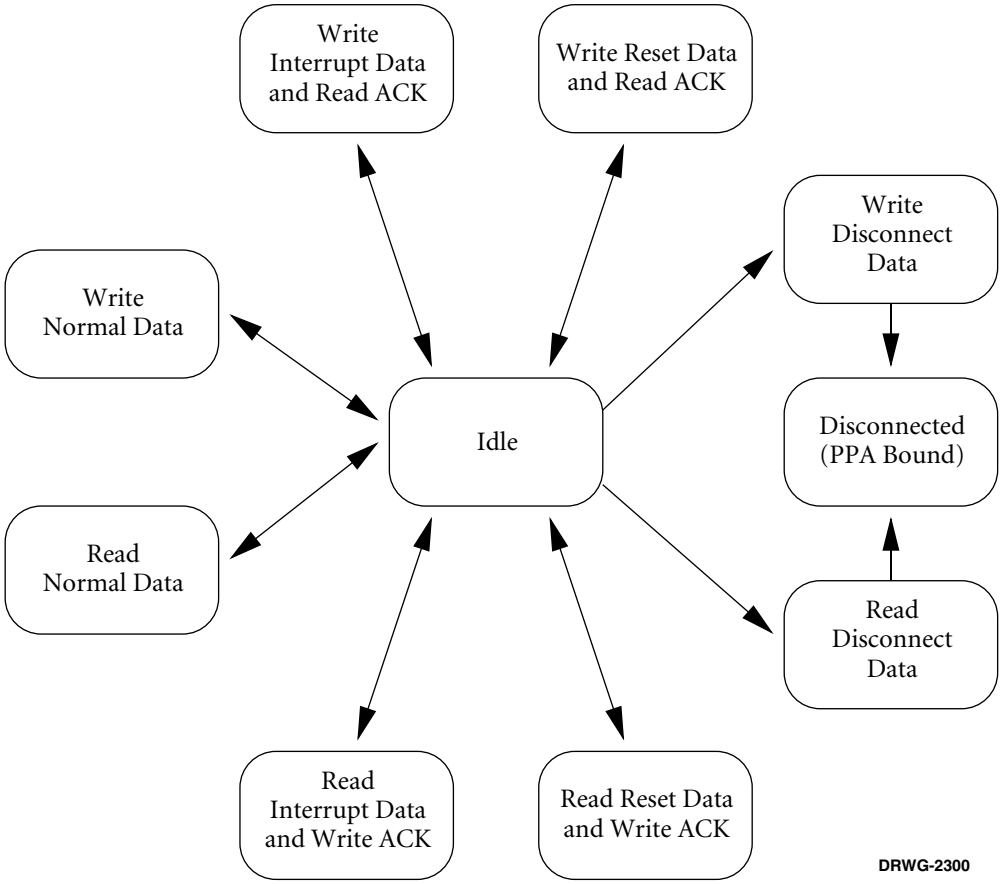


Figure 2–4: X.25 Connection Operation

- Read an X.25 interrupt with 1 to 32 bytes of data using a cs_read request, and write acknowledgment
- Read an X.25 reset with qos data using a cs_read request, and write acknowledgment
- Read an X.25 clear request with qos data using a cs_read request

2.1.2.1 X.25 Normal Data Transfer

Most applications need to read and write normal data. The CS API supports these operations using cs_read and cs_write requests, respectively.

X.25 D-bit, M-bit, and Q-bit

When writing normal data, the application has full control of the D-bit, M-bit, and Q-bit supported by the X.25 protocol. The application controls these bits for each cs_write request by appropriately specifying a proto_flag parameter value. The symbolically named values CS_DF_X25D, CS_DF_X25MORE, or CS_DF_X25Q, respectively, can be used in combinations consistent with the X.25 protocol rules.

When reading normal data, the application has full access to the D-bit, M-bit, and Q-bit supported by the X.25 protocol. The application detects these bits by checking the returned value of the ret_flags parameter. The symbolically named values CS_DF_X25D, CS_DF_X25MORE, and CS_DF_X25Q, respectively, can appear in combinations consistent with the X.25 protocol rules.

The M-bit (CS_DF_X25MORE) specifies that the data is part of a larger message or complete packet sequence.[1] X.25 protocol rules allow the M-bit only in full data packets, or partial data packets that also contain the D-bit. If the application sets the M-bit (with-

1. A complete packet sequence consists of a set of data packets for which all data packets (except possibly the last one) are filled and have the M-bit set to one; the last data packet might or might not be full, and has the M-bit reset to zero or has the D-bit set to one.

out setting the D-bit) when writing data, the buf_length parameter must equal Freeway's segmentation buffer size; since the application cannot dynamically determine Freeway's segmentation buffer size, it is the programmer's responsibility to specify this value correctly for the current Freeway configuration.

The D-bit (CS_DF_X25D) specifies that confirmation of data delivery is required from the DTE that receives the data; such confirmation provides an end-to-end confirmation of delivery, rather than the normal X.25 acknowledgment that the data has merely entered the network. The D-bit also ends the current complete packet sequence, whether or not the M-bit is cleared.

Note

An SVC can use the D-bit only if the qos parameter HF_D_BIT_SUPPORT facility was successfully negotiated during call establishment. A PVC can use the D-bit only if the X.25 network is configured to support D-bit use on that PVC.

The Q-bit (CS_DF_X25Q) specifies that the data is qualified and is not part of the normal data stream. The Q-bit is used to distinguish between two logically separated categories of information (such as control information and actual data), and is often used to exert control operations (such as X.25 PAD functions) outside the normal data stream. The Q-bit must be the same for all packets within a complete packet sequence.

Writing Normal Data

When issuing a cs_write request, the application uses the block_time parameter to select whether the function blocks until the transmission is acknowledged, blocks only until the transmit window is open, or returns immediately. To write normal data, the proto_flag parameter must be zero, or it must specify a combination of the X.25 D-bit, M-bit, and Q-bit.

Although the application can issue a blocking cs_write request, it must check the return status to determine whether or not normal data was actually written. An error status might indicate a need for the application to take other action (such as reading interrupt data, reading reset data, reading disconnect data, or issuing a cs_reset request) before again attempting to write normal data.

Reading Normal Data

When issuing a cs_read request, the application uses the block_time parameter to select whether the function blocks until data is available or returns immediately. To read normal data, the event parameter value must be CS_READ_COMPLETE.

Although the application can issue a blocking cs_read request, it must check the return status to determine whether or not normal data was actually read. An error status might indicate a need for the application to take other action (such as reading interrupt data, reading reset data, reading disconnect data, or issuing a cs_reset request) before again attempting to read normal data.

2.1.2.2 X.25 Interrupt Data Transfer

Some applications require access to X.25 interrupt procedures, which allow the transfer of up to 32 bytes of data through an X.25 interrupt packet. Since the X.25 interrupt packet is not subject to normal flow control procedures, this feature can be used when the transmit window is closed to determine if the application in the remote DTE is still alive.

Writing Interrupt Data

To write interrupt data, the application issues a cs_write request with the proto_flag parameter set to CS_DF_X25OOB. The block_time parameter value is used to select whether the function blocks until the interrupt is acknowledged or returns immediately. In the latter case, the CS API might later inform the application when interrupt has been

acknowledged by returning CS_INDX25OOB_ACK (CS_INTERRUPT_ACK) as an indication status within the ind_array specified in a cs_select request or as a non-blocking I/O event.

Although the application can issue a blocking cs_write request, it must check the return status to determine if interrupt data was actually written. An error status might indicate a need for the application to take other action (such as reading interrupt data, reading reset data, reading disconnect data, or issuing a cs_reset request) before again attempting to write interrupt data.

Note

The X.25 protocol rules require that one interrupt be acknowledged before another interrupt is sent. It is the application's responsibility to check for interrupt acknowledgment before sending a second interrupt.

Reading and Acknowledging Interrupt Data

The CS API can inform the application that interrupt data is present by returning CS_INDX25OOB (CS_INTERRUPT) as an error status for cs_read or cs_write requests, as an indication status within the ind_array specified in a cs_select request, or as a non-blocking I/O event.

To read interrupt data, the application issues a cs_read request with the event parameter value set to CS_DF_X25OOB (CS_INTERRUPT). When the application reads interrupt data, the CS API automatically acknowledges receipt of the X.25 interrupt.

The application must check the return status to determine if interrupt data was actually read. An error status might indicate a need for the application to take other action (such as reading reset data, reading disconnect data, or issuing a cs_reset request).

2.1.2.3 X.25 Circuit Reset

The application or the network DCE can reset an active X.25 PVC or SVC at any time. Resetting a virtual circuit cancels all operations in progress. Any data written (whether normal data or interrupt data) that is not yet acknowledged remains unacknowledged. Only the reset itself and the qos data associated with the reset are acknowledged.

Issuing a Reset Request

The application resets a virtual circuit by issuing a cs_reset request. The qos parameter can be used to specify qos data to be sent in the actual X.25 reset request or to adjust the virtual circuit's local priority on Freeway.

The application uses the block_time parameter to specify a time limit for the cs_reset request to complete. If the cs_reset function returns the error status CS_CALL_TIMEOUT, the application cannot use the virtual circuit for data transfer until a CS_INDX25RSET_ACK (CS_RESET_SUCCESS) is returned as an indication status within the ind_array specified in a cs_select request or as a non-blocking I/O event.

Although the application can issue a blocking cs_reset request by specifying a non-zero block_time parameter value, the application must check the return status to determine whether or not the reset was successfully completed. An error status might indicate a need for the application to take other action (such as reading reset data, reading disconnect data, or issuing another cs_reset request) before resuming data transfer.

Detecting and Acknowledging a Reset Indication

The CS API informs the application when the network DCE has reset the virtual circuit by returning CS_RESET as an error status for cs_read or cs_write requests, as an indication status within the ind_array specified in a cs_select request, or as a non-blocking I/O event.

After receiving notification of a CS_RESET status, the application issues a cs_read request specifying an event parameter value of CS_RESET to read the qos data associated with the DCE reset indication. The CS API automatically acknowledges the reset.

2.1.2.4 X.25 Procedure Errors — CS_STA_ERROR

When Freeway detects a violation of the X.25 protocol on any virtual circuit, it can reset the virtual circuit. When this occurs, Freeway reports the error status CS_STA_ERROR to the application. The CS_STA_ERROR status serves to notify the application that any data written (whether normal data or interrupt data) and not yet acknowledged remains unacknowledged. The application should issue a cs_reset request to resynchronize its own virtual circuit operations with those of Freeway.

2.1.3 X.25 Connection Termination

The application or the network DCE can disconnect an active X.25 SVC at any time. Also, the application or the network DCE can cease to support PVC operation at any time.

2.1.3.1 PVC Connection Termination

The network DCE cannot disconnect a PVC, but can report the PVC failed using a HF_CAUSE facility code value of 1 in a reset indication. The network DCE can also report the PVC operational using a HF_CAUSE facility code value of 9 in a subsequent reset indication.

The application also cannot disconnect a PVC, but can terminate its use of a PVC connection by issuing a cs_unbind request. A PVC that is not bound to an application does not acknowledge receipt of data from the DCE, but can hold up to a full packet window's worth of received data for delivery to the next application that binds to the PVC.

2.1.3.2 SVC Connection Termination

The CS API informs the application when the network DCE has terminated an SVC by returning CS_LOST_CONN as an error status for cs_read or cs_write requests, as an indication status within the ind_array specified in a cs_select request, or as a non-blocking I/O event.

After receiving notification of a CS_LOST_CONN status, the application issues a cs_read request specifying an event parameter value of CS_LOST_CONN to read the qos data associated with the DCE clear indication. No further use of the SVC is possible without re-establishing the connection.

The application terminates an active SVC by issuing a cs_disconnect request. The application can specify optional qos data for transfer in the associated X.25 DTE clear request.

2.2 HDLC Applications

2.2.1 HDLC Connection Establishment

Before data transfer can take place, a CS API application must perform the following actions in the sequence given to establish an HDLC connection. Figure 2–5 illustrates these actions and their associated application state transitions.

1. Issue a cs_attach request to attach the application to the desired HDLC data link.
2. Issue a cs_bind request to bind the application to the HDLC protocol service.
3. Issue a blocking or non-blocking cs_connect request to initiate the HDLC connection. This enables the associated data link.
4. If the block_time parameter in the cs_connect request was zero, the application can continue other processing while Freeway tries to establish the connection; in this case, the application must periodically issue a cs_connect_nb_remote request to determine whether or not the connection has been established.
5. Look for the blocking cs_connect request or the cs_connect_nb_remote request to indicate whether or not the connection has been established. Once the connection is established, data transfer is allowed.

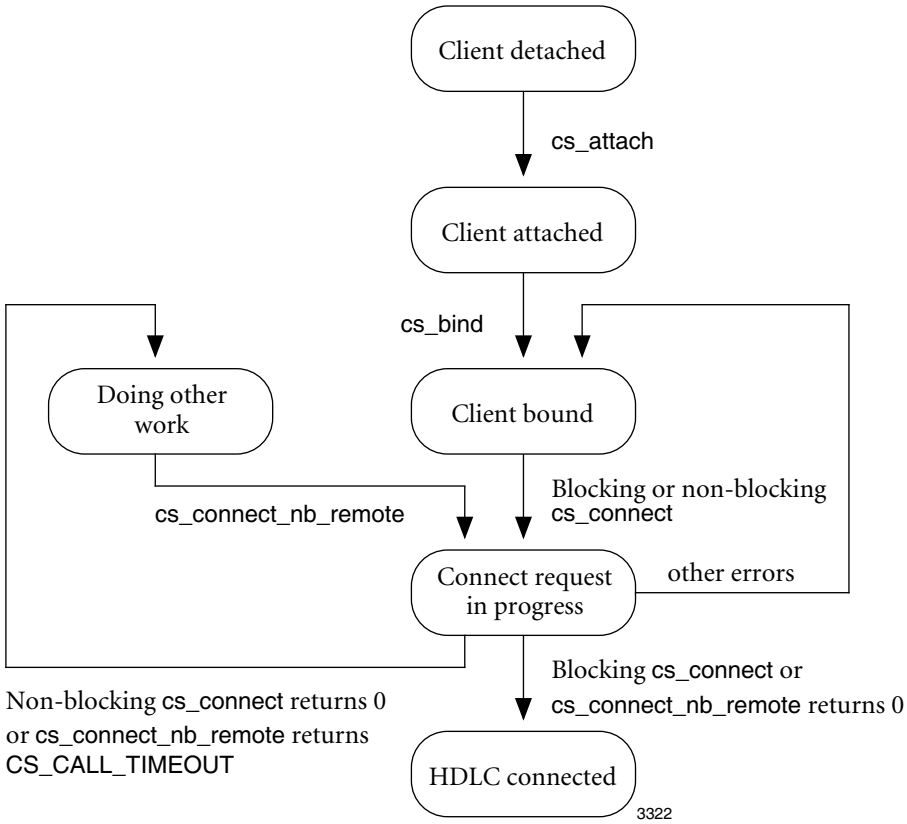


Figure 2–5: HDLC Connection Establishment

2.2.2 HDLC Connection Operation

After the application establishes an HDLC connection, the CS API allows the application considerable freedom in the actual operation of the HDLC connection. Various CS API functions are used in combination to achieve the operations listed below. Figure 2–6 illustrates these operational choices of action.

- Initiate any of the following write operations:
  - Write normal data using a cs_write request
  - Write UI-frame data using a cs_write request

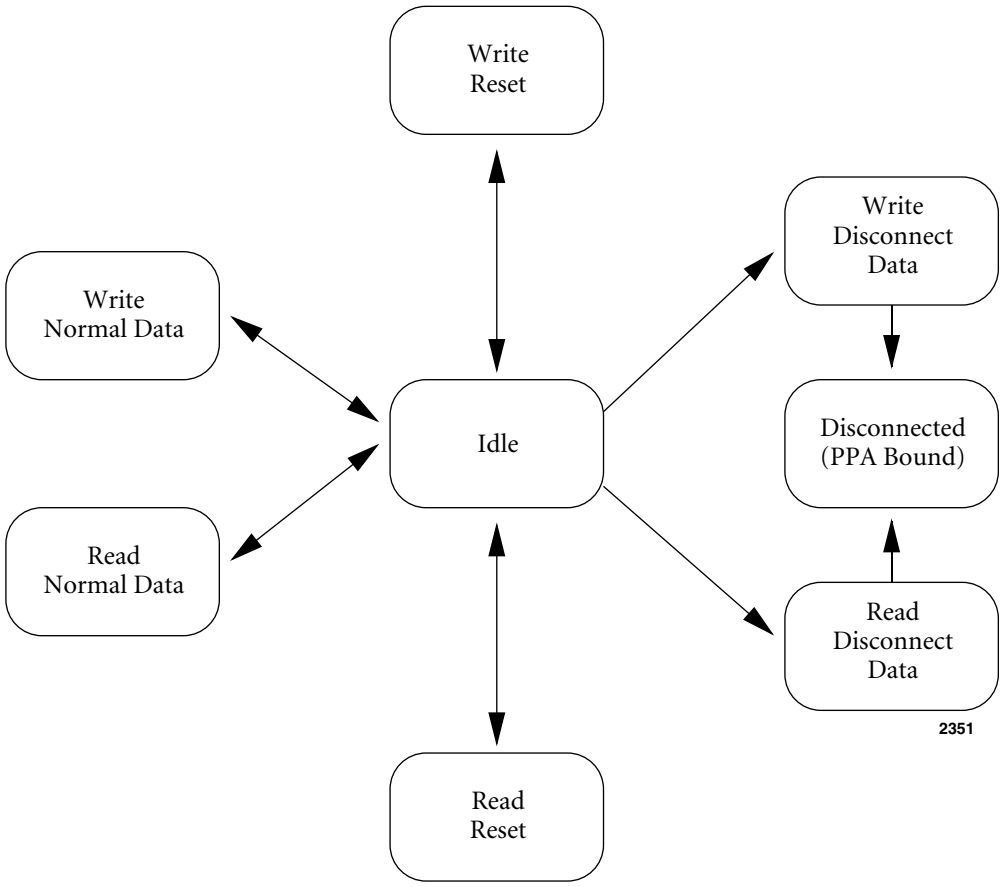


Figure 2–6: HDLC Connection Operation

- Write an HDLC reset using a cs_reset request
- Write an HDLC disconnect request using a cs_disconnect request

- Initiate any of the following read operations:
  - Read normal data using a cs_read request
  - Read UI-frame data using a cs_read request
  - Read an HDLC reset event using a cs_select request or as an error status returned by cs_read or cs_write
  - Read an HDLC disconnect event using a cs_select request or as an error status returned by cs_read or cs_write

2.2.2.1 HDLC Normal Data Transfer

Most applications need to read and write normal data. The CS API supports these operations using cs_read and cs_write requests.

Writing Normal Data

When issuing a cs_write request, the application uses the block_time parameter to select whether the function blocks until the transmission is acknowledged, blocks only until the transmit window is open, or returns immediately. To write normal data, the proto_flag parameter must always be zero (NULL).

Although the application can issue a blocking cs_write request, it must check the return status to determine whether or not normal data was actually written. An error status might indicate a need for the application to take other action before again attempting to write normal data.

An application that needs to detect the completion of the transmission of a message or file may do so by accounting for the receipt of an event notification (CS_WRITE_FAILED or CS_WRITE_COMPLETE) for each call made to the cs_write function.

Reading Normal Data

When issuing a cs_read request, the application uses the block_time parameter to select whether the function blocks until data is available or returns immediately. To read normal data, the event parameter value must always be zero (NULL).

Although the application can issue a blocking cs_read request, it must check the return status to determine whether or not normal data was actually read. An error status might indicate a need for the application to take other action before again attempting to read normal data.

2.2.2.2 HDLC UI Frame Data Transfer

UI-frame transfers are supported when the ISO HDLC Option 4 has been selected during the configuration of the link. The CS API supports this feature with cs_read and cs_write requests that contain the CS_DF_UI special qualifier flag.

Writing UI Frame Data

To provide data for the transmission of a UI frame, the application uses the cs_write request in the manner described in Section 2.2.2.1 on page 44, with one difference: the proto_flag parameter of the cs_write request is set to CS_DF_UI instead of zero.

