Freeway®
Transport Subsystem Interface
Reference Guide
DC 900-1386D
Contents

List of Figures .................................................. 7
List of Tables .................................................. 9
Preface ................................................................ 11

1 Overview ................................................................ 17
  1.1 Product Overview .............................................. 17
  1.1.1 Freeway Server ............................................. 18
  1.1.2 Embedded ICP .............................................. 18
  1.2 Freeway Client-Server Environment Using TSI .......... 22
  1.2.1 Establishing Freeway Internet Addresses ............... 23
  1.2.2 Defining the TSI Configuration ......................... 23
  1.2.3 Establishing a Freeway TSI Connection ................. 24
  1.2.4 Exchanging Data through the Freeway Message Multiplexer .... 24
  1.2.5 Closing a Freeway Session ............................ 24
  1.3 TSI Overview and Features ................................ 25

2 TSI Concepts ..................................................... 29
  2.1 Configuration in the Freeway Environment ............... 29
  2.2 Blocking versus Non-blocking I/O .......................... 30
  2.2.1 I/O Completion Handler for Non-Blocking I/O ......... 31
  2.3 Buffer Management ........................................... 31
  2.3.1 Overview of the Freeway System Buffer Relationships .... 32
    2.3.1.1 Example Calculation to Change ICP, Client, and Server Buffer Sizes .... 33
  2.3.2 Client TSI Buffer Configuration .......................... 35
    2.3.2.1 TSI Buffer Pool Definition .......................... 36
Freeway Transport Subsystem Interface Reference Guide

2.3.2.2 Connection-Specific Buffer Definition .................................. 38
2.3.2.3 TSI Buffer Size Negotiation ............................................. 39
2.3.3 Server TSI Buffer Configuration ........................................... 40
2.3.4 Buffer Allocation and Release ............................................. 40
2.3.5 Cautions for Changing Buffer Sizes ...................................... 41
2.3.6 Using Your Own Buffers ................................................... 41
2.3.7 Buffer Management (Client versus Server-Resident Applications) ... 43
2.4 System Resource Requirements ............................................... 44
  2.4.1 Signal Processing ......................................................... 45

3 TSI Configuration
  3.1 Configuration Process Overview ............................................ 47
  3.2 Introduction to TSI Configuration ......................................... 50
    3.2.1 TSI Configuration Language ......................................... 50
    3.2.2 Rules of the TSI Configuration File ................................ 51
    3.2.3 Binary Configuration File Management ................................ 52
    3.2.4 On-line Configuration File Processing .............................. 52
  3.3 TSI Connection Definition ................................................... 53
    3.3.1 Parameters for the "main" Definition ................................ 54
    3.3.2 Parameters for the Connection Definition (Non-transport Specific) 56
    3.3.3 Parameters for Connection Definition (TCP/IP Socket Transport) 58
    3.3.4 Parameters for Connection Definition (Shared-Memory Transport) 59
  3.4 Example TSI Configurations .................................................. 60
    3.4.1 TCP/IP Socket Transport Interface .................................. 60
    3.4.2 Shared-Memory Transport Interface (VxWorks Only) ............ 62
  3.5 Protagate's Freeway Server TSI Configuration .......................... 65
  3.6 Miscellaneous TSI Configuration Details ................................ 67
    3.6.1 TSI Configuration Error Messages .................................. 67
    3.6.2 Protagate Definition Language (PDL) Grammar ..................... 69

4 TSI Functions
  4.1 Overview of TSI Functions .................................................. 71
    4.1.1 TSI Error Handling ................................................... 71
    4.1.2 Categories of TSI Functions ......................................... 72
      4.1.2.1 Summary of TSI Functions .................................... 73
    4.1.3 TSI Data Structures ................................................... 76

DC 900-1386D
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.3.1 TSI System Configuration</td>
<td>76</td>
</tr>
<tr>
<td>4.1.3.2 TSI Connection Status</td>
<td>78</td>
</tr>
<tr>
<td>4.1.3.3 TSI Connection Definition List</td>
<td>78</td>
</tr>
<tr>
<td>4.2 tBufAlloc</td>
<td>80</td>
</tr>
<tr>
<td>4.3 tBufFree</td>
<td>82</td>
</tr>
<tr>
<td>4.4 tConnect</td>
<td>84</td>
</tr>
<tr>
<td>4.5 tDisconnect</td>
<td>88</td>
</tr>
<tr>
<td>4.6 tInit</td>
<td>91</td>
</tr>
<tr>
<td>4.7 tListen</td>
<td>95</td>
</tr>
<tr>
<td>4.8 tPoll</td>
<td>100</td>
</tr>
<tr>
<td>4.9 tPost</td>
<td>107</td>
</tr>
<tr>
<td>4.10 tRead</td>
<td>109</td>
</tr>
<tr>
<td>4.11 tSyncSelect</td>
<td>114</td>
</tr>
<tr>
<td>4.12 tTerm</td>
<td>117</td>
</tr>
<tr>
<td>4.13 tWrite</td>
<td>120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A TSI Common Error Codes</td>
<td>125</td>
</tr>
<tr>
<td>A.1 Internal Error Codes</td>
<td>125</td>
</tr>
<tr>
<td>A.2 Command-Specific Error Codes</td>
<td>137</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B UNIX, VxWorks, and VMS I/O</td>
<td>143</td>
</tr>
<tr>
<td>B.1 UNIX Environment</td>
<td>143</td>
</tr>
<tr>
<td>B.1.1 Blocking I/O operations</td>
<td>143</td>
</tr>
<tr>
<td>B.1.2 Non-Blocking I/O Operations</td>
<td>144</td>
</tr>
<tr>
<td>B.1.3 SOLARIS use of SIGALRM</td>
<td>144</td>
</tr>
<tr>
<td>B.1.4 Polling I/O Operations</td>
<td>145</td>
</tr>
<tr>
<td>B.2 VxWorks Environment</td>
<td>145</td>
</tr>
<tr>
<td>B.2.1 Blocking I/O Operations</td>
<td>146</td>
</tr>
<tr>
<td>B.2.2 Non-Blocking I/O Operations</td>
<td>146</td>
</tr>
<tr>
<td>B.3 VMS Environment</td>
<td>147</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C TSI Logging and Tracing</td>
<td>149</td>
</tr>
<tr>
<td>C.1 TSI Logging</td>
<td>149</td>
</tr>
<tr>
<td>C.2 TSI Tracing</td>
<td>150</td>
</tr>
<tr>
<td>C.2.1 Trace Definitions</td>
<td>150</td>
</tr>
<tr>
<td>C.2.2 Decoded Trace Layout</td>
<td>152</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1-1: Freeway Configuration ........................................... 19
Figure 1-2: Embedded ICP Configuration ................................... 20
Figure 1-3: A Typical Freeway Environment ............................. 23
Figure 1-4: TSI Environment ................................................. 26
Figure 1-5: TSI in the Freeway Operating Environment .................. 27
Figure 2-2: Client TSI Configuration File Changes ...................... 34
Figure 2-1: Client DLI Configuration File Changes (BSC Example) .... 34
Figure 2-3: Server MuxCfg TSI Configuration File Changes ............ 35
Figure 2-4: TSI Buffer Size Example .................................... 37
Figure 2-5: DLI Buffer Size Example .................................... 38
Figure 2-6: Comparison of malloc and tBufAlloc Buffers ............... 42
Figure 2-7: Using the malloc Function for Buffer Allocation ............ 42
Figure 3-1: TSI Architecture ................................................ 49
Figure 3-2: Example Configuration for TCP/IP Socket Transport Interface .......... 61
Figure 3-3: Example Configuration for Shared-Memory Transport Interface .......... 63
Figure 3-4: TSI Configuration File (MuxCfg) for Protogate Server-Resident TSI .... 66
Figure 4-1: TSI System Configuration Data Structure ..................... 76
Figure 4-2: TSI Connection Status Data Structure ....................... 78
Figure C-1: TSI Trace File Format ........................................ 158
Figure C-2: TRACE_FCB ‘C’ Structure .................................... 158
Figure C-3: TSI_TRACE_HDR ‘C’ Structure ................................ 159
List of Tables

Table 2-1: Required Values for Calculating New $\text{MaxBufSize}$ Parameter .......................... 34
Table 3-1: TSI Parameters for “main” Definition ................................................................. 54
Table 3-2: TSI Parameters for Non-Transport Specific Connection ................................. 56
Table 3-3: TSI Parameters for TCP/IP Socket Transport Connection ............................... 58
Table 3-4: TSI Parameters for Shared-Memory Transport Connection .............................. 59
Table 4-1: TSI Function Groups ......................................................................................... 72
Table 4-2: TSI Functions: Syntax and Parameters (Listed in Typical Call Order) .............. 74
Table 4-3: TSI System Configuration Data Structure Fields .............................................. 77
Table 4-4: TSI Connection Status Data Structure Fields .................................................... 79
Table A-1: TSI Command-specific Error Codes .................................................................. 137
Preface

Purpose of Document

This document describes Protogate's transport subsystem interface (TSI). The TSI helps you develop applications interfacing with a Freeway communications server or embedded intelligent communications processor (ICP).

Note

In this document, the term “Freeway” can mean either a Freeway server or an embedded ICP. For the embedded ICP, also refer to the user guide for your ICP and operating system (for example, the ICP2432 User Guide for Windows NT).

Intended Audience

This document should be read by application programmers. You should be familiar with the C programming language and have some knowledge of networks.

If your application will use Protogate's data link interface (DLI), which uses the TSI, you will need to use TSI only for configuration (see Chapter 3 of this document). Additionally, you should be familiar with the Freeway Data Link Interface Reference Guide.

If your application will interface directly to the TSI, you should be familiar with the programming details described in this document. If your TSI application will interface to one of Protogate's protocol services running on a Freeway ICP, you will also need to be
familiar with the Freeway Client-Server Interface Control Document and your particular protocol programmer’s guide.

**Organization of Document**

Chapter 1 is an overview of Freeway and the TSI.

Chapter 2 describes various TSI concepts that you should understand before writing an application program.

Chapter 3 describes the TSI configuration services.

Chapter 4 gives details of each TSI function.

Appendix A lists additional error codes not included in the reference sections. It also provides summary tables of all TSI error codes as they relate to specific TSI function calls.

Appendix B compares I/O handling in the UNIX, VMS, and VxWorks environments.

Appendix C describes the TSI logging and tracing capabilities.

**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 3100 Hardware Installation Guide
  DC-900-2002
- Freeway 3200 Hardware Installation Guide
  DC-900-2003
- Freeway 3400 Hardware Installation Guide
  DC-900-2004
- Freeway 3600 Hardware Installation Guide
  DC-900-2005
- 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 Electrical Interfaces (Addendum to DC-900-1501)
  DC-900-1566
- ICP2432 Hardware Installation Guide
  DC-900-1502

Freeway Software Installation and Configuration Support

- Freeway Message Switch User Guide
  DC-900-1588
- Freeway Release Addendum: Client Platforms
  DC-900-1555
- Freeway User Guide
  DC-900-1333
- Freeway Loopback Test Procedures
  DC-900-1533

Embedded ICP Software Installation and Programming Support

- ICP2432 User Guide for Digital UNIX
  DC-900-1513
- ICP2432 User Guide for OpenVMS Alpha
  DC-900-1511
- ICP2432 User Guide for OpenVMS Alpha (DLITE Interface)
  DC-900-1516
- ICP2432 User Guide for Solaris STREAMS
  DC-900-1512
- ICP2432 User Guide for Windows NT
  DC-900-1510
- ICP2432 User Guide for Windows NT (DLITE Interface)
  DC-900-1514

Application Program Interface (API) Programming Support

- Freeway Data Link Interface Reference Guide
  DC-900-1385
Document Conventions

This document follows the most significant byte first (MSB) and most significant word first (MSW) conventions for bit-numbering and byte-ordering. In all packet transfers
between the client applications and the ICPs, the ordering of the byte stream is preserved.

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.