Reading UI Frame Data

To obtain the data from a received UI frame, the application uses the cs_read request in the manner described in Section 2.2.2.1 on page 44, with the proviso that UI-frame data is indicated by a returned ret_flag parameter value of CS_DF_UI.

2.2.2.3 HDLC Reset

The application or the remote DTE can reset an active HDLC connection at any time. Resetting an HDLC connection cancels all operations in progress. Any data written that is not yet acknowledged remains unacknowledged.

Issuing a Reset Request

The application resets an HDLC connection by issuing a cs_reset request. The application uses the block_time parameter to specify a time limit for the cs_reset request to complete. If the cs_reset function returns the error status CS_CALL_TIMEOUT, the application cannot use the HDLC connection for data transfer until a CS_RESET_SUCCESS is returned as an indication status within the ind_array specified in a cs_select request.

Although the application can issue a blocking cs_reset request by specifying a non-zero block_time parameter value, the application must check the return status to determine whether or not the reset was successfully completed.

Detecting a Reset Indication

The CS API informs the application when the remote DTE has reset the HDLC connection by returning CS_RESET as an error status for cs_read or cs_write requests, or as an indication status within the ind_array specified in a cs_select request.

After receiving notification of a CS_RESET status, the application must perform any higher-level recovery procedures required by its application.

2.2.3 HDLC Connection Termination

The application or the remote DTE can terminate an active HDLC connection at any time. The CS API informs the application when the remote DTE has terminated an HDLC connection by returning CS_LOST_CONN as an error status for cs_read or cs_write requests, or as an indication status within the ind_array specified in a cs_select request. The application terminates an HDLC connection by issuing a cs_disconnect request.

2.3 Client Program Environment

This section offers tips for meeting application program requirements whose solutions might be influenced by the environment in which the program executes.

2.3.1 Handling Unsolicited Input

An application program that performs other work in addition to X.25 communications functions must be able to efficiently decide what task to do next. For example, a program that interacts with both Freeway and a human operator must be able to handle unsolicited input from either source. In this case, the application program should not block waiting for input from one source. Instead, the application program must alternately check for input from either source, though it may sleep periodically for a short interval to avoid monopolizing the client CPU.

2.3.2 Child Processes

The CS API implementation does not allow two processes to share the same socket. On UNIX systems, a forked child process begins execution with an open socket owned by the parent process. Child processes must call the cs_init function before any other CS API function to allocate new system resources to the child and avoid affecting parent-process access to the socket.

Chapter

3 CS API Run-time File Dependencies

The CS API accesses two text files. The first file is the run-time CS API configuration file. The second file is an optional run-time event log file named Dbug*nnnn*.X25, where *nnnn* is the hex value of the process ID.

3.1 CS API Configuration File

The CS API reads its configuration file for specific run-time parameters on the initial attachment request (cs_attach). These parameters give the CS API flexibility to support different operating requirements. The application specifies the name of the CS API configuration file as a parameter to the cs_init function.

The CS API uses its configuration file to associate each X.25 circuit name with a DLI session name. The CS API configuration file is formatted a follows:

- Any line starting with the # character in the first column is treated as a comment line and is ignored.
- Configuration lines consist of the following fields, separated by a blank, a comma, or a tab:

  <circuit name> <session name> <protocol> <pvc><flow>

  where:

  - **circuit name** is the local name passed in the cs_attach function call
  - **session name** is the name used in the DLI configuration file

- **protocol** is either X.25, HDLC, or HDLC_RAW and is used as the default protocol in the cs_bind function if the application passes a NULL for the protocol parameter
- **pvc** is the PVC station number previously configured on the ICP
- **flow** is the flow control for this circuit; if blank, the default from the "sys_defaults" record is used. Flow control specifies the number of outstanding cs_write requests permitted without a CS_WRITE_COMPLETE notification

Note

A special circuit name, sys_defaults, is used to identify the DLI configuration file and define the maximum number of CS client connections and maximum flow control.

For example:

```
#This is a comment line
#ckt name     DLI_file     max_clients     default_flow
sys_defaults  dli_config   35              8

#ckt name     sess name     protocol     PVC     flow
data_link     RawSess0      HDLC         0       8
SVC           RawSess1      X.25         0       8
PVC_1         RawSess1      X.25         1       8
```

The configuration lines are not case sensitive.

The CS API software is capable of multiplexing many X.25 circuits over one or more ports on Freeway. Because the CS API uses the underlying DLI and TSI API software when talking to Freeway, correct setup of the DLI and TSI configuration files is essential to proper operation.

When you receive your copy of Protogate X.25 for Freeway, examine the default setup in these configuration files carefully. Typically, the DLI configuration file should define

a MANAGER raw session and a USER raw session for each physical port on Freeway. The X.25 circuit configuration file should associate each HDLC or X.25 circuit name with the DLI USER raw session name for the correct port on Freeway.

For additional information on configuration of CS API, DLI, TSI and the Freeway ICPs for X.25 operations, see the *X.25/HDLC Configuration Guide*.

3.2 CS API Log File

The CS API can record run-time events into a log file for monitoring or diagnostic purposes. The log file is a circular file that contains up to 20,000 records of 80 bytes each. The log file is located in the client's local directory and is named Dbug*nnnn*.X25, where *nnnn* is the hex value of the process ID.

Debug logging is activated by placing debuglog function calls throughout the code to be monitored. For example, on March 17 at 12:34:14, the following line of code:

```
debuglog (“value returned for %d tries was %d”, 2, 5);
```

results in the following text line being added to the debug log file:

```
Mar 17 12:34:14 – value returned for 2 tries was 5
```

Chapter

4 CS API Operational Modes

Protogate's CS API for Freeway X.25/HDLC provides full support for non-blocking I/O for all CS API functions that involve LAN or WAN transactions. In these cases, the function value returned indicates whether the function was called without error. When the action initiated by the called function later completes, the CS API informs the application's event handler routine of the completion status of the requested action.

Protogate's CS API also supports blocking I/O to ease programming complexity for simple applications that do not require support for non-blocking I/O. If the application does not declare an event handler routine when initializing the CS API, it must wait for or poll for the function results. If the application declares an event handler routine, it may use the block_time parameter in any CS API function to control whether the function uses non-blocking I/O or to specify a time limit on blocking I/O.

Protogate's CS API uses the data link interface (DLI) raw mode feature to provide an X.25-specific interface between the DLI and an application. The CS API communicates with the DLI using non-blocking I/O only.

The sections that follow describe the operational differences between using non-blocking I/O and blocking I/O in the CS API functions.

4.1 Non-Blocking I/O Operations

When using non-blocking I/O, CS API function calls return immediately unless blocking for that function is specifically requested by the application. To initialize the CS API for non-blocking I/O operations, the application must provide an event handler routine and pass a pointer to that routine to the CS API through the cs_init function call.

Each CS API function that communicates with Freeway or the ICP has a block_time parameter, in seconds, which determines the timeout for that function. A CS API function executes using non-blocking I/O if the application has registered an event handler routine and the call to the function is made with the block_time parameter set to zero. If the block_time parameter is set to something other than zero, the function will block until completion or until it times out. If the block_time parameter is set to –1, the function will block until completion.

When an application calls a CS API function using non-blocking I/O, the application must inspect the return value to determine if a CS API error occurred. If the function returns CS_NO_ERROR, an event will be generated to notify the application of the success or failure upon completion of the request to Freeway or the ICP. Events are generated only for functions called using non-blocking I/O with a block_time of zero or those called with a block_time greater than zero that have timed out.

Table 4–1 lists all of the CS API events and the ICP commands that generate the events. In some cases, events are generated specific to CS API function calls.

An important consideration in the design of code using non-blocking I/O is the fact that the event handler may receive the completion of an event before the call that generates the event completes.

Take, for example, the case of an attach. The design calls for calling the cs_attach function and using the returned client ID to initialize some information tables.

Table 4–1: Non-blocking I/O Events

Function	Event Class	Description or ICP Command
Generic (not directly associated with a CS API function call)		
	CS_ABORT	IABORT
	CS_AUTO_CONNECT	IAUTO
	CS_BADCID	Invalid client ID
	CS_BUF_OVERFLOW	DLI buffer overflow
	CS_CALL_TIMEOUT	CS API call has timed out
	CS_DEAD_SOCKET	TCP/IP socket connection lost
	CS_DLI_FATAL	Fatal DLI error
	CS_ERROR	IERROR
	CS_FILE_NOT_FOUND	Configuration file not found
	CS_FW_UNBOUND	Freeway has disconnected
	CS_HANGUP	IHANGUP
	CS_ICP_READY	ICP has completed the protocol download and is ready for use (this event always follows a CS_ICP_RESETTING event)
	CS_ICP_RESETTING	ICP has received a reset command and is downloading the protocol
	CS_INC_CALL	ICALL
	CS_INTERRUPT	IINT
	CS_INTERRUPT_ACK	IINTC
	CS_INVALID_BUFLEN	Invalid write buffer length
	CS_INVALID_CIRCUIT	Invalid circuit name (cs_attach)
	CS_INVALID_ICPHDR	Manager passed bad icphdr length
	CS_INVREQ	Invalid request for current state
	CS_LOST_CONN	ISTAFAIL, ISTAOK (after an IABORT)
	CS_MAX_UNACKS	Maximum unacknowledged writes; window closed
	CS_MEM_EXAUSTED	System memory pool is exhausted
	CS_NOBIND	Not bound
	CS_NO_ERROR	No error
	CS_NO_MEMORY	Insufficient memory to process request
	CS_NOT_ASYNC	DLI must be non-blocking I/O
	CS_NOT_INIT	cs_init not called yet
	CS_QUEUE_OVERFLOW	Event queue has filled and data may be lost

Table 4–1: Non-blocking I/O Events (*Cont'd*)

Function	Event Class	Description or ICP Command
	CS_READ_COMPLETE	There is a read pending; for manager CS API, this could be any command not covered by other events
	CS_REJECT	IREJECT
	CS_RESET	IRSET
	CS_RESET_ACK	IRSETC
	CS_STA_ERROR	IABORT or ISTAFAIL received
	CS_SVRERR	Severe error — write has timed out
	CS_SYS_RESOURCE	Unable to allocate resources
	CS_UNKNOWN_ERROR	Unknown error
	CS_WRITE_COMPLETE	Write through the ICP to the WAN has completed; the application may keep local counts of writes for each connection and decrement the count when this event is received
	CS_WRITE_FAILED	IFAILURE
cs_accept		
	CS_ACCEPT_HANGUP	IHANGUP
	CS_ACCEPT_REJECT	IREJECT
	CS_ACCEPT_SUCCESS	IACKNOWLEDGE
cs_attach		
	CS_ATTACH_FAILED	dlOpen failure
	CS_ATTACH_SUCCESS	dlOpen success
cs_bind		
	CS_BIND_FAILED	ICLOSE_SESSION
	CS_BIND_REJECT	IREJECT (pvc)
	CS_BIND_SUCCESS	IOPEN_SESSION
cs_connect		
	CS_CONN_HANGUP	IHANGUP
	CS_CONN_REJECT	IREJECT
	CS_CONN_SUCCESS	ICONNECT, IENABLE
	CS_CONN_TIMEOUT	ITIMOUT
cs_deregister		
	CS_DEREG_REJECT	IREJECT
	CS_DEREG_SUCCESS	IACKNOWLEDGE

Table 4–1: Non-blocking I/O Events (*Cont'd*)

Function	Event Class	Description or ICP Command
cs_detach		
	CS_DETACH_FAILED	dlClose failure
	CS_DETACH_SUCCESS	dlClose success
cs_disconnect		
	CS_DISCONN_REJECT	IREJECT
	CS_DISCONN_SUCCESS	ITONE, IHANGUP, IDISABLE
cs_redirect		
	CS_REDIR_HANGUP	IHANGUP
	CS_REDIR_REJECT	IREJECT
	CS_REDIR_SUCCESS	IACKNOWLEDGE
cs_refuse		
	CS_REFUSE_HANGUP	IHANGUP
	CS_REFUSE_REJECT	IREJECT
	CS_REFUSE_SUCCESS	ITONE
cs_register		
	CS_REG_REJECT	IREJECT
	CS_REG_SUCCESS	IACKNOWLEDGE
cs_reset		
	CS_RESET_HANGUP	IHANGUP
	CS_RESET_REJECT	IREJECT
	CS_RESET_SUCCESS	IRSETC
cs_unbind		
	CS_UNBIND_REJECT	IREJECT
	CS_UNBIND_SUCCESS	ICLOSE_SESSION

The following code segment illustrates this example:

```
client_id = cs_attach(...);
if (client_id)
{
     ... initialize tables, e.g., set the Service Access Point (SAP)
}
```

When the CS_ATTACH_SUCCESS event is received by the event handler, a bind will be issued using information from the client_id's information tables, such as the SAP.

The problem with this design is that CS_ATTACH_SUCCESS could be received by the event handler before the cs_attach function completes or before the client_id tables have been initialized.

The CS API contains the following two functions that provide event notification control for the user application:

cs_suspend_events	Suspends delivery of generated events
cs_resume_events	Resumes delivery of generated events

The previous code segment could be modified as follows:

```
cs_suspend_events();
client_id = cs_attach(...);
if (client_id)
{
     ... initialize tables, e.g., set the Service Access Point (SAP)
}
cs_resume_events(1);
```

The application is now assured that CS_ATTACH_SUCCESS will not be received by the event handler until after the tables have been initialized.

The CS API does not re-enter the event handler and therefore any function calls executed from within the user's event handler will automatically be protected from the above scenario.

4.2 Blocking I/O Operations

When using blocking I/O, the application must wait for or poll for return events. To initialize the CS API for using blocking I/O, the application passes a NULL pointer instead of declaring an event handler when calling the cs_init function. Even when using blocking I/O, functions allow the application to request non-blocking execution by specifying a block_time parameter of zero.

When an application calls a CS API function using blocking I/O, the application must inspect the return value to determine if a CS API error occurred. If the function returns CS_NO_ERROR, the requested action has been completed. Otherwise, the function return value specifies a timeout, an error, or an event. If an event was indicated (see the IS_EVENT macro in the cs_dfine.h file), the application must call cs_read and pass the event code to retrieve the associated data. Table 4–2 shows a list of function return values for CS API functions called using blocking I/O.

Table 4–2: Blocking I/O Function Return Values

Function	Return Value	ICP Command or Event
Generic (for more than one CS API function call)		
	CS_BADCID	Client ID invalid
	CS_BUF_OVERFLOW	DLI buffer overflow
	CS_CALL_TIMEOUT	Function block_time limit reached
	CS_DEAD_SOCKET	TCP/IP socket connection lost
	CS_DLI_FATAL	Fatal DLI error
	CS_ERROR	IERROR
	CS_FILE_NOT_FOUND	Configuration file not found
	CS_FW_UNBOUND	Freeway has disconnected
	CS_ICP_READY	ICP has completed the protocol download and is ready for use (this event always follows a CS_ICP_RESETTING event)
	CS_ICP_RESETTING	ICP has received a reset command and is downloading the protocol

Table 4–2: Blocking I/O Function Return Values (*Cont'd*)

Function	Return Value	ICP Command or Event
	CS_INTERRUPT	IINT
	CS_INVALID_BUFLEN	Invalid write buffer length
	CS_INVALID_CIRCUIT	Invalid circuit name (cs_attach)
	CS_INVALID_ICPHDR	Manager passed bad icphdr length
	CS_INVREQ	Invalid request for current state
	CS_LOST_CONN	ISTAFAIL, ISTAOK (after an IABORT)
	CS_MAX_UNACKS	Maximum unacknowledged writes; window closed
	CS_MEM_EXAUSTED	System memory pool is exhausted
	CS_NOBIND	Not bound
	CS_NO_ERROR	No error
	CS_NO_MEMORY	Insufficient memory to process request
	CS_NOT_ASYNC	DLI must be non-blocking I/O
	CS_NOT_INIT	The cs_init function must be called first
	CS_REJECT	IREJECT
	CS_RESET	IRSET
	CS_STA_ERROR	IERROR
	CS_SVRERR	Severe error — write has timed out
	CS_SYS_RESOURCE	Unable to allocate resources
	CS_UNKNOWN_ERROR	Unknown error
	CS_WRITE_FAILED	IFAILURE
cs_accept		
	CS_ACCEPT_HANGUP	IHANGUP
	CS_ACCEPT_REJECT	IREJECT
	CS_NO_ERROR	IACKNOWLEDGE
cs_attach		
	client ID number	dlOpen success
	CS_ATTACH_FAILED	dlOpen failure
	CS_INVALID_CIRCUIT	Named circuit not found

Table 4–2: Blocking I/O Function Return Values (*Cont'd*)

Function	Return Value	ICP Command or Event
cs_bind		
	CS_BIND_FAILED	ICLOSE_SESSION
	CS_BIND_REJECT	IREJECT (pvc)
	CS_NO_ERROR	IOPEN_SESSION
cs_connect or cs_connect_nb_remote		
	CS_CONN_HANGUP	IHANGUP
	CS_CONN_REJECT	IREJECT
	CS_CONN_TIMEOUT	ITIMOUT
	CS_NO_ERROR	ICONNECT, IENABLE
cs_deregister		
	CS_DEREG_REJECT	IREJECT
	CS_NO_ERROR	IACKNOWLEDGE
cs_detach		
	CS_DETACH_FAILED	dlClose failure
	CS_NO_ERROR	dlClose success
cs_disconnect		
	CS_DISCONN_REJECT	IREJECT
	CS_NO_ERROR	ITONE, IHANGUP, IDISABLE
cs_init		
	CS_INVALID_CIRCUIT	Configuration files problem
	CS_NO_ERROR	Normal initialization complete
	CS_NOT_ASYNC	TSI configuration must be non-blocking I/O
	CS_SYS_RESOURCE	Client system resources insufficient
cs_read		
	≥ 0	Data size
	< 0	Error
cs_redirect		
	CS_NO_ERROR	IACKNOWLEDGE
	CS_REDIR_HANGUP	IHANGUP
	CS_REDIR_REJECT	IREJECT

Table 4–2: Blocking I/O Function Return Values (*Cont'd*)

Function	Return Value	ICP Command or Event
cs_refuse		
	CS_NO_ERROR	ITONE
	CS_REFUSE_HANGUP	IHANGUP
	CS_REFUSE_REJECT	IREJECT
cs_register		
	CS_NO_ERROR	IACKNOWLEDGE
	CS_REG_REJECT	IREJECT
cs_reset		
	CS_NO_ERROR	IRSETC
	CS_RESET_HANGUP	IHANGUP
	CS_RESET_REJECT	IREJECT
cs_select		
	CS_NO_ERROR	N/A
cs_unbind		
	CS_NO_ERROR	ICLOSE_SESSION
	CS_UNBIND_REJECT	IREJECT
cs_write		
	CS_MAX_UNACKS	Flow control limit reached
	CS_WRITE_FAILED	IFAILURE

Since blocking I/O specifically excludes any event handler, the application must call the cs_select function to detect incoming data or indications of special conditions on the line, or to receive return events if the called function timed out.

For additional information on the cs_select function, see Section 5.4.3 on page 105.

4.3 Multitasking Operations

Some operating systems are non-preemptive multitasking and therefore the application must decide when to give up the CPU to allow lower-level functions to process I/O

reads and writes. The Freeway VxWorks system, for instance, may be operated with all tasks at the same priority level and time slicing disabled. The CS API provides a function for such operating systems. The cs_sleep function releases the CPU for a specified amount of time to allow other processes to access the CPU. Chapter 5 describes the cs_sleep function in more detail. CS API calls using blocking I/O automatically call the internal api_sleep function to give up the CPU.

4.4 Event-driven Program

An event-driven program is a program that reacts to events rather than loop and poll for work to perform. It allows the application programmer to concentrate on small pieces of code and not worry about coordination with other parts of the program. Not all applications lend themselves to exclusive event handling due to interactions with outside influences. Even in the worse case, however, some portion of the application can be placed in an event handler.

What is an event handler? An event handler is a function that is called by a lower-level subsystem, such as the CS API, to handle events that have taken place in the system. The most famous example of an event-driven program or system is MicroSoft Windows and all programs written to run under Windows.

The CS API provides the application programmer with the capability of writing event-driven code. In the following paragraphs, we will explore some simple code that implements an event handler for the X.25 CS API.

To set up your code to run using non-blocking I/O, you must provide an event handler (see Section 4.5 on page 69 for a description of the event handler's calling parameters):

```
void event_handler(int client_id, int event, int data_flag)
{
}
```

and pass the address of that handler to the cs_init function:

```
main()
{
     ...
     cs_init(”cs_config”, event_handler);
     ...
}
```

That is all that is needed to set up an event handler. Now, how do we activate the event handler and what code should be executed within the event handler?

The application must first call the cs_attach function to attach to Freeway, then the cs_bind function to bind to the ICP board. We need to place the cs_attach function call in the main program loop, but the cs_bind function call may be placed in the event handler. Let's assume that for this example we are going to attach five different clients. The following main loop code makes five attaches:

```
#include <cs_api.h>
#define NOT_ATTACHED    0
#define ATTACHED        1
#define BOUND           2
#define CONNECTED       3

int client_status[6] = {0,0,0,0,0,0};   /* client ID 0 is not used */

main()
{
     char *circuit[] = {"test1","test2","test3","test4","test5"};
     int client_id;
     ...
     cs_init("cs_config", event_handler);
     ...
     for (i = 0; i < 5; i++)
     {
           client_id = cs_attach(circuit[i], 0);
           if (client_id <= 0)
           {
                 printf("Unable to attach, error = %d\n", client_id);
                 ...
           }
     }
     ...
}
```

When an attach completes, we will want to update the client_id status and call the cs_bind function. The following code provides for handling successful and unsuccessful attaches:

```
void event_handler(int client_id, int event, int data_flag)
{
     switch(event)
     {
           case CS_ATTACH_SUCCESS:
                 client_status[client_id] = ATTACHED;
                 cs_bind(client_id, 0, 0);
                 break;

           case CS_ATTACH_FAILED:
                 printf("Attach failed for client %d\n",client_id);
                 break;
     }
}
```

The application programmer need only concern him or herself with what action to take if the attach succeeded or failed. In this case, a successful attach is followed by a bind and a failed attach results in a message to the screen.

The next step is to handle the case of a successful or failed bind. Again we will only display a message for a failure. For a successful bind we will register an incoming call filter. Our modified event handler code appears as shown below:

```
void event_handler(int client_id, int event, int data_flag)
{
     int ret_flag;
     char buf[80];
     struct cs_qos_st qos;

     switch(event)
     {
           case CS_ATTACH_SUCCESS:
                 client_status[client_id] = ATTACHED;
                 cs_bind(client_id, 0, 0);
                 break;

           case CS_ATTACH_FAILED:
                 printf("Attach failed for client %d\n",client_id);
                 break;

           case CS_BIND_SUCCESS:
                 client_status[client_id] = BOUND;
                 ...   /* set up call filter qos parameters here*/
                 cs_register(client_id, &qos, 0);
```

```
                  break;

            case CS_BIND_FAILED:
                  printf("Bind failure on client %d\n", client_id);
                  break;

            case CS_BIND_REJECT:
                  printf("Bind rejected on client %d\n", client_id);
                  if (data_flag)
                  {
                     cs_read(client_id, buf, sizeof(buf), &ret_flag, 0);
                     printf("%s\n",buf);
                  }
            break;
      }
}
```

Note that in the above event handler, we recovered the bind reject message and displayed it on the screen. The CS API event handler contains a data_flag parameter to inform the handler function if there is data associated with the event.

The next step is to handle an incoming call. The modified code below handles an incoming call by calling the cs_listen function, then the cs_accept function to accept the call:

```
void event_handler(int client_id, int event, int data_flag)
{
     int ret_flag;
     char buf[80];
     struct cs_listen_st ret_ind;
     struct cs_qos_st qos;

     switch(event)
     {
            case CS_ATTACH_SUCCESS:
                  client_status[client_id] = ATTACHED;
                  cs_bind(client_id, 0, 0);
                  break;

            case CS_ATTACH_FAILED:
                  printf("Attach failed for client %d\n",client_id);
                  break;

            case CS_BIND_SUCCESS:
                  client_status[client_id] = BOUND;
                  ...   /* set up call filter qos parameters here*/
                  cs_register(client_id, &qos, 0);
                  break;

            case CS_BIND_FAILED:
```

```
                printf("Bind failure on client %d\n", client_id);
                break;

          case CS_BIND_REJECT:
                printf("Bind rejected on client %d\n", client_id);
                if (data_flag)
                {
                   cs_read(client_id, buf, sizeof(buf), &ret_flag, 0);
                   printf("%s\n",buf);
                }
                break;

          case CS_INC_CALL:
                cs_listen(&client_id, 1, &ret_ind, &qos, 0);
                cs_accept(client_id, ret_ind.token, NULL, 0);
                cs_deregister(client_id, 0);
                break;
     }
}
```

In this case we are accepting the call. We could have refused or redirected the incoming call. We are also deregistering our call filter to avoid having additional incoming calls placed on hold while the initial call is active.

The next step is to handle the accept and deregister return events:

```
void event_handler(int client_id, int event, int data_flag)
{
     int ret_flag;
     char buf[80];
     struct cs_listen_st ret_ind;
     struct cs_qos_st qos;

     switch(event)
     {
          case CS_ATTACH_SUCCESS:
                client_status[client_id] = ATTACHED;
                cs_bind(client_id, 0, 0);
                break;

          case CS_ATTACH_FAILED:
                printf("Attach failed for client %d\n",client_id);
                break;

          case CS_BIND_SUCCESS:
                client_status[client_id] = BOUND;
                ...   /* set up call filter qos parameters here*/
                cs_register(client_id, &qos, 0);
                break;
```

```
                case CS_BIND_FAILED:
                      printf("Bind failure on client %d\n", client_id);
                      break;

                case CS_BIND_REJECT:
                      printf("Bind rejected on client %d\n", client_id);
                      if (data_flag)
                      {
                         cs_read(client_id, buf, sizeof(buf), &ret_flag, 0);
                         printf("%s\n",buf);
                      }
                      break;

                case CS_INC_CALL:
                      cs_listen(&client_id, 1, &ret_ind, &qos, 0);
                      cs_accept(client_id, ret_ind.token, NULL, 0);
                      cs_deregister(client_id, 0);
                      break;

                case CS_ACCEPT_SUCCESS:
                      client_status[client_id] = CONNECTED;
                      break;

                case CS_ACCEPT_REJECT:
                case CS_ACCEPT_HANGUP:
                      printf("Accept failed on client %d\n", client_id);
                      if (data_flag)
                      {
                         cs_read(client_id, buf, sizeof(buf), &ret_flag, 0);
                         printf("%s\n",buf);
                      }
                      break;

                case CS_DEREG_SUCCESS:
                      break;

                case CS_DEREG_REJECT:
                      printf("Deregister failed on client %d\n", client_id);
                      if (data_flag)
                      {
                         cs_read(client_id, buf, sizeof(buf), &ret_flag, 0);
                         printf("%s\n",buf);
                      }
                      break;
          }
     }
```

As can be seen, each event is handled with a few simple lines of code. For more complicated situations, the handler could call another function to process the event.

4.5 Event Handler

The CS API user-defined event handler is called to notify the application program of a non-blocking I/O event. The function's calling parameters are:

```
event_handler(int client_id, int event, int data_flag)
```

where:

- client_id is the ID number returned by cs_attach()
- event is the non-blocking I/O event (see cs_dfine.h)
- data_flag is the number of data bytes associated with the event

If the data_flag is other than zero, the application must call cs_read and pass the event parameter to get the event's data buffer. The cs_listen or cs_connect_nb_remote function can also be used for CS_INC_CALL or connection-related events.

Caution

Failure to read queued data could eventually exhaust the available memory pool.

Chapter

5 CS API Reference

This chapter provides a detailed reference for the call service application program interface (CS API) described in general terms in Chapter 1, and in more detail in Chapter 2, Chapter 3, and Chapter 4. Table 5–1 shows the supported CS API requests grouped functionally. Beginning with Section 5.1.1 on page 75, each CS API function is described in detail. The definitions of the symbolically named values in this chapter are defined in include files shown in Appendix A.

Table 5–1: CS API Function Groups

Group	Functions
Connection preparation	cs_init, cs_attach, cs_bind
Active connection	cs_connect, cs_connect_nb_remote
Passive connection	cs_register, cs_listen, cs_accept, cs_redirect, cs_refuse
Data transfer	cs_read, cs_reset, cs_select, cs_write
Connection shutdown	cs_deregister, cs_disconnect, cs_unbind, cs_detach, cs_terminate
Miscellaneous	cs_sperror, cs_sleep, cs_config, cs_getpid, debuglog, cs_suicide, cs_suspend_events, cs_resume_events, cs_gen_event

The CS API itself is provided as a C library to be linked with the application programs. An include file (cs_api.h) provides the compile-time definitions needed by those programs. Section A.1 on page 144 gives the source file for the cs_api.h include file.

Most library functions return an integer value to the calling application. This integer value can be compared to the symbolic values for CS API errors shown in Table 5–2, which are defined in the cserrno.h include file (see Section A.4 on page 151). Refer back

to Table 4–2 on page 59 for a list of function return values for CS API functions called using blocking I/O (Table 4–1 on page 55 lists the non-blocking I/O events).

In some instances, the CS API functions can return an error code from the TSI or DLI. Refer to the *Freeway Transport Subsystem Interface Reference Guide* and the *Freeway Data Link Interface Reference Guide* for the TSI and DLI error codes. Applications can use the cs_sperror function for error reporting, which returns a pointer to the error text string. See Section 5.6.1 on page 119.

Table 5–2: CS API Errors Defined in cserrno.h Include File

CS API Error	Description
CS_BADCID	The client ID is invalid.
CS_CALL_TIMEOUT	The CS API call has timed out. In a non-blocking I/O application, the requested action may still complete later and be reported as a non-blocking I/O event. The application should increase its block_time.
CS_FILE_NOT_FOUND	Configuration file not found
CS_INVALID_BUFLEN	The cs_write function cannot write a buffer larger than that allowed by the DLI.
CS_INVALID_CIRCUIT	Invalid circuit name. The cs_init function could not access one or more of the run-time configuration files or the cs_attach function did not find the circuit name in the CS API configuration file previously specified in the cs_init function.
CS_INVALID_ICPHDR	The MANAGER_API application passed an invalid ICP header length.
CS_INVREQ	Invalid request for current state.
CS_MAX_UNACKS	The maximum number of unacknowledged writes has been reached; window closed. Look for a CS_WRITE_COMPLETE event and try again later.
CS_NOBIND	Not bound; call the cs_bind function.
CS_NO_ERROR	No error (guaranteed to be 0).
CS_NO_MEMORY	Cannot allocate memory. System memory allocation for reads, writes, or storage of completed reads has been exhausted.
CS_NOT_ASYNC	The DLI is not configured for non-blocking I/O. The cs_init function found that the DLI configuration is for blocking I/O only, while CS API requires non-blocking I/O.

Table 5–2: CS API Errors Defined in cserrno.h Include File (*Cont'd*)

CS API Error	Description
CS_NOT_INIT	The cs_init function has not been called yet. It must be called before any other CS API function.
CS_STA_ERROR	The CS API has received an IABORT and has not yet received a corresponding ISTAOK, or has received ISTAFAIL.
CS_SVRERR	Severe error — write has timed out
CS_SYS_RESOURCE	The cs_init or cs_terminate function has encountered a system error in allocating or releasing system resources (signal, semaphore, and so on).
CS_UNKNOWN_ERROR	Unknown error

Each of the following subsections contains a functional description of the routine, including descriptions of its operations using non-blocking I/O, an arguments section which provides a brief description of the calling parameters, error return values, and packet exchanges between the CS API (on the host computer) and the ICP (on Freeway).

You do not need to understand the packet exchanges with the ICP to use the CS API effectively. The information is provided to document how the CS API handles the Freeway X.25 low-level interface.

Packet exchange sections are formatted as follows:

```
<command mnemonic>
-----------------> (is an outgoing packet)

<command mnemonic>
<----------------- (is an incoming packet)
```

Incoming packets are annotated with two fields:

```
<command mnemonic>
<----------------- <error return>   <event class>
```

The following commands may be received at any time and are not included in the description for each CS API function:

```
Command                 Error Return    Event

IABORT                                                  |
<----------------       CS_NO_ERROR     CS_ABORT        | packet
ISTAOK                                                  | pairs
<----------------       CS_LOST_CONN    CS_LOST_CONN    |


ISTAFAIL (PVC only)
<----------------       CS_LOST_CONN    CS_LOST_CONN
IERROR
<----------------       CS_ERROR        CS_ERROR
```

(The X.25 cause code is contained in the cs_read return buffer for the CS_ERROR event. The cause/diagnostic codes are described in Appendix C.)

The following events are not associated with any X.25 command and can occur at any time:

CS_DEAD_SOCKET The TCP/IP socket connection has been lost.

CS_ICP_RESETTING The ICP is in a reset mode (someone issued a dlControl() command).

CS_ICP_READY The ICP is now ready and the application must close the connection by calling cs_detach().

When the ICP has been reset, there is no need for the application to disconnect or unbind, only to call cs_detach().

Since blocking I/O applications do not have event handlers, some of the special events, such as the ones above, can be detected by calling the cs_select function and examining the indication array. For additional information on the cs_select function, see Section 5.4.3 on page 105.

5.1 Connection Preparation

The following CS API functions are called for Freeway preparation services.

5.1.1 cs_init

The cs_init function initializes the CS API and lower-level support systems. It must be called only once per application, and must be called before any other CS API function. It calls the DLI subsystem to get configuration data, allocates system resources, and registers the user's event handler.

Synopsis

```
int cs_init(config_file, int_func)
char *config_file;
void (*int_func)(int,int,int);
```

Arguments

config_file	CS API configuration file name
int_func	User event handler or NULL if using blocking I/O

Return Value

On success, this function returns zero. If the DLI is configured for blocking I/O, the CS_NOT_ASYNC error is returned. On failure, this function returns a number less than zero. (The error code is returned by the DLI.)

5.1.2 cs_attach

The cs_attach function attaches the application to the specified ICP. It reads the CS API configuration file to retrieve the service access point, PVC address, and DLI connection name, then opens a connection to Freeway.

When using non-blocking I/O, the cs_attach function returns the client ID or a negative error code. If the open was successful, the CS_ATTACH_SUCCESS event is queued. If an error occurred, the CS_ATTACH_FAILED event is queued.

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_attach(circuit_id, block_time)
char *circuit_id;
int block_time;
```

Arguments

circuit_id	Circuit ID name as listed in the CS API configuration file
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns the connection identification value (client_id) that is used to identify this attachment to all future CS API requests.

On failure, this function returns a number less than or equal to zero.

5.1.3 cs_bind

The cs_bind function binds an application to an X.25 protocol service. When using non-blocking I/O and the bind call completes:

- the CS_BIND_SUCCESS event is queued if the call was successful
- the CS_BIND_FAILED event is queued if the call was unsuccessful
- the CS_BIND_REJECT event is queued if a PVC IREJECT command was received

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_bind(client_id, sap, block_time)
int client_id;
int sap;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
sap	Service access point (if 0, the default from the CS API configuration file is used)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Packet Exchanges, Error Returns, and Events Generated

For HDLC and X.25 SVC:

```
Command                 Error Return    Event

HOPEN_SESSION
---------------->
IOPEN_SESSION
<----------------       CS_NO_ERROR     CS_BIND_SUCCESS

          or

ICLOSE_SESSION
<----------------       CS_NOBIND       CS_BIND_FAILED
```

For X.25 PVC, when IOPEN_SESSION is received:

```
Command                 Error Return    Event

HOPEN_PVC
---------------->
IOPEN_PVC
<----------------       CS_NO_ERROR     CS_BIND_SUCCESS

          or

IREJECT
<----------------       CS_SVRERR       CS_BIND_REJECT
```

5.2 Active Connection Handling

The following CS API functions support applications that actively establish outgoing calls. For information on waiting passively for incoming calls, see Section 5.3 on page 86.

5.2.1 cs_connect

The cs_connect function requests an X.25 connection. When using non-blocking I/O and the connection completes:

- the CS_CONN_SUCCESS event is queued if the connection was successful
- the CS_CONN_REJECT event is queued if an IREJECT command was received
- the CS_CONN_HANGUP event is queued if an IHANGUP command was received
- the CS_CONN_TIME event is queued if an ITIMOUT command was received

When using blocking I/O, this function returns CS_NO_ERROR if the connection is established before the specified block_time expires. If this function returns CS_CALL_TIMEOUT, the application must call cs_connect_nb_remote to check for completion of the connection request.

Synopsis

```
int cs_connect(client_id, dest, dest_length, qos, ret_qos,
               block_time)
int client_id;
char *dest;
int dest_length;
struct cs_qos_st *qos;
struct cs_qos_st *ret_qos;
int block_time;
```

Arguments

client_id Connection ID from the cs_attach function

dest	X.25 address (called DTE address for X.25) expressed as an ASCII string of decimal digits 0–9
dest_length	Length of dest in bytes
qos	Quality of service functions (NULL = none)
ret_qos	Returned call service functions (NULL = none)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

The qos parameter is optional. The structure cs_qos_st (defined in the cs_struc.h include file) is used to specify quality of service. The item_list field must either be zero, or must point to a byte array containing call service specifications selected from the list below.

HF_BLCLUS	Bilateral closed user group selection facility
HF_CLLED	Called DTE address; can be specified in qos only if the dest and dest_length parameters are NULL
HF_CLLNG	Calling DTE address
HF_CLUSG	Closed user group selection facility
HF_CLUSGOAS	Closed user group with outgoing access selection facility
HF_D_BIT_SUPPORT	D-bit support request/indication
HF_FASNR	Fast select facility (unrestricted)

HF_FASR	Fast select facility with restrictions
HF_NONSTD	Non-standard user facilities
HF_NWUSID	Network user identification facility
HF_PDSIZE	Packet data size facility
HF_PRIORITY	Virtual circuit local priority on Freeway
HF_PWSIZE	Packet window size facility
HF_RPOA	RPOA selection facility
HF_RQCRGIN	Request charging information facility
HF_RVFC	Reverse charging facility
HF_THRUCLASS	Throughput class facility
HF_TRDLYSEL	Transit delay selection and indication facility
HF_USER	User data facility

Each item_list specification contains from one to three sub-fields: a function code byte, an item size indicator byte, and a variable-length item data field. You must specify the sub-fields contiguously in the order given, and you must omit sub-fields not listed for a given item from the specification. Refer to Section 5.7 for a complete description of these specifications, including the sub-fields.

The ret_qos parameter is also optional. The structure cs_qos_st (defined in the cs_struc.h include file) is used to receive the actual negotiated quality of service. The item_list field must point to a byte array allocated by the application to receive call service specifications from the list given above for the qos parameter. The application uses the max_length field to specify the size of the item_list buffer. The CS API uses the tot_length field to return the length of the portion of the item_list buffer actually used.

Under special circumstances, the DTE to which the call was presented might not be the DTE originally specified as the called DTE. In this case, the called line address modified notification facility (HF_CLDADMOD) ret_qos parameter might be present.

The block_time parameter specifies the number of seconds the function may wait for the connection request to be granted (or rejected). If a block_time of zero is specified, or if a non-zero block time is specified and the request times out, the application must issue the cs_connect_nb_remote function to complete the connection request. When using non-blocking I/O, the application may wait for a CS_CONN_SUCCESS event before invoking cs_connect_nb_remote.

Packet Exchanges, Error Returns, and Events Generated

For X.25:

```
Command                 Error Return    Event

HCALL
----------------->
ICONNECT                                                |
<-----------------      CS_NO_ERROR     CS_CONN_SUCCESS | packet
HFLOW_ADJUST                                            | pairs
----------------->                                      |

          or

IREJECT
<-----------------      CS_SVRERR       CS_CONN_REJECT

          or

IHANGUP
<-----------------      CS_LOST_CONN    CS_CONN_HANGUP
```

For HDLC:

```
Command                 Error Return    Event

HENABLE
----------------->
IENABLE
```

```
<----------------       CS_NO_ERROR     CS_CONN_SUCCESS
```

or

```
IREJECT
<----------------       CS_SVRERR       CS_CONN_REJECT
```

or

```
Command                 Error Return    Event

ITIMOUT
<----------------
HDISABLE                                                |
---------------->                                       | packet
IDISABLE                                                | pairs
<----------------       CS_SVRERR       CS_CONN_TIMEOUT |
```

5.2.2 cs_connect_nb_remote

The cs_connect_nb_remote function completes the non-blocking X.25 connection request. This function returns the connection data for a previous cs_connect request that did not complete within the specified block_time.

Synopsis

```
int cs_connect_nb_remote(client_id, ret_qos, block_time)
int client_id;
struct cs_qos_st *ret_qos;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
ret_qos	Returned call service functions (NULL = none)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

The ret_qos parameter is optional. The structure cs_qos_st (defined in the cs_struc.h include file) is used to receive the actual negotiated quality of service. The item_list field must point to a byte array allocated by the application to receive call service specifications from the list given below. The application uses the max_length field to specify the size of the item_list buffer, and the CS API uses the tot_length field to return the length of the portion of the item_list buffer actually used.