Revision History

The revision history of the Freeway Transport Subsystem Interface Reference Guide, Protogate document DC 900-1386D, is recorded below:

<table>
<thead>
<tr>
<th>Document Revision</th>
<th>Release Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC 900-1386A</td>
<td>February 1997</td>
<td>Initial release using new document number: Include Release Notes for changes that potentially affect existing user applications.</td>
</tr>
<tr>
<td>Special Freeway Server 2.5 Release</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC 900-1386B</td>
<td>June 1998</td>
<td>Incorporate previous release notes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Modify Buffer Management Section 2.3 on page 31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Enhance error detection and reporting (Chapter 4 and Appendix A)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Regarding “Error Handling for Dead Socket Detection,” refer to the Freeway Data Link Interface Reference Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dual Ethernet support is now handled by the “Added Interfaces” configuration described in the Freeway User Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simpact’s browser interface is no longer supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modify Section 1.1 on page 17 for embedded ICPs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Add IsSyncSelect function (Section 4.11 on page 114)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Add new error codes to Appendix A</td>
</tr>
<tr>
<td>DC 900-1386C</td>
<td>December 1999</td>
<td>Add LocalPort TSI configuration parameter (Table 3–3 on page 58)</td>
</tr>
</tbody>
</table>
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.
This document describes Protogate's transport subsystem interface (TSI) to the Freeway communications server. The TSI presents a consistent, high-level, common interface across multiple clients, operating systems, and transport services. The TSI provides connection-oriented data services to your client application with a subroutine library. This library provides functions that permit your application to access, configure, establish and terminate connections, and exchange data with a TSI peer application. Within the Freeway server, the TSI is used by Freeway's message multiplexor (MsgMux) to communicate with client applications.

**Note**

In this document, the term “Freeway” can mean either a Freeway server or an embedded ICP. For the embedded ICP, also refer to the user guide for your ICP and operating system (for example, the Freeway User Guide).

### 1.1 Product Overview

Protogate provides a variety of wide-area network (WAN) connectivity solutions for real-time financial, defense, telecommunications, and process-control applications. Protogate's Freeway server offers flexibility and ease of programming using a variety of LAN-based server hardware platforms. Now a consistent and compatible embedded intelligent communications processor (ICP) product offers the same functionality as the Freeway server, allowing individual client computers to connect directly to the WAN.
Both Freeway and the embedded ICP use the same data link interface (DLI). Therefore, migration between the two environments simply requires linking your client application with the proper library. Various client operating systems are supported (for example, UNIX, VMS, and Windows NT).

Protogate protocols that run on the ICPs are independent of the client operating system and the hardware platform (Freeway or embedded ICP).

1.1.1 Freeway Server

Protogate’s Freeway communications servers enable client applications on a local-area network (LAN) to access specialized WANs through the DLI. The Freeway server can be any of several models (for example, Freeway 1100/1150, Freeway 1200/1300, Freeway 2000/4000, or Freeway 8000/8800). The Freeway server is user programmable and communicates in real time. It provides multiple data links and a variety of network services to LAN-based clients. Figure 1-1 shows the Freeway configuration.

To maintain high data throughput, Freeway uses a multi-processor architecture to support the LAN and WAN services. The LAN interface is managed by a single-board computer, called the server processor. It uses the commercially available VxWorks operating system to provide a full-featured base for the LAN interface and layered services needed by Freeway.

Freeway can be configured with multiple WAN interface processor boards, each of which is a Protogate ICP. Each ICP runs the communication protocol software using Protogate’s real-time operating system.

1.1.2 Embedded ICP

The embedded ICP connects your client computer directly to the WAN (for example, using Protogate’s ICP2432 PCIbus board). The embedded ICP provides client applications with the same WAN connectivity as the Freeway server, using the same data link interface (via the DLITE embedded interface). The ICP runs the communication protocol software using Protogate’s real-time operating system. Figure 1-2 shows the embedded ICP configuration.
Figure 1-1: Freeway Configuration
Figure 1-2: Embedded ICP Configuration
Summary of product features:

- Provision of WAN connectivity either through a LAN-based Freeway server or directly using an embedded ICP
- Elimination of difficult LAN and WAN programming and systems integration by providing a powerful and consistent data link interface
- Variety of off-the-shelf communication protocols available from Protogate which are independent of the client operating system and hardware platform
- Support for multiple WAN communication protocols simultaneously
- Support for multiple ICPs (two, four, eight, or sixteen communication lines per ICP)
- Wide selection of electrical interfaces including EIA-232, EIA-449, EIA-530, and V.35
- Creation of customized server-resident and ICP-resident software, using Protogate's software development toolkits
- Freeway server standard support for Ethernet and Fast Ethernet LANs running the transmission control protocol/internet protocol (TCP/IP)
- Freeway server standard support for FDDI LANs running the transmission control protocol/ internet protocol (TCP/IP)
- Freeway server management and performance monitoring with the simple network management protocol (SNMP), as well as interactive menus available through a local console, telnet, or rlogin
1.2 Freeway Client-Server Environment Using TSI

Freeway acts as a gateway that connects a client on a local-area network to a wide-area network. Through Freeway, a client application can exchange data with a remote data link application. Your client application must interact with the Freeway server and its resident ICPs before exchanging data with the remote data link application.

One of the major Freeway components is the message multiplexor (MsgMux) that manages the data traffic between the LAN and the WAN environments. The client application typically interacts with the Freeway MsgMux through a TCP/IP BSD-style socket interface (or a shared-memory interface if it is a server-resident application (SRA)). The ICPs interact with the MsgMux through the DMA and/or shared-memory interface of the industry-standard bus to exchange WAN data.

From the client application’s point of view, the complexities are handled through a simple and consistent transport subsystem interface (TSI) which provides connection-oriented functions (tConnect, tWrite, tRead, and tDisconnect). If your application interfaces directly with the TSI, many of the overhead details (such as I/O, header, and protocol specifics) must be handled by your application. In this case, you must be familiar not only with this document, but also with the Freeway Client-Server Interface Control Document. If your TSI application will communicate with a remote data link application, you must also be familiar with the appropriate protocol programmer’s guide.

Figure 1–3 shows a typical Freeway connected to a locally attached client by a TCP/IP network across an Ethernet LAN interface. Running a client TSI application in the Freeway client-server environment requires the basic steps described in Section 1.2.1 through Section 1.2.5.
1.2.1 Establishing Freeway Internet Addresses

Freeway must be addressable in order for a client application to communicate with it. In the Figure 1-3 example, the TCP/IP Freeway server name is freeway2, and its unique Internet address is 192.52.107.100. The client machine where the client application resides is client1, and its unique Internet address is 192.52.107.99. Refer to the Freeway User Guide to initially set up your Freeway and download the operating system, server, and protocol software to Freeway.

1.2.2 Defining the TSI Configuration

After establishing the addressing for your client machine and Freeway server, you must define the TSI connections between your client application and Freeway. To accomplish this, you first define the configuration parameters in a TSI ASCII configuration file, and then you run the tsicfg preprocessor program to create a binary configuration file (see Chapter 3). The tInit function uses the binary configuration file to initialize the TSI environment. If your application uses the DLI, refer to the Freeway Data Link Interface Reference Guide for DLI configuration.