HF_CLLED	Called DTE address

HF_CLLNG	Calling DTE address
HF_D_BIT_SUPPORT	D-bit support request/indication
HF_NONSTD	Non-standard user facilities
HF_PDSIZE	Packet data size facility
HF_PRIORITY	Virtual circuit local priority on Freeway
HF_PWSIZE	Packet window size facility
HF_THRUCLASS	Throughput class facility
HF_USER	User data facility

Each item_list specification contains from one to three sub-fields: a function code byte, an item size indicator byte, and a variable-length item data field. Refer to Section 5.7 for a complete description of these specifications, including the sub-fields.

Under special circumstances, the DTE to which the call was presented might not be the DTE originally specified as the called DTE. In this case, the called line address modified notification facility (HF_CLDADMOD) ret_qos parameter might be present.

5.3 Passive Connection Handling

The following CS API functions support applications that wait passively for incoming call indications. For information on the active establishment of outgoing calls, see Section 5.2 on page 79.

5.3.1 cs_register

The cs_register function registers the application as an incoming connection handler. When using non-blocking I/O, and the register call completes:

- the CS_REG_SUCCESS event is queued if the call was successful
- the CS_REG_REJECT event is queued if an IREJECT command was received

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_register(client_id, qos, block_time)
int client_id;
struct cs_qos_st *qos;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
qos	Quality of service functions for incoming call filter (NULL = none)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

The X.25 protocol service adds an entry into its connection handler list describing this client ID and the incoming call filter; thereafter, if the application has issued the cs_listen request, the X.25 protocol service sends the application an indication of any incoming connection indication that matches the client ID's filter.

Note

The application cannot register more than one incoming connection handler for each bound client_id; to alter the filter in an incoming connection handler currently registered for a given client_id, the application must first issue a cs_deregister request for that client_id before issuing another cs_register request for that client_id.

The qos parameter is optional. The structure cs_qos_st (defined in the cs_struc.h include file) is used to specify a filter giving the characteristics of incoming connection indications in which the application is interested. The item_list field must point to a byte array containing filter specifications selected from the list given below.

HF_DTE_REMOTE	ICF remote (calling) DTE address
HF_DTE_SA_HIGH	ICF local (called) DTE subaddress (high)
HF_DTE_SA_LOW	ICF local (called) DTE subaddress (low)
HF_ICF_CALLBUSY	An incoming call filter configuration parameter internal to Freeway.
HF_ICF_PRIORITY	An incoming call filter priority level internal to Freeway

HF_USER	User data match facility
HF_USER_MASK	User data mask for the incoming call filter

Each item_list specification contains from one to three sub-fields: a function code byte, an item size indicator byte, and a variable-length item data field. You must specify the sub-fields contiguously in the order given, and you must omit sub-fields not listed for a given item from the specification. Refer to Section 5.7 for a complete description of these specifications, including the sub-fields.

When an incoming connection request is received by the X.25 protocol service, it compares the characteristics of that connection indication to the filter specifications of this function; if all characteristics match and the application is currently listening, the service sends the connection indication to that application, allowing the application to accept, reject, or redirect the connection indication as desired.

After a CS API application accepts an incoming call, and if the HF_ICF_CALLBUSY parameter was not used to configure call redirection, Freeway places all additional incoming calls for that client ID on hold until one of the following events occurs:

- The current call is cleared
- The network DCE cancels the incoming call
- The CS API application issues a cs_deregister request

Calls can be placed on hold even if they match the registered incoming call handler profile for other CS API client IDs. After the currently active virtual circuit is disconnected, the application simply issues a cs_listen request to get the next incoming call.

If the CS API application does not want additional calls for a client ID to be placed on hold after accepting an incoming call, the application may issue a cs_deregister request to delete its incoming call handler profile from the X.25 protocol. After the currently active virtual circuit is disconnected, the application would then issue a cs_register request before issuing a cs_listen request. However, if the application uses the

HF_ICF_CALLBUSY quality of service parameter to configure the incoming call filter to redirect calls when busy, then the application need not call cs_deregister to avoid placing calls on hold.

For more information on handling incoming connection indications, see Section 1.5.2 on page 21.

Packet Exchanges, Error Returns, and Events Generated

Command	Error Return	Event
HREG_ICF ----------> IACKNOWLEDGE <----------	CS_NO_ERROR	CS_REG_SUCCESS
or		
IREJECT <----------	CS_SVRERR	CS_REG_REJECT

5.3.2 cs_listen

The cs_listen function waits for an incoming connection indication. When using non-blocking I/O, this function returns the connection data if called after notification of a CS_INC_CALL or CS_AUTO_CONNECT event for a specified client_id. When using blocking I/O, this function is used to check for (or wait for) an incoming call.

Synopsis

```
int cs_listen(client_array, client_length, ret_ind, ret_qos,
              block_time)
int *client_array;
int client_length;
struct cs_listen_st *ret_ind;
struct cs_qos_st *ret_qos;
int block_time;
```

Arguments

client_array	Array of connection IDs from the cs_attach function
client_length	Number of client IDs in client_array
ret_ind	Returned call indication parameters containing client ID, return token, and connection event
ret_qos	Returned call service functions (NULL = none)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

When the X.25 protocol service receives an incoming connection indication that is matched to one of this application's entries in the connection handling list, the service completes the connection and notifies the application by returning a function value of zero.

The ret_ind parameter contains information on the client ID from the client_array parameter that completed. The ret_ind structure (defined in the cs_struct.h include file) contains the following fields:

client client ID number

token token value to be returned in any cs_accept, cs_redirect, or cs_refuse call

state type of event, CS_INC_CALL or CS_AUTO_CONNECT

The ret_qos parameter is optional. The structure cs_qos_st (defined in the cs_struc.h include file) is used to receive the quality of service proposed by the remote DTE. The item_list field must point to a byte array allocated by the application to receive call service specifications from the list given below. The application uses the max_length field to specify the size of the item_list buffer. The CS API uses the tot_length field to return the length of the portion of the item_list buffer actually used.

HF_BLCLUS Bilateral closed user group selection facility

HF_CLLED Called DTE address

HF_CLLNG Calling DTE address

HF_CLREDR Call redirection or deflection notification facility

HF_CLUSG Closed user group selection facility

HF_CLUSGOAS Closed user group with outgoing access selection facility

HF_D_BIT_SUPPORT	D-bit support request/indication
HF_FASNR	Fast select facility (unrestricted)
HF_FASR	Fast select facility with restrictions
HF_NONSTD	Non-standard user facilities
HF_PDSIZE	Packet data size facility
HF_PRIORITY	Virtual circuit local priority on Freeway
HF_PWSIZE	Packet window size facility
HF_RVFC	Reverse charging facility
HF_THRUCLASS	Throughput class facility
HF_TRDLYSEL	Transit delay selection and indication facility
HF_USER	User data facility

Each item_list specification contains from one to three sub-fields: a function code byte, an item size indicator byte, and a variable-length item data field. Refer to Section 5.7 for a complete description of these specifications, including the sub-fields.

The application uses the block_time parameter to specify the maximum time for the function to wait for the connection indication. If the wait time expires with no connection indication, the CS_CALL_TIMEOUT error is returned. The application can issue the cs_listen request again after each return of the CS_CALL_TIMEOUT error.

For more information on handling incoming connection indications, see Section 1.5.2 on page 21.

5.3.3 cs_accept

The cs_accept function accepts an incoming X.25 connection request. When using non-blocking I/O, and the connection completes:

- the CS_ACCEPT_SUCCESS event is queued if the connection was successful
- the CS_ACCEPT_REJECT event is queued if an IREJECT command was received
- the CS_ACCEPT_HANGUP event is queued if an IHANGUP command was received

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_accept(client_id, token, qos, block_time)
int client_id;
int token;
struct cs_qos_st *qos;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
token	Connection token from the cs_listen function
qos	Quality of service functions (NULL = none)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number is less than zero.

Note

The value passed for the token parameter was originally obtained from the cs_listen function and serves to uniquely identify the incoming connection indication. Upon completion of this function, the application can begin transferring data over the established connection with the remote DTE.

The qos parameter is optional. The structure cs_qos_st (defined in the cs_struc.h include file) is used to specify quality of service. The item_list field must either be zero, or must point to a byte array containing call service facilities selected from the list given below.

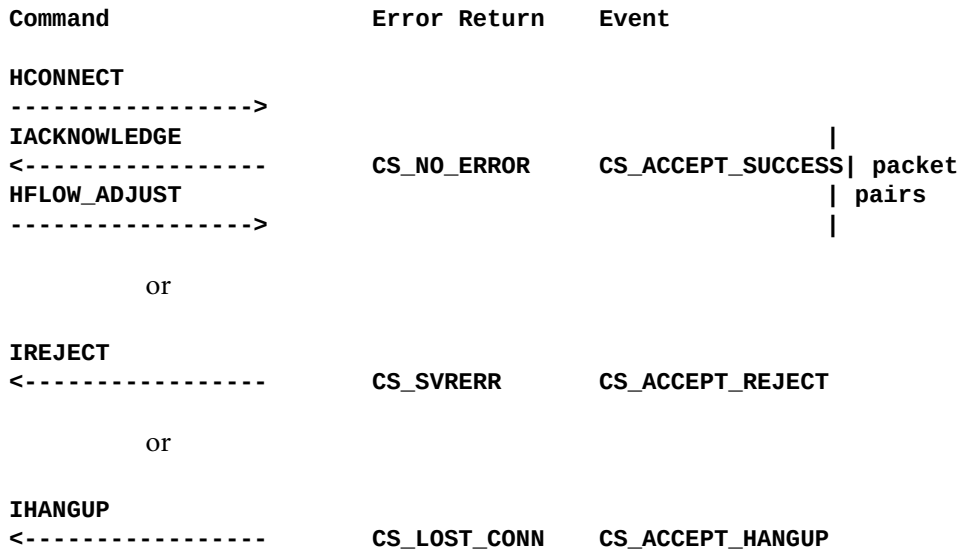
HF_CLLED	Called DTE address
HF_CLLNG	Calling DTE address
HF_D_BIT_SUPPORT	D-bit support request/indication
HF_NONSTD	Non-standard user facilities
HF_NWUSID	Network user identification facility
HF_PDSIZE	Packet data size facility
HF_PRIORITY	Virtual circuit local priority on Freeway
HF_PWSIZE	Packet window size facility
HF_RQCRGIN	Request charging information facility
HF_THRUCLASS	Throughput class facility
HF_USER	User data facility

Each item_list specification contains from one to three sub-fields: a function code byte, an item size indicator byte, and a variable-length item data field. You must specify the sub-fields contiguously in the order given, and you must omit sub-fields not listed for a

given item from the specification. Refer to Section 5.7 for a complete description of these specifications, including the sub-fields.

Under special circumstances, the DTE to which the call was presented might not be the DTE originally specified as the called DTE. In this case, the called line address modified notification facility (HF_CLDADMOD) qos parameter can be used in addition to those listed above.

Packet Exchanges, Error Returns, and Events Generated



5.3.4 cs_redirect

The cs_redirect function redirects an incoming X.25 connection request to another connection. When using non-blocking I/O, and the redirect call completes:

- the CS_REDIR_SUCCESS event is queued if the call was successful
- the CS_REDIR_REJECT event is queued if an IREJECT command was received
- the CS_REDIR_HANGUP event is queued if an IHANGUP command was received

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_redirect(client_id, token, qos, block_time)
int client_id;
int token;
struct cs_qos_st *qos;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
token	Connection token from the cs_listen function
qos	Quality of service functions (NULL = none)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

The application issuing the cs_redirect request was presumably informed of the incoming connection indication when it issued the cs_listen request, but has since determined that it does not want to handle the connection. The token parameter (originally obtained from the cs_listen response) identifies the incoming connection indication to be re-routed; the X.25 protocol service scans its connection handler list in an attempt to find another application to inform of this connection.

It is possible for this application to receive an indication of this same incoming connection in the future, although it might not have the same token value at that time. This is because the X.25 protocol service holds the connection indication internally if no application accepted or rejected the connection indication previously; changes to the registered connection handler list might cause the service to re-scan the list, at which point it might again inform the current application of the incoming connection indication.

The reserved qos parameter is provided for future expansion of cs_redirect request capabilities. Currently, the qos parameter must be NULL.

Packet Exchanges, Error Returns, and Events Generated

Command	Error Return	Event
HREDIRECT ---------->		
IACKNOWLEDGE <----------	CS_NO_ERROR	CS_REDIR_SUCCESS
or		
IREJECT <----------	CS_SVRERR	CS_REDIR_REJECT
or		
IHANGUP <----------	CS_LOST_CONN	CS_REDIR_HANGUP

5.3.5 cs_refuse

The cs_refuse function refuses an incoming X.25 connection request. When using non-blocking I/O, and the refuse call completes:

- the CS_REFUSE_SUCCESS event is queued if the call was successful
- the CS_REFUSE_REJECT event is queued if an IREJECT command was received
- the CS_REFUSE_HANGUP event is queued if an IHANGUP command was received

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_refuse(client_id, token, qos, block_time)
int client_id;
int token;
struct cs_qos_st *qos;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
token	Connection token from the cs_listen function
qos	Quality of service functions (NULL = none)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

The token parameter identifies that connection where the token value was returned by the cs_listen function.

The qos parameter is optional. The structure cs_qos_st (defined in the cs_struc.h include file) is used to convey the reason (cause and diagnostic codes) for refusing this connection and optional user data in the resulting X.25 clear request. The item_list field must point to a byte array containing the cause code and diagnostic code facilities and optional user data facility selected from the list given below.

HF_CAUSE	Cause code facility
HF_CLDEFLECT	Call deflection selection facility
HF_CLLED	Called DTE address
HF_CLLNG	Calling DTE address
HF_DIAG	Diagnostic code
HF_NONSTD	Non-standard user facilities
HF_USER	User data facility

Each item_list specification contains from one to three sub-fields: a function code byte, an item size indicator byte, and a variable-length item data field. You must specify the sub-fields contiguously in the order given, and you must omit sub-fields not listed for a given item from the specification. Refer to Section 5.7 for a complete description of these specifications, including the sub-fields.

Under special circumstances, the DTE to which the call was presented might not be the DTE originally specified as the called DTE. In this case, the called line address modified notification facility (HF_CLDADMOD) qos parameter can be used in addition to those listed above.

Packet Exchanges, Error Returns, and Events Generated

```
Command                  Error Return    Event

HHANGUP
----------------->
ITONE
<-----------------       CS_NO_ERROR     CS_REFUSE_SUCCESS
```

or

```
IREJECT
<-----------------       CS_SVRERR       CS_REFUSE_REJECT
```

or

```
IHANGUP
<-----------------       CS_LOST_CONN    CS_REFUSE_HANGUP
```

5.4 Data Transfer

The following CS API functions are called for data transfer.

5.4.1 cs_read

The cs_read function reads data from the ICP.

Synopsis

```
int cs_read(client_id, buf, buf_length, event, ret_flags,
            block_time)
int client_id;
char *buf;
int buf_length;
int event;
int *ret_flags;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
buf	Data read from the ICP
buf_length	Length of receive buffer in bytes
event	Event flag received in the event handler, an error code returned by another function call, or indication returned by cs_select
ret_flags	Returned special qualifier flag
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns the number of bytes received in buf. On failure, this function returns a number less than zero.

5.4.2 cs_reset

The cs_reset function resets an X.25 connection. When using non-blocking I/O, and the reset call completes:

- the CS_RESET_SUCCESS event is queued if the call was successful
- the CS_RESET_REJECT event is queued if an IREJECT command was received
- the CS_RESET_HANGUP event is queued if an IHANGUP command was received

When using blocking I/O, this function returns CS_NO_ERROR if the reset occurs before the specified block_time expires. If this function returns CS_CALL_TIMEOUT, the application must call cs_select to detect when a CS_INDX25RSET_ACK or CS_INDX25RSET indication completes the reset.

Synopsis

```
int cs_reset(client_id, qos, block_time)
int client_id;
struct cs_qos_st *qos;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
qos	Quality of service functions (NULL = none)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

If this function times out and no event handler routine exists, the application must use the cs_select function to re-check for the reset confirmation.

The qos parameter is optional. The structure cs_qos_st (defined in the cs_struc.h include file) is used to convey the reason (cause and diagnostic codes) for resetting this connection and the new virtual circuit priority following reset completion. The item_list field must point to a byte array containing cause code, diagnostic code and circuit priority specifications selected from the list given below.

HF_CAUSE	Cause code facility
HF_DIAG	Diagnostic code
HF_PRIORITY	Virtual circuit local priority on Freeway

Each item_list specification contains from one to three sub-fields: a function code byte, an item size indicator byte, and a variable-length item data field. You must specify the sub-fields contiguously in the order given, and you must omit sub-fields not listed for a given item from the specification. Refer to Section 5.7 for a complete description of these specifications, including the sub-fields.

Packet Exchanges, Error Returns, and Events Generated

For X.25:

```
Command                 Error Return    Event

HRSET
----------------->
IRSETC
<-----------------      CS_NO_ERROR     CS_RESET_SUCCESS

          or

IREJECT
<-----------------      CS_SVRERR       CS_RESET_REJECT
```

or

```
IHANGUP
<-----------------       CS_LOST_CONN    CS_RESET_HANGUP
```

or

```
Command                  Error Return    Event

IRESET
<-----------------       reset collision CS_RESET
                         (completes cs_reset)
```

For HDLC:

```
Command                  Error Return    Event

HINIT_SLP
------------------>
IGLITCH
<-----------------       CS_NO_ERROR     CS_RESET_SUCCESS
```

5.4.3 cs_select

This function is used only with blocking I/O. The cs_select function reports the availability of data or one of the following indications:

- CS_INDNOCONN (or CS_LOST_CONN)
- CS_INDX25OOB (or CS_INTERRUPT)
- CS_INDX2500B_ACK (or CS_INTERRUPT_ACK)
- CS_INDX25RSET (or CS_RESET)
- CS_INDX25RSET_ACK (or CS_RESET_ACK)

Synopsis

```
int cs_select(client_array, client_length, read_array, ind_array,
              block_time)
int *client_array;
int client_length;
int *read_array;
int *ind_array;
int block_time;
```

Arguments

client_array	Array of client IDs to search
client_length	Length of the client array
read_array	Return array of read completion flags
ind_array	Return array of indication flags
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

The returned read_array contains the number of read completions for each client_id. The returned ind_array contains zero or an indication type. There may be more than one indication type in the read completion queue, but only one is reported.

The cs_select function is similar to the UNIX, VMS, and Windows NT select function calls.

Example of cs_select Usage

Prior to calling cs_select as follows:

```
cs_select(client_array, 3, read_array, ind_array, 0);
```

The arrays look like the following (it is assumed that the user has filled in the client_array with the ID numbers returned by the cs_attach calls.

```
                  +--------+--------+--------+
client_array      |   1    |   2    |   3    |
                  +--------+--------+--------+

                  +--------+--------+--------+
read_array        |   0    |   0    |   0    |
                  +--------+--------+--------+

                  +--------+--------+--------+
ind_array         |   0    |   0    |   0    |
                  +--------+--------+--------+
```

If client ID 1 has two CS_READ_COMPLETE events and client ID 3 has a CS_HANGUP (–29046) and a CS_LOST_CONN (–29006) event, then the return arrays would look like the following:

```
              +--------+--------+--------+
client_array  |    1   |   2    |   3    |
              +--------+--------+--------+

              +--------+--------+--------+
read_array    |    2   |   0    |   0    |
              +--------+--------+--------+

              +--------+--------+--------+
ind_array     |    0   |   0    | -29046 | <== returns top event only
              +--------+--------+--------+
```

The application would call cs_read twice with the event parameter set to CS_READ_COMPLETE for client ID 1 and once with the event parameter set to –29046 for client ID 3. The next cs_select call would return the –29006 event for client ID 3.

5.4.4 cs_write

The cs_write function writes data to the ICP.

Synopsis

```
int cs_write(client_id, buf, buf_length, proto_flag, block_time)
int client_id;
char *buf;
int buf_length;
int proto_flag;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function	
buf	Data to be written to the ICP	
buf_length	Number of bytes in buf to write	
proto_flag	Special qualifier flag:	
	CS_DF_X25OOB	Interrupt data
	CS_DF_X25Q	Qualified data
	CS_DF_X25D	Delivery confirmation
	CS_DF_X25MORE	More data flag
	CS_DF_UI	UI frame data flag
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout	

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Packet Exchanges, Error Returns, and Events Generated

For USER:

```
HDATA
----------------->       for normal (I-frame) data
```

or

```
HUNDATA
----------------->       for UI-frame data
```

or

```
HINT
----------------->       CS_DF_X25OOB
```

For MANAGER:

Manager applications provide all ICPHDR data except the session number which is provided by the CS API.

5.5 Connection Shutdown

The following CS API functions are called for shutting down a connection or Freeway access.

5.5.1 cs_deregister

The cs_deregister function removes the application as an incoming connection handler. When using non-blocking I/O, and the cs_deregister call completes:

- the CS_DEREG_SUCCESS event is queued if the call was successful
- the CS_DEREG_REJECT event is queued if an IREJECT command was received

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_deregister(client_id, block_time)
int client_id;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Packet Exchanges, Error Returns, and Events Generated

Command	Error Return	Event
HDEL_ICF		
---------------->		
IACKNOWLEDGE		
<----------------	CS_NO_ERROR	CS_DEREG_SUCCESS
or		
IREJECT		
<----------------	CS_SVRERR	CS_DEREG_REJECT

5.5.2 cs_disconnect

The cs_disconnect function terminates an X.25 connection to a remote DTE. When using non-blocking I/O, and the disconnect call completes:

- the CS_DISCONN_SUCCESS event is queued if the call was successful
- the CS_DISCONN_REJECT event is queued if an IREJECT command was received

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_disconnect(client_id, qos, block_time)
int client_id;
struct cs_qos_st *qos;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
qos	Quality of service functions (NULL = none)
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

The qos parameter is optional. The structure cs_qos_st (defined in the cs_struc.h include file) is used to convey the reason (cause and diagnostic codes) for disconnecting this connection and optional user data in the resulting X.25 clear request. The item_list field

must point to a byte array containing the cause code and diagnostic code facilities and optional user data facility selected from the list given below.

HF_CAUSE	Cause code facility
HF_CLLED	Called DTE address
HF_CLLNG	Calling DTE address
HF_DIAG	Diagnostic code
HF_NONSTD	Non-standard user facilities
HF_USER	User data facility

Each item_list specification contains from one to three sub-fields: a function code byte, an item size indicator byte, and a variable-length item data field. You must specify the sub-fields contiguously in the order given, and you must omit sub-fields not listed for a given item from the specification. Refer to Section 5.7 for a complete description of these specifications, including the sub-fields.

Under special circumstances, the DTE to which the call was presented might not be the DTE originally specified as the called DTE. In this case, the called line address modified notification facility (HF_CLDADMOD) qos parameter can be used in addition to those listed above.

Packet Exchanges, Error Returns, and Events Generated

For X.25:

```
Command                 Error Return    Event

HHANGUP
----------------->
ITONE
<-----------------      CS_NO_ERROR     CS_DISCONN_SUCCESS
```

or

```
Command                 Error Return    Event

IREJECT
<-----------------      CS_SVRERR       CS_DISCONN_REJECT
```

or

```
IHANGUP
<-----------------      CS_LOST_CONN    CS_DISCONN_HANGUP
```

For HDLC:

```
Command                 Error Return    Event

HDISABLE
----------------->
IDISABLE
<-----------------      CS_NO_ERROR     CS_DISCONN_SUCCESS
```

5.5.3 cs_unbind

The cs_unbind function removes an application binding to a protocol service. When using non-blocking I/O, and the unbind call completes:

- the CS_UNBIND_SUCCESS event is queued if the call was successful
- the CS_UNBIND_REJECT event is queued if an IREJECT command was received

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_unbind(client_id, block_time)
int client_id;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Packet Exchanges, Error Returns, and Events Generated

For HDLC and X.25 SVC:

```
Command                 Error Return    Event

HCLOSE_SESSION
----------------->
ICLOSE_SESSION
<-----------------      CS_NO_ERROR     CS_UNBIND_SUCCESS
```

or

```
IREJECT
<-----------------      CS_SVRERR       CS_UNBIND_REJECT
```

For X.25 PVC:

```
Command                 Error Return    Event

HCLOSE_PVC
----------------->
ICLOSE_PVC
<-----------------
HCLOSE_SESSION
----------------->
ICLOSE_SESSION
<-----------------      CS_NO_ERROR     CS_UNBIND_SUCCESS
```

or

```
IREJECT
<-----------------      CS_SVRERR       CS_UNBIND_REJECT
```

5.5.4 cs_detach

The cs_detach function detaches the application from an ICP. When using non-blocking I/O, and the detach call completes:

- the CS_DETACH_SUCCESS event is queued if the call was successful
- the CS_DETACH_FAILED event is queued if the call was unsuccessful

If the function returns CS_CALL_TIMEOUT when using blocking I/O, the application must call cs_select to detect return event types mentioned above.

Synopsis

```
int cs_detach(client_id, block_time)
int client_id;
int block_time;
```

Arguments

client_id	Connection ID from the cs_attach function
block_time	Routine block time in seconds; if not 0, this routine will block until completion or timeout

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

5.5.5 cs_terminate

The cs_terminate function disconnects the application from the DLI and releases the system resources.

Synopsis

```
int cs_terminate()
```

Arguments

None

Return Value

None

5.6 Miscellaneous

The following functions provide miscellaneous services and debug logging.

5.6.1 cs_sperror

The cs_sperror function returns a string containing an error message for the specified CS API error number.

Synopsis

```
char *cs_sperror(errornum)
int errornum;
```

Arguments

errornum	Error number returned by other CS API calls

Return Value

This function always returns a pointer to an error text string.

5.6.2 cs_sleep

The cs_sleep function places the application in a sleep mode and releases the CPU.

Synopsis

```
void cs_sleep(duration)
int duration;
```

Arguments

duration	Number of seconds to sleep

Return Value

None

5.6.3 cs_config

The cs_config function returns the application's ppa_struct data describing the physical point of attachment to the WAN.

Synopsis

```
int cs_config(client_id, ppa)
int client_id;
struct ppa_struct *ppa;
```

where the following structure is provided by the calling routine to receive the configuration data:

```
struct ppa_struct
{
         int protocol;   /* SAP_X25 or SAP_SLP (HDLC)*/
         int level;      /* USER_API or MANAGER_API*/
         int port;       /* Port on ICP board      */
         int pvc;        /* PVC (station ID)       */
         int board;      /* ICP board ID           */
}ppa;
```

Arguments

client_id	Connection ID from the cs_attach function
ppa	Returned ppa structure

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

5.6.4 cs_getpid

The cs_getpid function returns the process id (pid).

Synopsis

```
int cs_getpid()
```

Arguments

None

Return Value

The return value is the process ID number.

5.6.5 debuglog

The debuglog function provides a debug message logging facility for applications.

Synopsis

```
int debuglog(format, ...)
char *format;
...
```

Arguments

format	Message format (same as in a printf call)
...	Parameters for the format (same as in a printf call)

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

The debug file is a circular file containing up to 20000 records of 80 bytes each. The debug file is named Dbugnnnn.X25, where nnnn is the hex value of the process ID.

For example, on March 17 at 12:34:14, the following call:

```
debuglog ("value returned for %d tries was %d", 2, 5);
```

results in the following text line being added to the debug log file:

```
Mar 17 12:34:14 – value returned for 2 tries was 5
```

5.6.6 cs_suicide

The cs_suicide function calls the DLI function DlPoll to write the trace file to disk and terminates the program without cleaning up system resources or X.25 connections. This function is provided solely for customer service support in diagnosing application problems with LAN access to Freeway.

Synopsis

```
int cs_suicide()
```

Arguments

None

Return Value

This routine does not return to the caller. The application program is terminated.

5.6.7 cs_suspend_events

The cs_suspend_events function suspends the delivery of generated events. Events are not dequeued until the application calls cs_resume_events. The suspend and resume calls are nestable; this call increments a counter and cs_resume_events decrements the counter.

Synopsis

```
int cs_suspend_events()
```

Arguments

None

Return Value

The return value is the suspend nesting level.

5.6.8 cs_resume_events

The cs_resume_events function resumes the deliver of generated events when the suspend counter goes to zero. The suspend and resume calls are nestable; this call decrements a counter and cs_suspend_events increments the counter.

Synopsis

```
int cs_resume_events(num)
int num;
```

Arguments

num	Amount to decrement the suspend count. If num is set to –1, the suspend count will be reset to zero and event dequeuing will be enabled.

Return Value

The return value is the suspend nesting level.

5.6.9 cs_gen_event

The cs_gen_event function allows the application to generate its own event.

Synopsis

```
int cs_gen_event(client_id, event, data, len)
int client_id;
int event;
char *data;
int len;
```

Arguments

client_id	Connection ID from the cs_attach function
event	Event ID (may be a CS API event or an application-defined event)
data	Event data buffer (may be NULL)
len	Length, in bytes, of data

Return Value

On success, this function returns zero. On failure, this function returns a number less than zero.

Note

The generated event is placed at the end of the event queue and read completed queue. The function works for both blocking and non-blocking applications.

5.6.10 cs_bufsize

The cs_bufsize function returns the maximum data buffer size allowed for the connection. This is the negotiated size between the client and an ICP.

Synopsis

```
int cs_bufsize(client_id)
int client_id;
```

Arguments

client_id	Connection ID from the cs_attach function

Return Value

On success, this function returns the maximum number of bytes allowed in a data packet for the passed client_id.

5.7 QOS Item Formats

This section describes the format of the qos items. Most of the items have three sub-fields: an identification code, a size (which might be variable), and data; however, some items have two sub-fields, and a few have only one. Table 5–3 shows the qos items listed by code showing their qos symbol and a brief description of their purpose. Table 5–4 shows the qos items listed alphabetically showing the accessed CS API requests and whether the requests use the qos or ret_qos parameters. The following list describes each qos item and its applicable sub-fields.

HF_BLCLUS This is the bilateral closed user group selection facility. It requires network support for closed user group facilities.

Code: HF_BLCLUS (32)

Size: Field length in bytes (4)

Data: Four-digit group number (string of decimal digits 0 through 9)

HF_CAUSE This is the cause code facility. It identifies the reason for the associated event or action.

Code: HF_CAUSE (14)

Data: Cause code byte (must be zero, or 128 through 255 when specified by the application)

HF_CLDADMOD This is the called line address modified notification facility. It reports that the HF_CLLED facility value differs from that specified in the original cs_connect request issued by the calling DTE identified by the HF_CLLNG facility.

Code: HF_CLDADMOD (39)

Data: Reason code byte

Table 5–3: CS API QOS Options Listed

qos Symbol	Code	Description
HF_CLLNG	1	Calling DTE
HF_CLLED	2	Called DTE
HF_FASNR	4	Fast select facility -- no restriction
HF_FASR	5	Fast select facility -- restriction on response
HF_RVFC	7	Reverse charging facility
HF_USER	10	User data
HF_CAUSE	14	Cause
HF_DIAG	15	Diagnostic
HF_NONSTD	24	Non-standard user facilities
HF_THRUCLASS	26	Throughput class
HF_PRIORITY	27	Virtual circuit priority on Freeway
HF_PWSIZE	28	Packet window size facility
HF_PDSIZE	29	Packet data size facility
HF_CLUSG	30	Closed user group selection facility
HF_CLUSGOAS	31	CUG with outgoing access selection facility
HF_BLCLUS	32	Bilateral closed user group facility
HF_NWUSID	33	Network user identification facility
HF_RQCRGIN	34	Request charging information facility
HF_RCVMONY	35	Receive charging monetary unit facility
HF_RCVSGCNT	36	Receive charging segment count facility
HF_RCVCLDR	37	Receive charging call duration facility
HF_RPOA	38	RPOA selection facility
HF_CLDADMOD	39	Called line address modified notification facility
HF_CLREDR	40	Call redirection or deflection notification facility
HF_TRDLYSEL	41	Transit delay selection and indication facility
HF_D_BIT_SUPPORT	46	D-bit support request and indication
HF_CLDEFLECT	48	Call deflection selection facility
HF_ICF_PRIORITY	64	ICF priority
HF_DTE_REMOTE	65	ICF remote (calling) DTE address
HF_DTE_SA_LOW	66	ICF local (called) DTE subaddress (low)
HF_DTE_SA_HIGH	67	ICF local (called) DTE subaddress (high
HF_USER_MASK	68	ICF user data mask
HF_ICF_CALLBUSY	69	ICF call busy configuration

Table 5–4: CS API Functions QOS Support

qos Symbol	Code	cs_accept	cs_connect	cs_connect_nb_remote	cs_disconnect	cs_refuse	cs_listen	cs_redirect	cs_register	cs_reset	cs_read
HF_BLCLUS	32		Q				R				
HF_CAUSE	14				Q	Q				Q	X
HF_CLDADMOD	39	Q	R	R	Q	Q					N
HF_CLDEFLECT	48					Q					
HF_CLLED	2	Q	QR	R	Q	Q	R				
HF_CLLNG	1	Q	QR	R	Q	Q	R				
HF_CLREDR	40						R				
HF_CLUSG	30		Q				R				
HF_CLUSGOAS	31		Q				R				
HF_DIAG	15				Q	Q				Q	X
HF_DTE_REMOTE	65								Q		
HF_DTE_SA_HIGH	67								Q		
HF_DTE_SA_LOW	66								Q		
HF_D_BIT_SUPPORT	46	Q	Q	R			R				
HF_FASNR	4		Q				R				
HF_FASR	5		Q				R				
HF_ICF_CALLBUSY	69								Q		
HF_ICF_PRIORITY	64								Q		
HF_NONSTD	24	Q	QR	R	Q	Q	R				
HF_NWUSID	33	Q	Q								
HF_PDSIZE	29	Q	QR	R			R				
HF_PRIORITY	27	Q	Q	R			R			Q	X

HF_CLDEFLECT This is the call deflection selection facility. It is used when refusing an incoming call to specify an alternate DTE address to which the call is to be deflected. The DTE must also include any CCITT-specified DTE facilities and user data to be sent to the alternate DTE.

Code: HF_CLDEFLECT (48)

Size: Field length in bytes

Data: Reason code byte (192 through 255) followed by alternate called DTE address (string of ASCII digits 0 through 9)

HF_CLLED This is the called DTE address. It is the network address of the destination DTE to which the SVC connection is directed.

Code: HF_CLLED (2)

Size: Called DTE address length in bytes (0 through 15, or 3 through 17 for TOA/NPI)

Data: Called DTE address (string of ASCII digits 0 through 9); if TOA/NPI format is used, the TOA and NPI digits must precede the DTE address

HF_CLLNG This is the calling DTE address. It is the network address of the DTE requesting the SVC connection.

Code: HF_CLLNG (1)

Size: Calling DTE address length in bytes (0 through 15, or 3 through 17 for TOA/NPI)

Data: Calling DTE address (string of ASCII digits 0 through 9); if TOA/NPI format is used, the TOA and NPI digits must precede the DTE address

HF_CLREDR This is the call redirection or call deflection notification facility. It informs the application that the call has been redirected by the network (or deflected by the called DTE), the reason for the redirection (or deflection), and the address of the originally called DTE.

Code: HF_CLREDR (40)

Size: Field length in bytes

Data: Reason code byte (1 = busy, 7 = call distribution within a hunt group, 9 = out-of-order, 15 = systematic, 192 through 255 = deflection by originally called DTE) followed by originally called DTE address (string of ASCII digits 0 through 9)

HF_CLUSG This is the closed user group selection facility. It requires network support for closed user group facilities. The HF_CLUSG and HF_CLUSGOAS facilities cannot both appear in the same item_list.

Code: HF_CLUSG (30)

Size: Field length in bytes (2 or 4)

Data: Two-digit or four-digit group number (string of decimal digits 0 through 9)

HF_CLUSGOAS This is the closed user group with outgoing access selection facility. It requires network support for closed user group facil-

ities. The HF_CLUSG and HF_CLUSGOAS facilities cannot both appear in the same item_list.

Code: HF_CLUSGOAS (31)

Size: Field length in bytes (2 or 4)

Data: Two-digit or four-digit group number (string of decimal digits 0 through 9)

HF_DIAG This is the diagnostic code facility. It provides an additional diagnosis of the reason for the associated event or action. A list of diagnostic codes appears in Appendix C.

Code: HF_DIAG (15)

Data: Diagnostic code byte

HF_DTE_REMOTE This is the remote DTE address—the portion of the main DTE address that must match the actual calling DTE address in incoming calls. If the DTE address length for the X.25 network has not previously been configured on Freeway, this parameter can be rejected.

Code: HF_DTE_REMOTE (65)

Size: Field length in bytes (1 through 15, or 3 through 17 for TOA/NPI)

Data: Remote DTE address (string of ASCII digits 0 through 9); if TOA/NPI format is used, the TOA and NPI digits must precede the DTE address

HF_DTE_SA_HIGH This is the highest local DTE subaddress—the highest acceptable called DTE subaddress in incoming calls. The DTE subad-

dress immediately follows the main DTE address, whose length (DTE_address_length[1]) has previously been configured.

Code: HF_DTE_SA_HIGH (67)

Size: Field length in bytes (1 through 15 (17 for TOA/NPI) minus the configured DTE_address_length)

Data: Highest local DTE subaddress (string of ASCII digits 0 through 9)

HF_DTE_SA_LOW This is the lowest local DTE subaddress—the lowest acceptable called DTE subaddress in incoming calls. The DTE subaddress immediately follows the main DTE address, whose length (DTE_address_length) has previously been configured.

Code: HF_DTE_SA_LOW (66)

Size: Field length in bytes (1 through 15 (17 for TOA/NPI) minus the configured DTE_address_length)

Data: Lowest local DTE subaddress (string of ASCII digits 0 through 9)

HF_D_BIT_SUPPORT This is the D-bit support request/indication.

Code: HF_D_BIT_SUPPORT (46)

Data: Single byte value (0 = D-bit not supported; 1 = D-bit supported)

1. The DTE_address_length must be set by the x25_manager utility when configuring Freeway X.25 call service parameters for data links.

HF_FASNR This is the fast select facility (unrestricted). It allows the called DTE to accept the call.

Code: HF_FASNR (4)

HF_FASR This is the fast select facility with restrictions. It prevents the called DTE from accepting the call.

Code: HF_FASR (5)

HF_ICF_CALLBUSY This is an incoming call filter configuration parameter internal to Freeway. The default configuration is to hold additional calls when an existing call is already associated with the incoming call filter. This parameter may be used to request call redirection rather than call holding.

Code: HF_ICF_CALLBUSY (69)

Size: Field size in bytes (1)

Data: 0 = hold additional calls when call busy
1 = redirect additional calls when call busy

HF_ICF_PRIORITY This is an incoming call filter priority level internal to Freeway. The ICF priority is independent of the virtual-circuit local priority HF_PRIORITY that can be set when the application accepts a call or resets a virtual circuit.

Code: HF_ICF_PRIORITY (64)

Size: Field size in bytes (1)

Data: Value zero through 255, where zero is the lowest ICF priority

HF_NONSTD This is the non-standard user facility.

Code: HF_NONSTD (24)

Size: Field size in bytes

Data: Non-standard facilities marker code byte 0, followed by parameter byte (–1, 0, or 15), and actual non-standard facilities (binary)

HF_NWUSID This is the network user identification facility. Not all networks support the use of this facility.

Code: HF_NWUSID (33)

Size: Field size in bytes

Data: Network user identification

HF_PDSIZE This is the packet data size facility. It is used to negotiate the allowed size of the data field within X.25 data packets exchanged between the local and remote DTE. It does not apply to the size of data transfers requested by the application using cs_read and cs_write requests.

Code: HF_PDSIZE (29)

Data: Two packet data size selection bytes (for local and remote DTE); selection codes 4 through 12 indicate the power-of-two data sizes 16 through 4096

HF_PRIORITY This is the virtual circuit local priority facility on Freeway. It selects the degree to which Freeway is to favor transmitting data sent using this virtual circuit. When Freeway has data ready to send on more than one virtual circuit, it sends data first on the

virtual circuit with the higher local priority. This facility does not affect the handling of received data, which is always on a first-come, first-served basis.

Code: HF_PRIORITY (27)

Data: Priority code byte (0 = low; 1 = normal; 2 = high)

HF_PWSIZE This is the packet window size facility. It is used to negotiate the allowed size of the packet transmit window for the local and remote DTE. It does not apply to the transmit window applied by the CS API to application cs_write requests.

Code: HF_PWSIZE (28)

Data: Two packet window size selection bytes (for local and remote DTE). Packet windows are 1 through 7, or extended, 1 through 127. The extended packet window is allowed only if Freeway is configured to support extended packet sequence numbers.

HF_RCVCLDR This is the receiving charging call duration facility. It can be reported by the network DCE to the DTE for accounting purposes.

Code: HF_RCVCLDR (37)

Size: Field size in bytes (multiple of 4)

Data: List of tariff periods (string of ASCII digits 0 through 9), each expressed as a four-digit code giving the number of days, hours, minutes, and seconds for the period

HF_RCVMONY This is the receiving charging monetary unit facility. It can be reported by the network DCE to the DTE for accounting purposes.

Code: HF_RCVMONY (35)

Size: Field size in bytes

Data: List of monetary unit octets (binary)

HF_RCVSGCNT This is the receiving charging segment count facility. It can be reported by the network DCE to the DTE for accounting purposes.

Code: HF_RCVSGCNT (36)

Size: Field size in bytes (multiple of 8)

Data: List of segment count information sets. Each information set contains a DCE segment count binary longword and a DTE segment count binary longword for a specific tariff period. Information for multiple tariff periods is sequentially listed.

HF_RPOA This is the RPOA selection facility.

Code: HF_RPOA (38)

Size: Field size in bytes (multiple of 4)

Data: List of four-digit RPOA transit points (string of decimal digits 0 through 9)

HF_RQCRGIN This is the request charging information facility. It requests the network DCE to provide charging information facilities (HF_RCVCLDR, HF_RCVMONY, or HF_RCVSGCNT) when the virtual circuit is terminated.

Code: HF_RQCRGIN (34)

HF_RVFC This is the reverse charging facility. It requests the network to reverse the charges for the call by charging the called DTE.

Code: HF_RVFC (7)

HF_THRUCLASS This is the throughput class facility. It is used to negotiate the throughput class for data transferred using this virtual circuit through the network. The throughput class negotiated for the local DTE can differ from that negotiated for the remote DTE. The throughput class does not apply to the actual data rate at the interface between the network DTE and the attached local or remote DTE. Instead, the throughput class applies to the propagation rate through the network.

Code: HF_THRUCLASS (26)

Data: Two throughput class selection bytes (for local and remote DTE). Selection codes 3 through 12 indicate throughput classes 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, and 48000, respectively. For CCITT 1988, a selection code of 13 indicates throughput class 64000.

HF_TRDLYSEL This is the transit delay selection and indication facility. It gives the nominal maximum network transit delay applicable to data transferred through the network.

Code: HF_TRDLYSEL (41)

Size: Field size in bytes (2)

Data: Transit delay in seconds (16-bit binary)

HF_USER This is the user data facility. It is used to pass user data associated with SVC connection and termination.

Code: HF_USER (10)

Size: Field size in bytes (1 through 16 normally, or fast select, 1 through 128)

Data: User data (binary)

HF_USER_MASK This is the user data mask for the incoming call filter. It specifies a binary mask of bits to be logically ANDed with the actual HF_USER facility data in the incoming call and the HF_USER facility data in a registered incoming call filter before comparing the HF_USER facilities data fields.

Code: HF_USER_MASK (68)

Size: Field size in bytes (1 through 16)

Data: User data mask (binary)

Appendix

A CS API Include Files

This appendix shows the text of source files normally required by application programs for inclusion of the definitions of symbolically named values and data structures referenced in this document.

The cs_api.h file includes all files required by application programs using the CS API. See Section A.1.

The cs_dfine.h file contains system definitions. See Section A.2.

The cs_struc.h file contains system data structures. See Section A.3.

The cs_errno.h file provides symbol definitions for error or indication status codes required by the API. See Section A.4.

The cs_proto.h file contains ANSI C function prototypes. See Section A.4.

The cs_x25.h file contains X.25 packet and configuration definitions. See Section A.6.

A.1 cs_api.h