Figure 1-3: A Typical Freeway Environment
1.2.3 Establishing a Freeway TSI Connection

After the TSI configuration is properly defined, your client TSI application uses the `tConnect` function to establish a TSI connection with the Freeway MsgMux through the TCP/IP BSD-style socket interface.

1.2.4 Exchanging Data through the Freeway Message Multiplexor

After the TSI connection is established, the client application can then exchange data through the Freeway MsgMux using the `tRead` and `tWrite` functions. If your application needs to exchange data with a remote data link application through a Freeway ICP, you must handle the protocol-specific link configuration and other details pertaining to the ICP. In this case, refer to your particular protocol programmer’s guide.

1.2.5 Closing a Freeway Session

When your application finishes exchanging data, it calls the `tDisconnect` function to disconnect from the Freeway MsgMux.
1.3 TSI Overview and Features

The TSI provides a transport-independent data transfer mechanism with a common interface across varying operating systems. The TSI shields data transfer applications from any dependencies on the underlying transport service (TCP/IP sockets, shared memory, and so on). In addition, TSI applications are easily ported, due to the consistent TSI interface across all supported operating systems.

The TSI consists of the TSI configuration preprocessor program, tsicfg, and a statically linked reference library containing a set of flexible and easy-to-use functions to establish, maintain, and terminate a connection with a TSI peer application.

The tsicfg preprocessor provides flexibility to the TSI application by allowing the runtime characteristics of the TSI to be modified without recompiling the application software. The TSI can be configured to use blocking or non-blocking I/O. Non-blocking I/O allows an application to service multiple TSI connections without blocking on any one connection. The tsicfg preprocessor program is described in Chapter 3.

The TSI operates within a client-server scheme, in which a TSI “server” application listens for incoming connection requests from TSI “client” applications. However, any TSI application can function as both a client and a server; that is, a TSI application can both listen for incoming connection requests and send connection requests to other TSI server applications. Figure 1–4 shows some possible applications within a TSI environment.

As shown in Figure 1–5, a TSI “server” application is used by Freeway’s MsgMux. Thus TSI “client” applications can communicate in a peer relationship with the Freeway TSI “server” application. Remote client applications can use the TSI’s TCP/IP socket interface, while server-resident applications (SRAs) can use the TSI’s shared-memory interface. Each TSI client application shown in Figure 1–5 is in a peer relationship with the Freeway TSI server application (that is, there is a TSI connection between each client TSI and the Freeway MsgMux).
Figure 1-4: TSI Environment
Figure 1-5: TSI in the Freeway Operating Environment
The major features of the TSI are summarized as follows:

- Communicates with the Freeway server's message multiplexor (MsgMux)
- Communicates with other TSI applications
- Provides transport-service-dependent operations
- Permits transport-service-independent applications
- Supports multiple TSI connections to multiple servers
- Supports blocking I/O
- Supports non-blocking I/O with notification by I/O completion handler (IOCH) or polling
- Provides advanced queuing techniques to minimize internal task switches under the VxWorks operating system
- Provides efficient buffer management to avoid excess memory movement
- Provides flexible text-based configuration services
- Provides an off-line configuration preprocessor program (tsicfg) to increase syntax and semantic checking capability and to reduce real-time (on-line) processing of the configuration parameters
- Provides configuration for all significant TSI service parameters
Chapter 2

TSI Concepts

Note
In this document, the term “Freeway” can mean either a Freeway server or an embedded ICP. For the embedded ICP, also refer to the user guide for your ICP and operating system (for example, the ICP2432 User Guide for Windows NT).

The following TSI concepts are described in this chapter:

• configuration at various levels of the Freeway environment

• blocking versus non-blocking I/O

• buffer management

• system resource requirements

2.1 Configuration in the Freeway Environment

There are several types of configuration required for a client TSI application to run in the Freeway environment:

• Freeway server configuration

• transport subsystem interface (TSI) connection configuration

• protocol-specific ICP link configuration (if applicable)
The Freeway server is normally configured only once, during the installation procedures described in the Freeway User Guide. TSI connection configuration is defined by specifying parameters in a TSI ASCII configuration file and then running the tsicfg preprocessor program to create a binary configuration file. Chapter 3 describes TSI configuration.

If your application communicates with one of Protogate’s data link protocols running on the ICP, you must be familiar with link configuration as described in your particular protocol programmer’s guide.

### 2.2 Blocking versus Non-blocking I/O

**Note**

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

Non-blocking I/O applications are useful when doing I/O to multiple channels with a single process where it is not possible to “block” (sleep) on any one channel. Blocking I/O applications are useful when it is reasonable to have the calling process wait for I/O completion. For example, if you wish to design an application requiring the input of a keyboard as well as background processing, non-blocking I/O would be more efficient, because your process can perform other tasks while waiting for keyboard input.

In the Freeway environment, the term blocking I/O indicates that the open, close, connect, disconnect, read and write functions do not return until the I/O is complete. For non-blocking I/O, these functions might return after the I/O has been queued at the client, but before the transfer to Freeway is complete. The client must handle I/O completions at the software interrupt level in the completion handler established by the tInit or tConnect function, or by periodic use of tPoll to query the I/O completion status.
The effects on different TSI functions, resulting from the choice of blocking or non-blocking I/O, are explained in each function description in Chapter 4.

### 2.2.1 I/O Completion Handler for Non-Blocking I/O

When your application uses non-blocking I/O and an I/O condition occurs, the current task is preempted by a high-priority task called an I/O completion handler (IOCH) which is designated to handle the I/O. This high-priority IOCH is written by the application programmer and should adhere to the following real-time criteria to prevent the IOCH from impacting overall system performance:

- minimize the amount of processing performed within the IOCH so other vital system operations are not prevented from executing
- allow the non-preemptive priority routines to complete the processing
- avoid activities such as disk I/O which might block the operations

### 2.3 Buffer Management

This section describes how the Freeway buffer management system operates. For users who do not need a detailed understanding of the system design, Section 2.3.1 gives a system buffer overview and an example for reconfiguring your system buffers. Section 2.3.2 through Section 2.3.6 give the detailed information for those interested.

---

**Note**

Freeway buffer management is implemented in the TSI; however DLI uses the TSI system for its own buffer management. Therefore, the DLI perspective is also presented throughout this section. If your application interfaces to the TSI only (not the DLI), you can disregard the DLI-specific information.