```
/*************************************************************************
*               CONFIDENTIAL & PROPRIETARY INFORMATION
*              Distribution to Authorized Personnel Only
*           Unpublished/Copyright 1992 - 1996 Simpact, Inc.
*                       All Rights Reserved
*
* This document contains confidential and proprietary information of Simpact,
* Inc, ("Simpact") and is protected by copyright, trade secret and other state
* and federal laws. The possession or receipt of this information does not
* convey any right to disclose its contents, reproduce it, or use, or license
* the use, for manufacture or sale, the information or anything described
* therein. Any use, disclosure, or reproduction without Simpact's prior
* written permission is strictly prohibited.
*
* Software and Technical Data Rights
*
* Simpact software products and related documentation will be furnished
* hereunder with "Restricted Rights" in accordance with:
*
*      A. Subparagraph (c)(1)(ii) of the clause entitled Rights in Technical
*      Data and Computer Software (OCT 1988) located at DFARS 252.227-7013; or
*
*      B. Subparagraph (c)(2) of the clause entitled Commercial Computer
*      Software - Restricted Rights (JUN 1987) located at FAR 52.227.19.
*************************************************************************/

/*************************************************************************
*    MODULE: cs_api.h
*
*        Include header file.
*        Contains api header files required by application programs
*
*    MODIFICATIONS:
*        Original 04/94
*
*************************************************************************/
#ifndef CS_API_H
#define CS_API_H

#ifdef WINNT
#include <fwywinnt.h>
#endif

#include <cs_dfine.h>    /* definitions required by application */
#include <cs_struc.h>    /* api structure definitions           */
#include <cs_proto.h>    /* api function prototypes             */
#include <cs_errno.h>    /* errors returned to application      */
#include <cs_x25.h>      /* x25 low-level header file           */

#endif  /* ifndef CS_API_H */
```

A.2 cs_dfine.h

```
/*****************************************************************************
*               CONFIDENTIAL & PROPRIETARY INFORMATION
*              Distribution to Authorized Personnel Only
*           Unpublished/Copyright 1992 - 1996 Simpact, Inc.
*                        All Rights Reserved
*
* This document contains confidential and proprietary information of Simpact,
* Inc, ("Simpact") and is protected by copyright, trade secret and other state
* and federal laws. The possession or receipt of this information does not
* convey any right to disclose its contents, reproduce it, or use, or license
* the use, for manufacture or sale, the information or anything described
* therein. Any use, disclosure, or reproduction without Simpact's prior
* written permission is strictly prohibited.
*
* Software and Technical Data Rights
*
* Simpact software products and related documentation will be furnished
* hereunder with "Restricted Rights" in accordance with:
*
*      A. Subparagraph (c)(1)(ii) of the clause entitled Rights in Technical
*      Data and Computer Software (OCT 1988) located at DFARS 252.227-7013; or
*
*      B. Subparagraph (c)(2) of the clause entitled Commercial Computer
*      Software - Restricted Rights (JUN 1987) located at FAR 52.227.19.
*****************************************************************************/

/*****************************************************************************
*    MODULE: cs_dfine.h
*
*        Include header file.
*        Contains definitions required by application programs
*
*    MODIFICATIONS:
*        Original 04/94
*        war   12/15/95   add dead socket detection (add CS_DEAD_SOCKET)
*
*****************************************************************************/

/*
** Generic integer definitions
*/
typedef unsigned short  unsigned16;

/*
** event types
*/
#define EVBASE             -29000

#define CS_REJECT          EVBASE- 1
#define CS_RESET           EVBASE- 2
 /* empty slot */
#define CS_INTERRUPT       EVBASE- 4
#define CS_INTERRUPT_ACK   EVBASE- 5
#define CS_LOST_CONN       EVBASE- 6
#define CS_ABORT           EVBASE- 7
#define CS_INC_CALL        EVBASE- 8
#define CS_AUTO_CONNECT    EVBASE- 9
#define CS_QUEUE_OVERFLOW  EVBASE-10
#define CS_WRITE_COMPLETE  EVBASE-11
#define CS_READ_COMPLETE   EVBASE-12
#define CS_ATTACH_SUCCESS  EVBASE-13
#define CS_ATTACH_FAILED   EVBASE-14
#define CS_DETACH_SUCCESS  EVBASE-15
#define CS_DETACH_FAILED   EVBASE-16
#define CS_BIND_SUCCESS    EVBASE-17
```

```
#define CS_BIND_FAILED     EVBASE-18
#define CS_BIND_REJECT     EVBASE-19
#define CS_UNBIND_SUCCESS  EVBASE-20
#define CS_UNBIND_REJECT   EVBASE-21
#define CS_REG_SUCCESS     EVBASE-22
#define CS_REG_REJECT      EVBASE-23
#define CS_DEREG_SUCCESS   EVBASE-24
#define CS_DEREG_REJECT    EVBASE-25
#define CS_CONN_SUCCESS    EVBASE-26
#define CS_CONN_REJECT     EVBASE-27
#define CS_CONN_HANGUP     EVBASE-28
#define CS_DISCONN_SUCCESS EVBASE-29
#define CS_DISCONN_REJECT  EVBASE-30
#define CS_ACCEPT_SUCCESS  EVBASE-31
#define CS_ACCEPT_REJECT   EVBASE-32
#define CS_ACCEPT_HANGUP   EVBASE-33
#define CS_REDIR_SUCCESS   EVBASE-34
#define CS_REDIR_REJECT    EVBASE-35
#define CS_REDIR_HANGUP    EVBASE-36
#define CS_REFUSE_SUCCESS  EVBASE-37
#define CS_REFUSE_REJECT   EVBASE-38
#define CS_REFUSE_HANGUP   EVBASE-39
#define CS_RESET_SUCCESS   EVBASE-40
#define CS_RESET_REJECT    EVBASE-41
#define CS_RESET_HANGUP    EVBASE-42
#define CS_WRITE_TIMEOUT   EVBASE-43
#define CS_MEM_EXAUSTED    EVBASE-44
#define CS_DLI_FATAL       EVBASE-45
#define CS_HANGUP          EVBASE-46
#define CS_ERROR           EVBASE-47
#define CS_CONN_TIMEOUT    EVBASE-48
#define CS_WRITE_FAILED    EVBASE-49
#define CS_ICP_RESETTING   EVBASE-50
#define CS_ICP_READY       EVBASE-51
#define CS_DEAD_SOCKET     EVBASE-52
#define CS_BUF_OVERFLOW    EVBASE-53
#define CS_FW_UNBOUND      EVBASE-54

#define EVEND              EVBASE-55


/*
** macro to determine if a returned value is an event, returns
** 1 (TRUE) if it is an event, else returns 0 (FALSE)
*/
#define IS_EVENT(n) (n < EVBASE && n > EVEND)

/*
** icp x.25 service access points
*/
#define SAP_X25  11    /* X.25 access point (over MLP or SLP) */
#define SAP_DIAG 12    /* X.25 diagnostic access point        */
#define SAP_MLP  13    /* Non-X.25 MLP access point           */
#define SAP_SLP  14    /* Non-X.25 HDLC LAPB access point     */

/*
** ICP X.25 API Data Flag definitions
*/
#define CS_DF_X25Q     0x01         /* Data flag: Qualified data        */
#define CS_DF_X25MORE  0x02         /* Data flag: More data             */
#define CS_DF_X25D     0x04         /* Data flag: Delivery confirmation */
#define CS_DF_UI       0x08         /* Data flag: Data for UI frame     */

#define CS_DF_X25OOB   CS_INTERRUPT /* Data flag: OOB (interrupt)  */
#define CS_DF_X25RSET  CS_RESET     /* Data flag: Reset data       */
#define CS_DF_NOCONN   CS_LOST_CONN /* Read flag: disconnect data  */

/*
```

```
** indication types (for compatability)
*/
#define CS_INDNOCONN      CS_LOST_CONN
#define CS_INDX25OOB      CS_INTERRUPT
#define CS_INDX25OOB_ACK  CS_INTERRUPT_ACK
#define CS_INDX25RSET     CS_RESET
#define CS_INDX25RSET_ACK CS_RESET_SUCCESS

/*
** ICP X.25 Host Facility code definitions used in the qos item list
*/
#define HF_CLLNG          1   /* Calling DTE                          */
#define HF_CLLED          2   /* Called DTE                           */
#define HF_FASCN          3   /* Fast Select Configuration            */
#define HF_FASNR          4   /* Fast Select Facility - no restriction
                                 on response */
#define HF_FASR           5   /* Fast Select Facility - restriction on
                                 response    */
#define HF_RVCN           6   /* Reverse Charging Configuration       */
#define HF_RVFC           7   /* Reverse Charging Facility            */
#define HF_INCM           8   /* Incoming Call Configuration          */
#define HF_STDSV          9   /* DDN standard or basic services       */
#define HF_USER          10   /* User Data                            */
#define HF_T2XCN         11   /* T2X Timer Configuration              */
#define HF_R2XCN         12   /* R2X Retry Configuration              */
#define HF_TLX           13   /* Timer configuration                  */
#define HF_CAUSE         14   /* Cause                                */
#define HF_DIAG          15   /* Diagnostic                           */
#define HF_PREC          16   /* Packet precedence                    */
#define HF_LCN           17   /* LCN bounds (LIC, HIC, LTC, HTC, LOC, HOC)*/
#define HF_CERT          18   /* Certification mode                   */
#define HF_FLOW          19   /* Negotiate flow control parameters    */
#define HF_CLR           20   /* Automatic clear confirmation         */
#define HF_PASSPKT       21   /* Pass packet to network w/o processing*/
#define HF_CLM           22   /* Control line monitoring (CTS/DCD)    */
#define HF_CLTMR         23   /* Control line timer reset values      */
#define HF_NONSTD        24   /* Non standard facility data           */
#define HF_RESTART       25   /* Auto-restart on SABM/UA              */
#define HF_THRUCLASS     26   /* Throughput class                     */
#define HF_PRIORITY      27   /* Virtual circuit priority             */
#define HF_PWSIZE        28   /* Packet window size facility          */
#define HF_PDSIZE        29   /* Packet data size facility            */
#define HF_CLUSG         30   /* Closed user group selection facility */
#define HF_CLUSGOAS      31   /* Closed user group with outgoing access
                                 selection facility                   */
#define HF_BLCLUS        32   /* Bilateral closed user group facility */
#define HF_NWUSID        33   /* Network user identification facility */
#define HF_RQCRGIN       34   /* Request charging information facility*/
#define HF_RCVMONY       35   /* Recv charging monitary unit facility */
#define HF_RCVSGCNT      36   /* Recv charging segment count facility */
#define HF_RCVCLDR       37   /* Recv charging call duration facility */
#define HF_RPOA          38   /* RPOA selection facility              */
#define HF_CLDADMOD      39   /* Called line address modified
                                 notification facility                */
#define HF_CLREDR        40   /* Call redirection notification facility */
#define HF_TRDLYSEL      41   /* Transit delay selection and indication
                                 facility            */
#define HF_REJ           42   /* Packet level REJ support             */
#define HF_MOD128        43   /* Packet level MOD 128 support         */
#define HF_X25_PROFILE   44   /* X.25 profile selection               */
#define    UNRESTRICTED   0   /*    Default         */
#define    CCITT_1976     1   /*    not implemented */
#define    CCITT_1980     2
#define    CCITT_1984     3
#define    ISO_8208       4
#define    CCITT_1988     5

#define HF_ADDR_LEN      45   /* Call service: Local DTE address length */
```

```
#define HF_D_BIT_SUPPORT 46   /* API_USER client use only             */
#define HF_TOANPI        47   /* Call service: DTE address format     */
#define HF_CLDEFLECT     48   /* Call deflection selection facility   */

#define HF_ICF_PRIORITY  64   /* ICF: Priority                        */
#define HF_DTE_REMOTE    65   /* ICF: Remote (calling) DTE address    */
#define HF_DTE_SA_LOW    66   /* ICF: Local (called) DTE subaddress (low) */
#define HF_DTE_SA_HIGH   67   /* ICF: Local (called) DTE subaddress (high)*/
#define HF_USER_MASK     68   /* ICF: User data mask                  */
#define HF_ICF_CALLBUSY  69   /* ICF: Call-busy (0=Hold (default), 1=Redirect) */

#define HF_NEGP1        128   /* Registration facilities negotiable in p1 */
#define HF_NEGANY       129   /* Registration facilities negotiable in any
                                 state  */
#define HF_AVAIL        130   /* Availability of facilities           */
#define HF_NON_NEG      131   /* Non-negotiable facilities            */
#define HF_DFTHRU       132   /* Default throughput class assignment  */
#define HF_NSPACK       133   /* Non-standard default packet sizes    */
#define HF_NSWIN        134   /* Non-standard default window sizes    */
#define HF_LOGCHAN      135   /* Logical channel types ranges         */
```

A.3 cs_struc.h

```
/*****************************************************************************
*               CONFIDENTIAL & PROPRIETARY INFORMATION
*              Distribution to Authorized Personnel Only
*           Unpublished/Copyright 1992 - 1996 Simpact, Inc.
*                       All Rights Reserved
*
* This document contains confidential and proprietary information of Simpact,
* Inc, ("Simpact") and is protected by copyright, trade secret and other state
* and federal laws. The possession or receipt of this information does not
* convey any right to disclose its contents, reproduce it, or use, or license
* the use, for manufacture or sale, the information or anything described
* therein. Any use, disclosure, or reproduction without Simpact's prior
* written permission is strictly prohibited.
*
* Software and Technical Data Rights
*
* Simpact software products and related documentation will be furnished
* hereunder with "Restricted Rights" in accordance with:
*
*      A. Subparagraph (c)(1)(ii) of the clause entitled Rights in Technical
*      Data and Computer Software (OCT 1988) located at DFARS 252.227-7013; or
*
*      B. Subparagraph (c)(2) of the clause entitled Commercial Computer
*      Software - Restricted Rights (JUN 1987) located at FAR 52.227.19.
*****************************************************************************/

/*****************************************************************************
*    MODULE: cs_struc.h
*
*       Include header file.
*       Contains api structures required by application programs
*
*    MODIFICATIONS:
*        Original 04/94
*
*****************************************************************************/
/*
** ICP SAP Header structure - only used for CS_MANAGER level
*/
typedef struct icphdr       /* service access point sub-header */
{
    unsigned16  command;     /* SAP command          */
    unsigned16  modifier;    /* SAP command modifier */
    unsigned16  link;        /* SAP data link ID     */
    unsigned16  circuit;     /* SAP station ID       */
    unsigned16  session;     /* SAP session ID       */
    unsigned16  sequence;    /* SAP not used         */
    unsigned16  reserved[2];
} ICPHDR;



struct cs_listen_st
{
    int client;      /* cid with incomming connection indication */
    int token;
    int state;       /* incomming connection state               */
};

struct cs_qos_st
{
    int  max_length;   /* Maximum length of item_list          */
    int  tot_length;   /* Current total length of item_list    */
    char *item_list;   /* item list of quality of service list */
};
```