---
2.3.1 Overview of the Freeway System Buffer Relationships

In the Freeway environment, user-configurable buffers exist in the ICP, the client, and the server. These buffers must be coordinated for proper operation between the client application, the Freeway server, and the ICP. The default sizes for each of these buffers are designed for operation in a typical Freeway system. However, if your system requires reconfiguration of buffer sizes, the basic procedure is as follows (Section 2.3.1.1 gives an example calculation):

Step 1: As a general rule, define the ICP buffer size first. ICP buffers must be large enough to contain the largest application data buffer transmitted or received. Most Protogate protocols on a Freeway ICP provide a data link interface (DLI) configuration parameter (such as msgBlkSize for BSC) through which the user can configure the ICP message buffer size. The typical default ICP buffer size for most Protogate protocols is 1024. Refer to your protocol-specific Programmer’s Guide to determine the parameter name and default.

Note

If your application does not interface to the DLI, the protocol-specific ICP buffer size is also software configurable. Refer to your protocol-specific Programmer’s Guide.

Step 2: Define the client buffers in the client’s TSI configuration file. The TSI buffer pool is defined in the configuration file’s “main” section. An optional connection-specific maximum buffer size is allowed in each connection definition. These two configurations are detailed in Section 2.3.2.1 and Section 2.3.2.2, respectively. The buffer size specified in the associated connection definition must be large enough to contain the ICP buffer size.
Step 3: Define the server buffers in the `MuxCfg` server TSI configuration file, which is located in your boot directory. This file is similar to the client TSI configuration file. As with the client, define the TSI buffer pool size in the `MuxCfg` file's "main" section. Then define the optional connection-specific maximum buffer size for each connection. Simply define the connection buffer size for the largest associated client requirement. The buffer pool size must be at least as large as the largest connection buffer size. Section 3.5 on page 65 discusses the `MuxCfg` file in detail, and Figure 3-4 on page 66 shows a `MuxCfg` file.

### 2.3.1.1 Example Calculation to Change ICP, Client, and Server Buffer Sizes

Step 1: Determine the maximum bytes of data your application must be able to transfer. For this example calculation, we are assuming a maximum of 1500 bytes to be transferred using the BSC protocol and interfacing to Protogate's DLI. This is the value that must be assigned to the ICP buffer size (the DLI `msgBlkSize` parameter for BSC).

Step 2: Based on the above 1500-byte `msgBlkSize` parameter, calculate a new `MaxBufSize` for the ICP, client and server. Table 2-1 summarizes the values used in this example.

\[
\text{MaxBufSize} = \text{msgBlkSize} + \text{DLI header size}
\]
\[
\text{MaxBufSize} = 1500 \text{ bytes} + 96 \text{ bytes} = 1596 \text{ bytes}
\]

Step 3: Make the required changes to the protocol-specific portion of the client DLI configuration file as shown in Figure 2-1.
Step 4: Make the required changes to the client TSI configuration file as shown in Figure 2–2.

```
main // TSI “main” section: //
{
    MaxBufSize = 1596; // Must be 1596 (or greater) //
    ...
}
Conn1 // Connection-specific parameters //
{
    MaxBufSize = 1596;
    ...
}
```

**Figure 2–2:** Client TSI Configuration File Changes
Step 5: Make the required changes to the server MuxCfg TSI configuration file (located in your boot directory) as shown in Figure 2-3.

```
main // MuxCfg “main” section: //
{
    MaxBufSize = 1596; // Must be 1596 (or greater) //
    ...
}
MuxConn1 // Connection-specific parameters //
{
    MaxBufSize = 1596;
    ...
}
```

**Figure 2-3: Server MuxCfg TSI Configuration File Changes**

### 2.3.2 Client TSI Buffer Configuration

For users who need to understand the details of the buffer management system, review Section 2.3.2 through Section 2.3.6 carefully. After you define the ICP buffer size as described in Step 1 on page 32, the next step is to define the client TSI buffers.

The TSI provides its own buffer management scheme. Definitions in the client TSI configuration file allow you to create fixed-sized buffers in a TSI-controlled buffer pool (see Section 2.3.2.1). Each connection can then optionally be assigned a unique maximum buffer size (see Section 2.3.2.2). TSI applications can then access these buffers using the tBufAlloc and tBufFree TSI functions.

---

**Note**

For applications using Protogate’s data link interface, the DLI uses the TSI buffer management system for its own buffer management. The dlBufAlloc and dlBufFree DLI functions provide access to buffers in the TSI buffer pool.
Your application is not required to use the TSI buffer management facilities, but Proto-
gate highly recommends it for the following reasons:

- TSI allocates all buffers up front, resulting in better real-time performance than
  allocation through C \texttt{malloc} and \texttt{free} functions
- The number of TSI buffers is configurable for operating environments with lim-
  ited system resources
- TSI allocates the buffer pool on boundaries which minimize memory access over-
  head
- TSI overhead is invisible to the user

2.3.2.1 TSI Buffer Pool Definition

The TSI buffer pool is configured through two parameter definitions in the “main” sec-

tion of the client TSI configuration file (Section 3.3.1 on page 54). The \texttt{MaxBufSize}

parameter specifies the maximum size of each buffer in the TSI buffer pool. The \texttt{MaxBuffers}

parameter specifies the maximum number of buffers available in the TSI

buffer pool and must support the maximum number of I/O requests that could be out-

standing at any one time. After adjusting \texttt{MaxBufSize} as described below, the product of

the \texttt{MaxBufSize} and \texttt{MaxBuffers} parameters defines the TSI buffer pool size.

\texttt{MaxBufSize} defines the maximum size of each buffer. This is the actual data size

the TSI user application has available for its own use. When the buffer pool is defined, TSI cal-

culates an “effective” buffer size which is \texttt{MaxBufSize} plus the additional bytes re
quired

for a TSI header plus any alignment bytes. Alignment bytes are required only if the value

of \texttt{MaxBufSize} plus the TSI header bytes is not divisible by 4.

This “effective” buffer size is invisible to the user application (regardless of whether it

interfaces to the DLI or the TSI); all interactions with the TSI buffer management facili-
ties are based on \texttt{MaxBufSize} and the connection-specific parameter described in

Section 2.3.2.2. If you define \texttt{MaxBufSize} as 1000 bytes, TSI assures that the buffer pool

can provide 1000 bytes for TSI application data.
Figure 2–4 illustrates an example buffer calculation assuming the following sizes:

- MaxBufSize is 1000 bytes
- The TSI header is 18 bytes
- The necessary alignment to make the total divisible by 4 is 2 bytes

TSI adds 18 bytes to the MaxBufSize value to include the TSI header, making the actual size of the buffer allocated by TSI 1018 bytes. Because this actual size is not divisible by 4, TSI increments the value to the next modulo-4 value, in this case, 1020. Regardless of the final size, your TSI application has control of only MaxBufSize bytes.

The TSI application program can obtain the value of MaxBufSize using a tPoll request for the system configuration. Refer to the TSI_POLL_GET_SYS_CFG option (described on page 101 and in Section 4.1.3.1 on page 76), which returns the iMaxBufSize field (described on page 77).

![Figure 2-4: TSI Buffer Size Example](image-url)
Note

The Figure 2–4 example, as viewed from the DLI application’s perspective is shown in Figure 2–5. Of the 1000 bytes specified by the TSI MaxBufSize parameter, 76 bytes are required for the DLI header. After calling dlOpen, the DLI application program can call dlPoll with the DLI_POLL_GET_SESS_STATUS option, which returns the usMaxSessBufSize field. This value is the actual data size available to the DLI application (924 bytes in the Figure 2–5 example).

Figure 2–5: DLI Buffer Size Example

2.3.2.2 Connection-Specific Buffer Definition

After the TSI buffer pool is defined, you have the option of defining a unique maximum buffer size for each connection in the client TSI configuration file. If undefined, the connection buffer size defaults to the MaxBufSize “main” definition for the TSI buffer pool described in the previous Section 2.3.2.1.
Note

The maximum connection buffer size should be at least as large as the defined ICP buffer size, plus any additional client requirements. For example, if you are using the DLI, you must include DLI overhead bytes in the total size of the application data area (see Figure 2–5).

To define a unique buffer size for a connection, use the connection-specific MaxBufSize parameter described in Section 3.3.2 on page 56. This connection buffer size is the buffer size the system allows the user for tWrite requests. No connection buffer size can be larger than MaxBufSize defined for the TSI buffer pool.

The connection buffer size does not change the actual size of the buffer (actual buffers are all MaxBufSize as defined for the TSI buffer pool); it only limits the acceptable size of application write buffers given to TSI through a tWrite request. It enforces a maximum data size that can be sent to the server in any one tWrite request. The tWrite function returns a TSI_WRIT_ERR_INVALID_LENGTH error if the write is attempted with a buffer exceeding the connection's maximum buffer size.

The tRead requests are not limited by the connection buffer size. The size of read requests, when using tRead, is defined by MaxBufSize for the TSI buffer pool (in the “main” definition of the TSI configuration file).

2.3.2.3 TSI Buffer Size Negotiation

A connection’s maximum buffer size can be changed “silently.” When the client’s connection to the Freeway server is accomplished, the client TSI and the server TSI negotiate a maximum buffer size for the established connection. If the sizes are different, the side with the larger connection buffer size changes its size to that of the smaller. After the connection is established, the negotiated maximum buffer size is available using a tPoll request for connection status. Refer to the TSI_POLL_GET_CONN_STATUS option (described on page 101 and in Section 4.1.3.2 on page 78), which returns the
usMaxConnBufSize field (described on page 79). Note that this “final” size is not available until the connection has been successfully established.

---

**Note**

The DLI application program can obtain the actual data size (after the TSI negotiation process during dlOpen) using a dlPoll request with the DLI POLL GET SESS STATUS option, which returns the usMaxSessBufSize field. Refer to the example provided in the Freeway Data Link Interface Reference Guide. A similar approach would apply to using the TSI usMaxConnBufSize field obtained by calling tPoll with the TSI POLL GET CONN STATUS option.

---

### 2.3.3 Server TSI Buffer Configuration

After defining the ICP buffers and the client TSI buffers, the final step is to define the server TSI buffers. The same TSI buffer management design details apply to the server TSI buffers that were described in Section 2.3.2 on page 35 for the client TSI buffers. The only difference is that the server buffer definitions are specified in the MuxCfg server TSI configuration file, which is located in your boot directory. As with the client, define the TSI buffer pool size in the MuxCfg file’s “main” section. Then define the optional connection-specific maximum buffer size for each connection. Simply define the connection buffer size for the largest associated client requirement. The buffer pool size must be at least as large as the largest connection buffer size. Section 3.5 on page 65 discusses the MuxCfg file in detail, and Figure 3–4 on page 66 shows a MuxCfg file. Refer back to Section 2.3.1.1 on page 33 for a sample calculation of ICP, client, and server buffer sizes.

### 2.3.4 Buffer Allocation and Release

The TSI application obtains a buffer from the TSI buffer pool using the tBufAlloc function. The returned buffer address points to the available data area as shown in Figure 2–4 on page 37. The size returned is always the MaxBufSize defined for the buffer.
pool (Table 3-1 on page 54). While the entire data area is available for user data, note the restrictions discussed previously in Section 2.3.2.2 regarding limits placed on tWrite requests by the connection's maximum buffer size definition. The user application releases a buffer back to the TSI buffer pool using the tBufFree function.

Note

DLI applications use the dlBufAlloc and dlBufFree functions to access buffers in the TSI buffer pool.

2.3.5 Cautions for Changing Buffer Sizes

If you need to change the buffer size of your application, keep the following cautions in mind:

- If you increase the ICP buffer size, there may be corresponding changes required in the client and server buffer sizes.

- If you have limited resources and increase the client or server MaxBufSize parameter, consider decreasing the number of buffers allocated in the buffer pool (the MaxBuffers parameter in the client TSI configuration file and the server MuxCfg file).

- Client read buffers too small for an inbound data buffer are returned to the client application with a TSI_READ_ERR_OVERFLOW error indication. Write requests with buffers too large are returned with a TSI_WRIT_ERR_INVALID_LENGTH error indication.

2.3.6 Using Your Own Buffers

If your TSI application needs to use its own buffers, it must know the exact number of overhead bytes used to store the TSI header information. Your application should call tPoll to get the TSI system configuration information (Section 4.8 on page 100) so that it can allocate buffers correctly. Each buffer must be at least iMaxBufSize + usOverhead
bytes in size (these values are described on page 77). Your application must give TSI the address of the memory buffer that is at usOverhead bytes from the beginning of the data area. Figure 2–6 shows a comparison of using the “C” malloc function versus the TSI tBufAlloc function for buffer allocation. Figure 2–7 is a “C” code fragment demonstrating the use of the malloc function.

**Note**
For information about using your own buffers in a DLI application, see the Freeway Data Link Interface Reference Guide.