```
struct ppa_struct      /* PPA fields parsed       */
{
    int protocol;      /* Protocol (X25 or HDLC)  */
    int level;         /* USER_API or MANAGER_API */
    int port;          /* Port on ICP board       */
    int pvc;           /* PVC (specific field)    */
    int board;         /* ICP board               */
};
```

A.4 cs_errno.h

```
/*****************************************************************************
*               CONFIDENTIAL & PROPRIETARY INFORMATION
*              Distribution to Authorized Personnel Only
*           Unpublished/Copyright 1992 - 1996 Simpact, Inc.
*                       All Rights Reserved
*
* This document contains confidential and proprietary information of Simpact,
* Inc, ("Simpact") and is protected by copyright, trade secret and other state
* and federal laws. The possession or receipt of this information does not
* convey any right to disclose its contents, reproduce it, or use, or license
* the use, for manufacture or sale, the information or anything described
* therein. Any use, disclosure, or reproduction without Simpact's prior
* written permission is strictly prohibited.
*
* Software and Technical Data Rights
*
* Simpact software products and related documentation will be furnished
* hereunder with "Restricted Rights" in accordance with:
*
*      A. Subparagraph (c)(1)(ii) of the clause entitled Rights in Technical
*      Data and Computer Software (OCT 1988) located at DFARS 252.227-7013; or
*
*      B. Subparagraph (c)(2) of the clause entitled Commercial Computer
*      Software - Restricted Rights (JUN 1987) located at FAR 52.227.19.
*****************************************************************************/

/*****************************************************************************
*    MODULE: cs_errno.h
*
*       Include header file for Application Programmer's Interface of
*       Simpact's Communication Server products.
*
*       Error symbolic definitions.
*
*
*    MODIFICATIONS:
*        Original 04/94
*
*****************************************************************************/
/*
** error return values --- WARNING, changes here must also be made to
** cs_sperror routine in cs_erlog.c, AND CS_UNKNOWN_ERROR must be the
** last error number
*/
#define ERBASE                -20000

#define CS_NO_ERROR                  0
#define CS_INVALID_CIRCUIT  ERBASE- 1  /* invalid circuit name - cs_attach */
#define CS_CALL_TIMEOUT     ERBASE- 2  /* CS API call has timed out        */
#define CS_NO_MEMORY        ERBASE- 3  /* can not allocate memory          */
#define CS_NOT_ASYNC        ERBASE- 4  /* DL API is not ASYNC              */
#define CS_NOT_INIT         ERBASE- 5  /* cs_init not called yet           */
#define CS_MAX_UNACKS       ERBASE- 6  /* max unack writes, window closed  */
#define CS_BADCID           ERBASE- 7  /* invalid client ID                */
#define CS_INVREQ           ERBASE- 8  /* invalid request for current state*/
#define CS_SVRERR           ERBASE- 9  /* server error                     */
#define CS_NOBIND           ERBASE-10  /* not bound                        */
#define CS_INVALID_ICPHDR   ERBASE-11  /* manager passed bad icphdr length */
#define CS_STA_ERROR        ERBASE-12  /* IABORT or ISTAFAIL received      */
#define CS_SYS_RESOURCE     ERBASE-13  /* unable to allocate resources     */
#define CS_INVALID_BUFLEN   ERBASE-14  /* invalid write buffer length      */
#define CS_FILE_NOT_FOUND   ERBASE-15  /* config file not found            */
#define CS_UNKNOWN_ERROR    ERBASE-16  /* Unknown error                    */
```

A.5 cs_proto.h

```
/*****************************************************************************
*               CONFIDENTIAL & PROPRIETARY INFORMATION
*              Distribution to Authorized Personnel Only
*           Unpublished/Copyright 1992 - 1996 Simpact, Inc.
*                       All Rights Reserved
*
* This document contains confidential and proprietary information of Simpact,
* Inc, ("Simpact") and is protected by copyright, trade secret and other state
* and federal laws. The possession or receipt of this information does not
* convey any right to disclose its contents, reproduce it, or use, or license
* the use, for manufacture or sale, the information or anything described
* therein. Any use, disclosure, or reproduction without Simpact's prior
* written permission is strictly prohibited.
*
* Software and Technical Data Rights
*
* Simpact software products and related documentation will be furnished
* hereunder with "Restricted Rights" in accordance with:
*
*      A. Subparagraph (c)(1)(ii) of the clause entitled Rights in Technical
*      Data and Computer Software (OCT 1988) located at DFARS 252.227-7013; or
*
*      B. Subparagraph (c)(2) of the clause entitled Commercial Computer
*      Software - Restricted Rights (JUN 1987) located at FAR 52.227.19.
*****************************************************************************/

/*****************************************************************************
*    MODULE: cs_proto.h
*
*       Include header file.
*       Contains api function prototypes required by application programs
*
*    MODIFICATIONS:
*        Original 04/94
*
*****************************************************************************/

#ifdef  __cplusplus
extern "C" {
#endif

#ifdef __STDC__

int cs_accept(int client_id, int token, struct cs_qos_st *qos,
    int block_time);
int cs_connect(int client_id, char *dest, int dest_length,
    struct cs_qos_st *qos, struct cs_qos_st *ret_qos, int block_time);
int cs_connect_nb_remote(int client_id, struct cs_qos_st *ret_qos,int
block_time);
int cs_deregister(int client_id, int block_time);
int cs_disconnect(int client_id, struct cs_qos_st *qos, int block_time);
int cs_listen(int *client_id_array, int client_id_length,
    struct cs_listen_st *ret_ind, struct cs_qos_st *qos, int block_time);
int cs_redirect(int client_id, int token, struct cs_qos_st *qos,
    int block_time);
int cs_refuse(int client_id, int token, struct cs_qos_st *qos,
    int block_time);
int cs_register(int client_id, struct cs_qos_st *filter, int block_time);
int cs_init(char *config_file, void (*int_func)(int,int,int));
int cs_attach(char *circuit_id, int block_time);
int cs_bind(int client_id, int sap, int block_time);
int cs_detach(int client_id, int block_time);
int cs_unbind(int client_id, int block_time);
void cs_sleep(int duration);
int cs_read(int client_id, char *buf, int buf_length, int proto_flag,
```

```
    int *ret_flags, int block_time);
int cs_reset(int client_id, struct cs_qos_st *rst, int block_time);
int cs_select(int *client_id_array, int client_id_length, int *read_array,
    int *ind_array, int block_time);
int cs_write(int client_id, char * buf, int buf_length, int proto_flag,
    int block_time);
void cs_terminate(void);
int cs_bufsize(int cid);
#ifdef WINNT
DWORD cs_getpid(void);
#else
int cs_getpid(void);
#endif
int cs_config(int client_id, struct ppa_struct *rec);
int cs_suspend_events(void);
int cs_resume_events(int num);
void cs_suicide(void);
int cs_gen_event(int client, int event, char *buf, int len);
char *cs_sperror(int errnum);
#if defined(VXWORKS) | defined(__MSDOS__) | defined(WINNT)
int debuglog(const char *format,...);
#endif

#else
int cs_accept();
int cs_connect();
int cs_connect_nb_remote();
int cs_deregister();
int cs_disconnect();
int cs_listen();
int cs_redirect();
int cs_refuse();
int cs_register();
int cs_init();
int cs_attach();
int cs_bind();
int cs_detach();
int cs_unbind();
void cs_sleep();
int cs_read();
int cs_reset();
int cs_select();
int cs_write();
void cs_terminate();
int cs_bufsize();
#ifdef WINNT
DWORD cs_getpid();
#else
int cs_getpid();
#endif
int cs_config();
int cs_suspend_events();
int cs_resume_events();
void cs_suicide();
int cs_gen_event();
char *cs_sperror();
int debuglog();

#endif

#ifdef  __cplusplus
}
#endif
```

A.6 cs_x25.h

```
/*************************************************************************
*               CONFIDENTIAL & PROPRIETARY INFORMATION
*              Distribution to Authorized Personnel Only
*              Unpublished/Copyright 1997 Simpact, Inc.
*                       All Rights Reserved
*
* This document contains confidential and proprietary information of Simpact,
* Inc, ("Simpact") and is protected by copyright, trade secret and other state
* and federal laws. The possession or receipt of this information does not
* convey any right to disclose its contents, reproduce it, or use, or license
* the use, for manufacture or sale, the information or anything described
* therein. Any use, disclosure, or reproduction without Simpact's prior
* written permission is strictly prohibited.
*
* Software and Technical Data Rights
*
* Simpact software products and related documentation will be furnished
* hereunder with "Restricted Rights" in accordance with:
*
*      A. Subparagraph (c)(1)(ii) of the clause entitled Rights in Technical
*      Data and Computer Software (OCT 1988) located at DFARS 252.227-7013; or
*
*      B. Subparagraph (c)(2) of the clause entitled Commercial Computer
*      Software - Restricted Rights (JUN 1987) located at FAR 52.227.19.
*************************************************************************/

/*************************************************************************
*    MODULE: cs_x25.h
*
*        Include header file.
*        Contains low-level X.25 header file required by CS_API
*        and selected applications.
*
*    MODIFICATIONS:
*        Replaces control.h include file.
*
*************************************************************************/
/*
*       THE FOLLOWING PACKET COMMAND TYPES SUPPORT
*       CLIENT SERVICE ACCESS POINT (SAP) SESSION MANAGEMENT.
*/
#define HOPEN_SESSION   -1          /* Client Open Session Request  */
#define IOPEN_SESSION   -2          /* Server Session Opened        */
#define HCLOSE_SESSION  -3          /* Client Close Session Request */
#define ICLOSE_SESSION  -4          /* Server Session Closed        */

/*
*       THE FOLLOWING PACKET TYPES ARE VALID FOR X.25 1984
*       AND ITS ASSOCIATED SUB-PROTOCOLS (MLP and HDLC LAPB).
*/
#define HCALL           1           /* HOST Call Request            */
#define ICALL           2           /* ICP Incoming Call            */
#define HCONNECT        3           /* HOST Call Accepted           */
#define ICONNECT        4           /* ICP Call Connected           */
#define HHANGUP         5           /* HOST Clear Request           */
#define IHANGUP         6           /* ICP Clear Indication         */
#define HTONE           7           /* HOST Clear Confirmation      */
#define ITONE           8           /* ICP Clear Confirmation       */
#define HRSET           9           /* HOST Reset Request           */
#define IRSET           10          /* ICP Reset Indication         */
#define HRSETC          11          /* HOST Reset Confirmation      */
#define IRSETC          12          /* ICP Reset Confirmation       */
#define HINIT_MLP       13          /* HOST Reset MLP               */
#define HINIT_SLP       HINIT_MLP   /* HOST Reset SLP               */
#define HUNDATA         14          /* HOST Unnumbered Data         */
```

```
#define HSTATS_32BIT_SAMPLE 15      /* HOST Sample 32-Bit Statistics
                                       (NO CLEAR) */
#define ISTATS_32BIT    16          /* ICP 32-Bit Statistics */
#define HDATA           17          /* HOST Data Packet             */
#define IDATA           18          /* ICP Data Packet              */
#define HINT            19          /* HOST Interrupt               */
#define IINT            20          /* ICP Interrupt                */
#define HINTC           21          /* HOST Interrupt Confirmation  */
#define IINTC           22          /* ICP Interrupt Confirmation   */
#define HENABLE         23          /* HOST Enable Comm-Link        */
#define IENABLE         24          /* ICP Link Active              */
#define HDISABLE        25          /* HOST Disable Comm-Link       */
#define IDISABLE        26          /* ICP Link Inactive            */
#define HCONFIG         27          /* HOST Configure Comm-Link     */
#define IROTATE         28          /* ICP Rotate Transmit Window   */

#define HREG_ICF        29          /* Client Register Incoming Call Filter */
#define IGLITCH         30          /* MLP/SLP Reset                */
#define HDEL_ICF        31          /* Client Delete Incoming Call Filter   */
#define IERROR          32          /* ICP Station Procedure Error  */
#define HSTATS          33          /* HOST Read 16-Bit Statistics (AND CLEAR) */
#define ISTATS          34          /* ICP 16-Bit Statistics        */
#define HREJECT         35          /* HOST Configure IREJECT Format */
#define IREJECT         36          /* ICP Command Reject           */
#define HBUFI           37          /* HOST Configure Buffer        */
#define IBUFIC          38          /* ICP Buffer Confirmation      */
/*                      39             HOST Reserved (Unused)       */
#define IABORT          40          /* ICP Abort                    */
#define HABORT          41          /* HOST Abort                   */
#define ISTAOK          42          /* ICP Station Ok               */
#define HCSCON          43          /* HOST Configure Call Service  */
#define ISTAFAIL        44          /* ICP Station Failure          */
#define HCONMLP         45          /* HOST Configure MLP           */
#define IAUTO           46          /* ICP Autoconnect              */
#define HREDIRECT       47          /* Client Redirect Incoming Call */
#define ITIMOUT         48          /* ICP Timeout on command from  */
#define HSTATES         49          /* HOST State Request           */
#define IDIAG           50          /* ICP Diagnostic               */
#define HSTATS_32BIT    51          /* HOST Read 32-Bit Statistics
                                       (AND CLEAR) */
#define ISTATES         52          /* ICP Station/Link States      */
#define HMONITOR        53          /* HOST Monitor Control Request */
#define IMONITOR        54          /* ICP Monitor Data Report      */
#define HVERSION        55          /* HOST Version Request         */
#define IVERSION        56          /* ICP Version                  */
#define HSTATS_CLEAR    57          /* HOST Clear 16-Bit/32-Bit Statistics
                                       (NO READ) */
/*                      58             ICP Reserved (Unused)        */
#define HSTATS_SAMPLE   59          /* HOST Sample 16-Bit Statistics
                                       (NO CLEAR) */
#define INOSTA          60          /* ICP No Stations On Link      */
#define HADJUST_FLOW    61          /* HOST Adjust Control          */
#define IACKNOWLEDGE    62          /* ICP acknowledges API request */
#define HOPEN_PVC       63          /* Client Open PVC Request      */
#define IOPEN_PVC       64          /* Server PVC Opened            */
#define HCLOSE_PVC      65          /* Client Close PVC Request     */
#define ICLOSE_PVC      66          /* Server PVC Closed            */
/*                      67             HOST Reserved (Unused)       */
/*                      68             ICP Reserved (Unused)        */
#define HCLSTATE        69          /* HOST read control line state */
#define ICLSTATE        70          /* ICP control line state (CTS/DCD) */
#define HREGRQ          71          /* Host registration request    */
#define IREGCON         72          /* ICP registration confirmation */
#define ISUCCESS        74          /* SLP transmission successful  */
#define IFAILURE        76          /* SLP transmission failed      */
#define HTEST           77          /* HOST TEST Frame Data         */
#define ITEST           78          /* ICP TEST Frame Data          */
#define HROTATE         79          /* HOST safe store ack          */
```

```
#define IUNDATA         80          /* ICP Unnumbered Data          */
#define HBUFCLEAR       81          /* HOST buffer clearing option  */

/*
* Baud rate selection values
*/
#define BAUD_MASK       0x0F        /* Baud rate mask (Bits 0-3)    */

                                    /* LOW SPEED SELECTIONS (Bit7 = 0) */
                                    /* Selections 0-4 invalid       */
#define BAUD_300        5           /* Baud rate selection (300)    */
#define BAUD_600        6           /* Baud rate selection (600)    */
#define BAUD_1200       7           /* Baud rate selection (1200)   */
#define BAUD_2400       8           /* Baud rate selection (2400)   */
#define BAUD_4800       9           /* Baud rate selection (4800)   */
#define BAUD_9600       10          /* Baud rate selection (9600)   */
#define BAUD_19200      11          /* Baud rate selection (19200)  */
#define BAUD_38400      12          /* Baud rate selection (38400)  */
#define BAUD_56000      13          /* Baud rate selection (56000)  */
#define BAUD_57600      14          /* Baud rate selection (57600)  */
#define BAUD_64000      15          /* Baud rate selection (64000)  */

                                    /* HIGH SPEED SELECTIONS (Bit7 = 1) */
#define BAUD_73728      0           /* Baud rate selection (73728)  */
#define BAUD_76800      1           /* Baud rate selection (76800)  */
#define BAUD_92160      2           /* Baud rate selection (92160)  */
#define BAUD_115200     3           /* Baud rate selection (115200) */
#define BAUD_122800     4           /* Baud rate selection (122800) */
#define BAUD_153600     5           /* Baud rate selection (153600) */
#define BAUD_184320     6           /* Baud rate selection (184320) */
#define BAUD_230400     7           /* Baud rate selection (230400) */
#define BAUD_307200     8           /* Baud rate selection (307200) */
/*
* Miscellaneous HCONFIG flag definitions
*/
#define DCE_ADDRESS     BIT4        /* HCONFIG link 1st data word   */
#define DCE_SABM        BIT5        /* HCONFIG link 1st data word   */
#define EXT_CLOCK       BIT6        /* HCONFIG link 1st data word   */
#define HIGH_SPEED      BIT7        /* HCONFIG link 1st data word   */

#define PVC_STATION     BIT12       /* HCONFIG station 1st data word */
/*
* Link configuration function codes
*/
#define SFWSIZE         1           /* Set frame window size        */
#define SDATSIZE        2           /* Set maximum frame size for link */
#define ST1TIME         3           /* Set T1 timer value           */
#define SN2             4           /* Set N2 (retry) value         */
#define ENCODING        5           /* Set bit encoding format      */
#define DATARATE        6           /* Select custom data rate      */
#define ST2TIME         7           /* Set T2 timer value           */
#define LAPB_MODULUS    8           /* Set HDLC LAPB modulus        */
#define CUSTOM_ADDRESS  9           /* Set HDLC custom addressing   */
#define INTEGRITY_TIMER 10          /* Set T4 integrity check timer */
#define IDLE_TIMER      11          /* Set T3 idle link timer       */
#define XMIT_CLOCK      12          /* Select ext. clock source (TC, RC) */
#define OPTION_SREJ     13          /* Enable SREJ frame support    */
#define OPTION_RAW      14          /* Enable RAW SDLC (no protocol) */

/*
* Station configuration function codes
*/
#define SPWSIZE         1           /* Set packet window size       */
#define SROTATE         2           /* Set flag to control IROTATEs */
#define SFLOW_CONTROL   3           /* Set flag to control IDATAs   */
#define SHOST_SAFE_STOR 4           /* Set flag to control IDATA acks */
                                    /* over the line until host has stored*/
```

Appendix

B Sample Programs

This appendix contains a pair of sample programs that connect and transfer 10 messages back and forth.

The pasv.c program shown in Section B.1 listens for a connection call and counts the number of messages transferred. It disconnects the connection after it has returned 10 messages.

The actv.c program shown in Section B.2 initiates the connection and message transfers. It remains active until it receives a disconnect command from the pasv.c program.

B.1 pasv.c

```
/*************************************************************************
*  Passive connection program. It receives and sends 10 messages then     *
*  disconnects.                                                           *
*************************************************************************/
#include <cs_api.h>
#include <stdio.h>

#define INACTIVE 0
#define BOUND    1

int active = 10;
int status = INACTIVE;

/*-----------------------------------*/
/* event handler for non-blocking I/O */
/*-----------------------------------*/
void event_handler(int client_id, int event, int data_flag)
{
    int ret;
    char buf[80];
    struct cs_listen_st ret_ind;

    /*--------------------------------*/
    /* log the event to the debug log */
    /*--------------------------------*/
    debuglog("event_handler called for client_id(%d), event(%d), data(%d)",
        client_id, event, data_flag);

    switch (event)
    {
        case CS_ATTACH_SUCCESS:
            /*---------------------------*/
            /* attach complete, now bind */
            /*---------------------------*/
             cs_bind(client_id,0,0);
            break;
```

```
case CS_BIND_SUCCESS:
    /*-------------------------------------------------*/
    /* bind complete, now register for incommng calls  */
    /*-------------------------------------------------*/
    cs_register(client_id,NULL,0);
    status = BOUND;
     break;

case CS_REG_SUCCESS:
case CS_ACCEPT_SUCCESS:
    break;

case CS_INC_CALL:
    /*-----------------------------------------------------------*/
    /* got an incoming call, accept it (since no requirements  */
    /* were registered with cs_register, we will get an        */
    /* automatic connect                                       */
    /*-----------------------------------------------------------*/
    cs_listen(&client_id,1,&ret_ind,NULL,0);
    cs_accept(client_id,ret_ind.token,NULL,0);
    break;

 case CS_AUTO_CONNECT:
     break;

case CS_READ_COMPLETE:
    /*--------------------------------------*/
    /* got a read, go get it and write again */
    /*--------------------------------------*/
    cs_read(client_id, buf, 80, event, &ret, 0);
    cs_write(client_id,"Hello",5,0,0);

    /*-------------------------------------------------*/
    /* decrement the active count and maybe disconnect */
    /*-------------------------------------------------*/
    active--;
    if (active <= 1)
        cs_disconnect(client_id, NULL, 0);
    break;

 case CS_WRITE_COMPLETE:
    break;

case CS_HANGUP:
```

```
        case CS_DISCONN_SUCCESS:
            /*------------------------------*/
            /* lost the connection, shutdown */
            /*------------------------------*/
            active = 0;
            break;

        default:
            /*-------------------------------------------*/
            /* some kind of unknown event, print the data */
            /*-------------------------------------------*/
            if (data_flag)
            {
                cs_read(client_id, buf, 80, event, &ret, 0);
                debuglog("Command(%d) failed, event(%d), data:",ret,event);
                debuglog("%s",buf);
            }
            active = 0;
            break;

    }   /* switch */
}

/*---------------------*/
/* main system routine */
/*---------------------*/
main()
{
    int client_id;

    /*----------------------*/
    /* initialize the system */
    /*----------------------*/
    if (!cs_init("cs_config",event_handler))
    {
        /*----------------------*/
        /* attach to the Freeway */
        /*----------------------*/
        cs_suspend_events();
        client_id = cs_attach("test1",0);
        if (client_id > 0)
            active = 10;
        cs_resume_events(1);


        while (active > 0);

        if (status == BOUND)
             cs_unbind(client_id,2000);
        if (client_id >= 0)
             cs_detach(client_id,2000);

        cs_terminate();
    }
```

```
    return(0);
}
```

B.2 actv.c

```
/*************************************************************************
*  Active connection program. It sends and receives messages until the    *
*  passive program disconnects.                                           *
*************************************************************************/
#include <cs_api.h>
#include <stdio.h>

#define INACTIVE 0
#define BOUND    1

int active = 0;
int status = INACTIVE;

/*-----------------------------------*/
/* event handler for non-blocking I/O */
/*-----------------------------------*/
void event_handler(int client_id, int event, int data_flag)
{
    int ret;
    char buf[80];
    struct cs_listen_st ret_ind;

    /*--------------------------------*/
    /* log the event to the debug log */
    /*--------------------------------*/
    debuglog("event_handler called for client_id(%d), event(%d), data(%d)",
        client_id, event, data_flag);

    switch (event)
    {
        case CS_ATTACH_SUCCESS:
            /*---------------------------*/
            /* attach complete, now bind */
            /*---------------------------*/
            cs_bind(client_id,0,0);
            break;

        case CS_BIND_SUCCESS:
            /*----------------------------------*/
            /* bind complete, now try to connect */
            /*----------------------------------*/
            cs_connect(client_id,"0",1,NULL,NULL,0);
             status = BOUND;
            break;

        case CS_CONN_SUCCESS:
            /*----------------------------------*/
            /* connect complete, write a message */
            /*----------------------------------*/
            cs_connect_nb_remote(client_id,NULL,0);
            cs_write(client_id,"Hello",5,0,0);
            break;
```

```
         case CS_WRITE_COMPLETE:
             break;

        case CS_READ_COMPLETE:
            /*-------------------------------------*/
            /* got a read, go get it and write again */
            /*-------------------------------------*/
            cs_read(client_id, buf, 80, event, &ret, 0);
            cs_write(client_id,"Hello",5,0,0);
            break;

        case CS_HANGUP:
            /*-------------------------------------*/
            /* clear active flag so we can shutdown */
            /*-------------------------------------*/
            active = 0;
            break;

        default:
            /*-------------------------------------------*/
            /* some kind of unknown event, print the data */
            /*-------------------------------------------*/
            if (data_flag)
            {
                cs_read(client_id, buf, 80, event, &ret, 0);
                debuglog("Command(%d) failed, event(%d), data:",ret,event);
                debuglog("%s",buf);
            }
            active = 0;
            break;

    }   /* switch */
}

/*---------------------*/
/* main system routine */
/*---------------------*/
main()
{
    int client_id;

    /*-----------------------*/
    /* initialize the system */
    /*-----------------------*/
    if (!cs_init("cs_config",event_handler))
    {
        /*-----------------------*/
        /* attach to the Freeway */
        /*-----------------------*/
        cs_suspend_events();
        client_id = cs_attach("test2",0);
        if (client_id > 0)
            active = 1;
       cs_resume_events(1);
```

```
        /*----------------------------*/
        /* loop until time to shutdown */
        /*----------------------------*/
        while (active);

        if (status == BOUND)
             cs_unbind(client_id,2000);
        if (client_id >= 0)
             cs_detach(client_id,2000);

        cs_terminate();
    }

    return(0);
}
```

Appendix

C X.25 Diagnostic Codes

Table C–1 shows the meaning of various X.25 diagnostic codes associated with the quality of service item HF_DIAG. Not all diagnostic codes need apply to a specific network, but those used are as coded in the table. A given diagnostic need not apply to all packet types (that is, reset indication, clear indication, restart indication, registration confirmation, and diagnostics packets).

The first diagnostic in each group is a generic diagnostic and can be used in place of the more specific diagnostics within the grouping. The decimal 0 diagnostic code can be used in situations where no additional information is available.

Table C–1: X.25 Diagnostic Codes for qos Item HF_DIAG

Diagnostic Description	Decimal Value	Hexadecimal Value
No additional information	0	0
Invalid P(S)	1	1
Invalid P(R)	2	2
Not defined	3–15	3–F
Packet type invalid	16	10
For state r1	17	11
For state r2	18	12
For state r3	19	13
For state p1	20	14
For state p2	21	15
For state p3	22	16
For state p4	23	17
For state p5	24	18
For state p6	25	19
For state p7	26	1A
For state d1	27	1B
For state d2	28	1C
For state d3	29	1D
Not defined	30–31	1E–1F
Packet not allowed	32	20
Unidentifiable packet	33	21
Call on one-way logical channel	34	22
Invalid packet type on a permanent virtual circuit	35	23
Packet on assigned logical channel	36	24
Reject not subscribed to	37	25
Packet too short	38	26
Packet too long	39	27
Invalid general format identifier	40	28

Table C–1: X.25 Diagnostic Codes for qos Item HF_DIAG (*Cont'd*)

Diagnostic Description	Decimal Value	Hexadecimal Value
Restart or registration packet with nonzero in bits 1 to 4 of octet 1, or bits 1 to 8 of octet 2	41	29
Packet type not compatible with facility	42	2A
Unauthorized interrupt confirmation	43	2B
Unauthorized interrupt	44	2C
Unauthorized reject	45	2D
Not defined	46–47	2E–2F
Time expired	48	30
For incoming call	49	31
For clear indication	50	32
For reset indication	51	33
For restart indication	52	34
Not defined	53–63	35–3F
Call set up, call clearing, or registration problem	64	40
Facility/registration code not allowed	65	41
Facility parameter not allowed	66	42
Invalid called address	67	43
Invalid calling address	68	44
Invalid facility/registration length	69	45
Incoming calls barred	70	46
No logical channel available	71	47
Call collision	72	48
Duplicate facility requested	73	49
Nonzero address length	74	4A
Nonzero facility length	75	4B
Facility not provided when expected	76	4C
Invalid CCITT-specified DTE facility	77	4D
Not defined	78–79	4E–4F

Table C–1: X.25 Diagnostic Codes for qos Item HF_DIAG (*Cont'd*)

Diagnostic Description	Decimal Value	Hexadecimal Value
Miscellaneous	80	50
Improper cause code from DTE	81	51
Not aligned octet	82	52
Inconsistent Q-bit setting	83	53
Not defined	84–95	54–5F
Not assigned	96–111	60–6F
International problem	112	70
Remote network problem	113	71
International protocol problem	114	72
International link out of order	115	73
International link busy	116	74
Transit network facility problem	117	75
Remote network facility problem	118	76
International routing problem	119	77
Temporary routing problem	120	78
Unknown called DNIC	121	79
Maintenance action[a]	122	7A
Not defined	123–127	7B–7F
Reserved for network-specific diagnostic information	128–255	80–FF

[a] This diagnostic may also apply to a maintenance action within a national network.

Appendix

D X.25 Packet Types Cross Reference

Table D–1 shows the cross reference between the short packet names in the cs_x25.h file and the DLI names used in the *X.25 Low-Level Interface* document.

Table D–1: X.25 Packet vs. DLI Packet Cross Reference

X.25 Packet	Value	DLI Packet
HOPEN_SESSION	-1	DLI_X25_HOST_OPEN_SESSION_REQ
IOPEN_SESSION	-2	DLI_X25_ICP_SESSION_OPENED
HCLOSE_SESSION	-3	DLI_X25_HOST_CLOSE_SESSION_REQ
ICLOSE_SESSION	-4	DLI_X25_ICP_SESSION_CLOSED
HCALL	1	DLI_X25_HOST_CALL_REQ
ICALL	2	DLI_X25_ICP_INCOMING_CALL
HCONNECT	3	DLI_X25_HOST_CALL_ACCEPTED
ICONNECT	4	DLI_X25_ICP_CALL_ACCEPTED
HHANGUP	5	DLI_X25_HOST_CLR_REQ
IHANGUP	6	DLI_X25_ICP_CLR_INDICATION
HTONE	7	DLI_X25_HOST_CLR_CONFIRMED
ITONE	8	DLI_X25_ICP_CLR_CONFIRMED
HRSET	9	DLI_X25_HOST_RESET_REQ
IRSET	10	DLI_X25_ICP_RESET_INDICATION
HRSETC	11	DLI_X25_HOST_RESET_CONFIRMED
IRSETC	12	DLI_X25_ICP_RESET_CONFIRMED
HINIT_MLP	13	DLI_X25_HOST_INIT_MLP
HINIT_SLP	HINIT_MLP	DLI_X25_HOST_INIT_SLP

Table D–1: X.25 Packet vs. DLI Packet Cross Reference (*Cont'd*)

X.25 Packet	Value	DLI Packet
HUNDATA	14	DLI_X25_HOST_UNNUMBERED_DATA
HSTATS_32BIT_SAMPLE	15	DLI_X25_HOST_32BIT_SAMPLE_STATISTICS
ISTATS_32BIT	16	DLI_X25_ICP_32BIT_STATISTICS
HDATA	17	DLI_X25_HOST_DATA
IDATA	18	DLI_X25_ICP_DATA
HINT	19	DLI_X25_HOST_INTERRUPT
IINT	20	DLI_X25_ICP_INTERRUPT
HINTC	21	DLI_X25_HOST_INT_CONFIRMED
IINTC	22	DLI_X25_ICP_INT_CONFIRMED
HENABLE	23	DLI_X25_HOST_ENABLE_LINK
IENABLE	24	DLI_X25_ICP_LINK_ENABLED
HDISABLE	25	DLI_X25_HOST_DISABLE_LINK
IDISABLE	26	DLI_X25_ICP_LINK_DISABLED
HCONFIG	27	DLI_X25_HOST_CFG_LINK
	27	DLI_X25_HOST_CFG_STATION
IROTATE	28	DLI_X25_ICP_ROTATE_XMIT_WINDOW
HREG_ICF	29	DLI_X25_HOST_ADD_INCALL_FILTER
IGLITCH	30	DLI_X25_ICP_MLP_SLP_RESET
HDEL_ICF	31	DLI_X25_HOST_DEL_INCALL_FILTER
IERROR	32	DLI_X25_ICP_ERROR
HSTATS	33	DLI_X25_HOST_GET_STATISTICS
ISTATS	34	DLI_X25_ICP_STATISTICS
HREJECT	35	DLI_X25_HOST_CFG_IREJECT_FORMAT
IREJECT	36	DLI_X25_ICP_CMD_REJECTED
HBUFI	37	DLI_X25_HOST_CFG_BUF
IBUFIC	38	DLI_X25_ICP_CFG_BUF_CONFIRMED
	39	
IABORT	40	DLI_X25_ICP_ABORT
HABORT	41	DLI_X25_HOST_ABORT
ISTAOK	42	DLI_X25_ICP_STATION_OK
HCSCON	43	DLI_X25_HOST_CFG_CALL_SERVICE

Table D–1: X.25 Packet vs. DLI Packet Cross Reference (*Cont'd*)

X.25 Packet	Value	DLI Packet
ISTAFAIL	44	DLI_X25_ICP_STATION_FAILED
HCONMLP	45	DLI_X25_HOST_CFG_MLP
IAUTO	46	DLI_X25_ICP_AUTO_CONNECT
HREDIRECT	47	DLI_X25_HOST_REDIRECT
ITIMOUT	48	DLI_X25_ICP_CMD_TIMEOUT
HSTATES	49	DLI_X25_HOST_GET_STATE
IDIAG	50	DLI_X25_ICP_DIAGNOSTICS
HSTATS_32BIT	51	DLI_X25_HOST_32BIT_GET_STATISTICS
ISTATES	52	DLI_X25_ICP_STATION_LINK_STATES
HMONITOR	53	DLI_X25_HOST_MONITOR_REQ
IMONITOR	54	DLI_X25_ICP_MONITOR_RSP
HVERSION	55	DLI_X25_HOST_GET_VERSION
IVERSION	56	DLI_X25_ICP_VERSION
HSTATS_CLEAR	57	DLI_X25_HOST_CLR_STATISTICS
	58	
HSTATS_SAMPLE	59	DLI_X25_HOST_SAMPLE_STATISTICS
INOSTA	60	DLI_X25_ICP_NO_STATIONS_ON_LINK
HADJUST_FLOW	61	DLI_X25_HOST_ADJUST_FLOW_CTRL
IACKNOWLEDGE	62	DLI_X25_ICP_ACK
HOPEN_PVC	63	DLI_X25_HOST_OPEN_PVC
IOPEN_PVC	64	DLI_X25_ICP_PVC_OPENED
HCLOSE_PVC	65	DLI_X25_HOST_CLOSE_PVC
ICLOSE_PVC	66	DLI_X25_ICP_PVC_CLOSED
	67	
	68	
HCLSTATE	69	DLI_X25_HOST_CTL_LINE_STATE_REQ
ICLSTATE	70	DLI_X25_ICP_CTL_LINE_STATE_RSP
HREGRQ	71	DLI_X25_HOST_REGISTER
IREGCON	72	DLI_X25_ICP_REGISTERED
	73	
ISUCCESS	74	DLI_X25_ICP_SLP_XMIT_OK

Table D–1: X.25 Packet vs. DLI Packet Cross Reference (*Cont'd*)

X.25 Packet	Value	DLI Packet
	75	
IFAILURE	76	DLI_X25_ICP_SLP_XMIT_ERROR
HTEST	77	DLI_X25_HOST_TEST_FRAME
ITEST	78	DLI_X25_ICP_TEST_FRAME
HROTATE	79	DLI_X25_HOST_ROTATE
IUNDATA	80	DLI_X25_ICP_UNNUMBERED_DATA
HBUFCLEAR	81	DLI_X25_HOST_BUF_CLR

Glossary

boot server — A client computer that downloads software onto Freeway (that is, "boots" Freeway). During this operation, Freeway becomes a client of the boot server.

CCITT — Consultative Committee of International Telephone and Telegraph

client — An entity on the LAN that uses the services offered by Freeway. To conform with the industry use of this term, a client refers to an application program which is running on a host computer somewhere on the network and communicates with Freeway through a LAN connection. Freeway supports clients on a number of different types of host computers. See also "process."

CPU — Central processing unit

CS API — A call service application program interface provides a programming library of routines to facilitate data transfer to and from Freeway using a standard interface across protocols.

CTS — Clear to send

CUG — Closed user group

D-bit — Delivery confirmation bit, used in X.25 data packets

DCD	Data carrier detect
DCE	Data circuit-terminating equipment
DDN	Defense data network
DNIC	Data network identification code
DTE	Data terminal equipment
Freeway	Freeway refers to the entire Freeway product in terms of hardware and software. The hardware includes items such as the server processor board and/or the ICPs. The software includes functions such as the APIs, Freeway management services, protocol services, and protocol software executing on the ICPs.
HDLC	High-level data link control
HIC	Highest incoming channel
HOC	Highest outgoing channel
HTC	Highest two-way channel
ICF	Incoming call filter
ICP	Protogate's intelligent communications processor (ICP) board that supports serial protocols. Freeway currently supports Protogate's ICP2432 and ICP2432B PCI-bus processors. An ICP is also referred to as a "WAN interface processor."
ICP-resident software	Protogate-supplied communication protocol software or user-customized software that runs on the ICP to process the data stream between the ICP and the WAN devices. Refer to the

	Freeway Protocol Software Toolkit Programmer's Guide for customized software.
IP	Internet protocol, described by RFC-791
ISO	International Standards Organization
LAN	Local area network
LAN protocol	The hardware and software which comprise the LAN and form the basis of communications between Freeway servers and clients. An example LAN protocol is TCP/IP running over Ethernet.
LAPB	Link access procedure balanced
LCN	Logical channel number
LIC	Lowest incoming channel
LOC	Lowest outgoing channel
LTC	Lowest two-way channel
M-bit	More data bit, used in X.25 data packets
MLP	Multilink procedure; uses multiple SLPs
MOD 8	Modulo 8 sequence numbers range from 0 through 7
MOD 128	Modulo 128 sequence numbers range from 0 through 127
operating system	Code that provides the necessary scheduling and management functions for tasks and services. The VxWorks operating system runs on the server processor board, and Protogate's OS/Impact real-time executive runs on the ICPs.

PPA	Physical point of attachment
process	Code executing on a LAN-based host computer and equivalent to the "client" term.
PVC	Permanent virtual circuit
P(R)	Packet receive sequence number field, ranges from 0 through 7 for MOD 8 operation or 0 through 127 for MOD 128 operation
P(S)	Packet send sequence number field, ranges from 0 through 7 for MOD 8 operation or 0 through 127 for MOD 128 operation
Q-bit	Qualifier bit, used in X.25 packets
QOS	Quality of service
REJ	Reject, an HDLC frame or X.25 packet type used to request retransmission of data
resource	A resource available in the Freeway product. Typically, resource refers to an ICP board within Freeway, a port on a board, or a WAN protocol running on an ICP.
RPOA	Recognized private operating agency
SABM	Set asynchronous balanced mode, an HDLC frame type used to start or reset the data link
SAP	Service access point
server processor	The Freeway server processor board, which is capable of executing the server's operating system and functions such as server management.

server-resident software	Software that runs on the server processor board and processes the data stream between the LAN and WAN connections. Server-resident software can be either supplied by Protogate or customized by the user.
service	A more generalized view of resource where a grouping is made on a per-protocol basis. For example, the X.25 protocol has a number of resources: the ICP, the ports available on the ICP, and the protocol running on the ICP. Collectively, these resources describe the service.
SLP	Single-link procedure, a LAPB data link
SVC	Switched virtual circuit
task	"Task" is used to differentiate between code executing on a Freeway server processor board or on the ICP, and code executing on a client computer, which is referred to as a "process".
TCP/IP	Transmission-control protocol/internet protocol
TOA/NPI	Type of address/number plan identification
UA	Unnumbered acknowledgment, an HDLC frame type used to acknowledge start or reset of the data link
UNIX	An industry-standard operating system commonly used on client workstations
WAN	Wide area network
WAN interface processor	A board containing hardware, and possibly software, used to offer a particular communication protocol service. For instance, Protogate's ICP2432B is a WAN interface processor

	board that supports serial communication protocols such as X.25.
X.25	A packet-switching communications protocol

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