---

**Figure 2–6**: Comparison of malloc and tBufAlloc Buffers

---

```
... PCHAR pBuf;
TSI_SYS_CFG sysCfg;
int iBufSize, iConnID;
...

tpoll (0, TSI_POLL_GET_SYS_CFG, (PCHAR*)NULL, (PINT)NULL, (PCHAR)&sysCfg);
iBufSize = (int) sysCfg.usOverhead + sysCfg.iMaxBufSize;
pBuf = (PCHAR) malloc (iBufSize);
...

fwrite (iConnID, &pBuf(sysCfg.usOverhead), 100, TSI_WRITE_NORMAL);
...```

---

**Figure 2–7**: Using the malloc Function for Buffer Allocation
2.3.7 Buffer Management (Client versus Server-Resident Applications)

Writing a server-resident application (SRA) using the TSI interface is much like writing a client application. The usual sequence of steps in a client application is:

1. Call `tBufAlloc` (Section 4.2 on page 80) to obtain a buffer.
2. Load the buffer with data.
3. Call `tWrite` (Section 4.13 on page 120), which copies the data for further processing.
4. Call `tBufFree` (Section 4.3 on page 82) to release the buffer allocated in Step 1.
5. Call `tRead` (Section 4.10 on page 109), supplying a NULL pointer for the buffer address so that the TSI allocates the buffer for the client application.
6. Call `tBufFree` (Section 4.3 on page 82) to release the buffer allocated by the TSI in Step 5.

However, there is one significant difference in the way buffers are managed for an SRA which uses the TSI interface (refer back to Figure 1-3 on page 23 to see how an SRA fits in a typical Freeway environment). When the TSI code at the SRA interacts with the TSI code at the MsgMux, the address of the buffer is passed instead of copying the contents of the buffer. At this point, the write is complete for the SRA; however, the buffer is still in use. Eventually the MsgMux writes the buffer to the ICP through the ICP driver. When the driver completes the write, the MsgMux releases the buffer. Therefore, even though the SRA allocated the buffer, it must not release it.

Therefore, Step 4 above is eliminated, and the modified sequence of steps for an SRA is shown below. Refer to the Freeway Server-Resident Application and Server Toolkit Programmer Guide for more information on SRAs.
1. Call \texttt{tBufAlloc} (Section 4.2 on page 80) to obtain a buffer.

2. Load the buffer with data.

3. Call \texttt{tWrite} (Section 4.13 on page 120), which passes the address of the buffer. Eventually the \texttt{MsgMux} writes the buffer to the ICP through the ICP driver. When the driver completes the write, the \texttt{MsgMux} releases the buffer.

4. Call \texttt{tRead} (Section 4.10 on page 109), supplying a NULL pointer for the buffer address so that the DLI allocates the buffer for the SRA.

5. Call \texttt{tBufFree} (Section 4.3 on page 82) to release the buffer allocated by the TSI in Step 5.

### 2.4 System Resource Requirements

When designing your TSI application, you must consider TSI resource requirements. They can be calculated as follows:

Total memory requirements = program size \\
+ (number of buffers \times size of buffer) \\
+ (number of connections \times 300) \\
+ (number of connections \times size of I/O queues \times 44) \\
+ 32,000

Where:

- “number of buffers” is defined by the TSI \texttt{MaxBuffers} parameter (page 54)
- “size of buffer” is defined by the TSI \texttt{MaxBufSize} parameter (page 55)
- “number of connections” is defined by the TSI \texttt{MaxConns} parameter (page 55)
- “size of I/O queues” is defined by the sum of the TSI \texttt{MaxInQ} parameter (page 56) and the TSI \texttt{MaxOutQ} parameter (page 56)
2.4.1 Signal Processing

The TSI disables all signals during processing. The signals are ultimately delivered when they are re-enabled at the end of the TSI call. If this constraint causes a problem for your client application, consider implementing one of the following:

- use non-blocking I/O as described in Section 2.2 on page 30
- use the Timeout TSI configuration parameter (page 57)

Under VMS, ASTs are disabled instead of signals.
Freeway Transport Subsystem Interface Reference Guide
3.1 Configuration Process Overview

The transport subsystem interface (TSI) consists of two major components:

- The `tsicfg` configuration preprocessor program defines the TSI environment prior to run time, using a text configuration file that you create or modify.
- The TSI reference library is used to build your TSI application.

The advantage of using the `tsicfg` preprocessor program is that you do not have to rebuild your application when you redefine the TSI environment.

The TSI configuration process is a part of the installation procedure and the loopback testing procedure described in the Freeway User Guide. However, during your client application development and testing, you might need to perform TSI configuration repeatedly.

The TSI configuration process is summarized as follows:
1. Create or modify a text file specifying the configuration of the transport subsystem interface (TSI) connections.

2. Execute the `tsicfg` preprocessor program with the text file from Step 1 as input. This creates the TSI binary configuration file. If the optional TSI binary configuration filename is supplied, the binary file is given that name plus the `.bin` extension. If the optional filename is not supplied, the binary file is given the same name as your TSI text configuration file plus the `.bin` extension.

   `tsicfg TSI-text-configuration-filename [TSI-binary-configuration-filename]`

**Note**

You must rerun `tsicfg` whenever you modify the text configuration file so that the TSI functions can apply the changes.

When your application calls the `tInit` function, the TSI binary configuration file is used to configure the TSI connections. Figure 3-1 shows the TSI architecture.

**Note**

The Freeway User Guide describes the make files and command files provided to automate the above process and copy the resulting binary configuration files to the appropriate directories. Additionally, each protocol programmer’s guide describes the related protocol specifics of the TSI configuration process.
Figure 3-1: TSI Architecture
3.2 Introduction to TSI Configuration

The tsicfg configuration preprocessor program translates a TSI text configuration file into a binary configuration file. During the translation process, tsicfg processes and verifies each configuration entry in the text configuration file, and the results are stored in the binary configuration file. This process ensures the validity of the configuration parameters before their use by the TSI reference library. The TSI configuration services provide the following features:

- Free-formatted configuration language
- Informative parameter names
- Procedure-like definition entry for each connection definition
- Extensive syntax checking capability
- Extensive semantic checking capability
- Connection-based definition capability

The TSI reference library is a set of function calls used by applications to exchange data between two or more applications over a well-defined transport interface (for example, TCP/IP, shared-memory, APPC/LU 6.2, and so on). The TSI reference library uses the TSI binary configuration file to configure the TSI services as well as connections managed by the TSI. Together with the TSI configuration services, the TSI reference library provides a simple network programming environment for transport-independent applications. It also provides end users the ability to configure individual TSI connections according to their needs.

3.2.1 TSI Configuration Language

The TSI text configuration file contains an optional “main” definition followed by one or more unique connection definitions. The “main” definition specifies general TSI characteristics and must use the name “main.” Each connection definition entry in the TSI text configuration file defines a specific type of TSI connection to be established
between the TSI application and a peer TSI application. Refer to Section 3.6.2 for details of the language grammar. The TSI configuration is described as follows:

```
main
{
    parameter-name = parameter-value;  // comments are ignored  //
}

connection-name
{
    parameter-name = parameter-value;
}
```

Each definition entry must be assigned a name that is unique within the configuration file; tsicfg makes no attempt to ensure the uniqueness of names within the same configuration file. Each `connection-name` uniquely identifies a connection within the same configuration file; it is supplied by the user. Each `parameter-name` is uniquely defined by `tsicfg`; the `parameter-value` is supplied by the user. Comments are considered white spaces and are ignored by `tsicfg`.

### 3.2.2 Rules of the TSI Configuration File

A connection or a parameter name must adhere to the following naming rules:

1. It is similar to variable names in the C language.
2. It can be a string of alphabetic (A through Z, a through z, and _) and numeric (0 through 9) characters.
3. The first character must be alphabetic.
4. The length must not be more than 20 characters.
5. Connection names are case-sensitive while parameter names are not.
6. The TSI does not verify the duplication of connection definition entries at the connection level or at the parameter level. That means if you have defined the same connection entry more than once, the first one is used. If you have defined a parameter within a connection definition entry more than once, the last value is used.
3.2.3 Binary Configuration File Management

The binary configuration file is created in the same directory as the location of the text configuration file (unless a different path is supplied with the optional filename described in Section 3.1 on page 47). On all but VMS systems, if a file already exists in that directory with the same name, the existing file is renamed by appending the .BAK extension. If the renamed file duplicates an existing file in the directory, that existing file is removed by the configuration preprocessor program.

Note

The default binary configuration name contains the period `.` character which plays a special role in the processing of the configuration files. See Section 3.2.4.

3.2.4 On-line Configuration File Processing

TSI can perform the configuration processing on-line. While this feature is available, Protogate recommends adherence to the off-line configuration file process previously described in Section 3.1 on page 47, which is better managed and slightly more efficient.

The off-line process can be performed on-line during TSI initialization (tini) by providing a configuration filename without an embedded `.` character. When such a filename is recognized, TSI attempts to open the file as a text file and calls the TSI configuration preprocessor program (tsicfg). The output file is named “filename”.bin. An error in the configuration file aborts the tini processing with an appropriate error in the TSI log file.

This on-line method requires the configuration text file and the tsicfg preprocessor program to reside in the same directory as the application executable. The resulting .bin file is placed in this same directory.
3: TSI Configuration

Note

Unless on-line configuration is desired, be sure a ‘.’ character appears in the configuration filename provided to `tInit`.

3.3 TSI Connection Definition

There are two groups of TSI configuration parameters: the “main” definition and the connection definitions. Some of the parameters are found in both sets; in this case, the parameter value in the “main” definition applies to the TSI application in general, while the value in a specific connection definition applies only to those TSI connections using that connection definition (unique connection-name). The connection definition parameters allow the application to use connections with different operating characteristics at the same time. For example, a client TSI application using the TCP/IP socket interface can connect to multiple TSI server applications on various remote host machines. For each host machine, the client specifies a connection definition in its TSI configuration file with the `server` parameter set to the IP (Internet Protocol) address of the remote host. In addition, if the configuration needs of the application change (for example, the IP address of a remote host changes), only the configuration file needs to be modified; the application does not need to be modified or re-compiled.

The configuration parameters can be one of three types of values: integer, boolean, or character string. Integer values can be specified as ANSI C decimal, octal, or hexadecimal constants. Boolean and string values are specified as ANSI C character strings (enclosed in double quotes). Boolean values must consist of a string containing the word “yes” or the word “no.” The TSI configuration preprocessor ignores the case of boolean values; that is, “Yes” and “YES” are also valid boolean values. In addition, `tsicfg` ignores the case of parameter names. The parameter definitions in the following sections are given in upper and lower case for readability only. TSI uses a default value for any parameters not explicitly defined.
3.3.1 Parameters for the “main” Definition

The first section in the TSI text configuration file, which is called “main,” specifies the TSI configuration for non-connection-specific operations. If a “main” definition is not specified in the TSI text configuration file, a default “main” entry is used.

The “main” TSI parameters are shown alphabetically in Table 3-1, along with the defaults. You need to include only those parameters whose values differ from the defaults.

### Table 3-1: TSI Parameters for “main” Definition

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsyncIO</td>
<td>“no”(^a)</td>
<td>boolean</td>
<td>Boolean value specifying the use of blocking or non-blocking I/O. A value of “no” specifies blocking I/O.</td>
</tr>
<tr>
<td>DualAddress</td>
<td>n/a</td>
<td>n/a</td>
<td>This parameter is replaced by the “Added Interfaces” configuration described in the Freeway User Guide.</td>
</tr>
<tr>
<td>InterruptTrace</td>
<td>“no”</td>
<td>boolean</td>
<td>A boolean value specifying whether interrupts should be locked out during tracing.</td>
</tr>
<tr>
<td>LogLev</td>
<td>0</td>
<td>0–7</td>
<td>An integer value defining the level of logging the TSI performs and stores in the file name defined by the LogName parameter. A higher level specifies more detailed logging; 0 specifies no logging.</td>
</tr>
<tr>
<td>LogName</td>
<td>“tsilog”</td>
<td>string (6–80)</td>
<td>A string of characters defining the name (path) of the file for storing the TSI logging information. If the path is not included, the current directory is assumed.</td>
</tr>
<tr>
<td>MaxBuffers</td>
<td>1024</td>
<td>4–4096</td>
<td>An integer value specifying the maximum number of buffers to be allocated during run time for the TSI buffer pool. To prevent your application running out of buffers, take care when you specify MaxBuffers to consider the number of TSI connections you need and the queue sizes (MaxInQ and MaxOutQ on page 56). See Section 2.3 on page 31 for details on buffer management.</td>
</tr>
</tbody>
</table>

\(^a\)For Protogate’s Freeway server TSI, AsyncIO must be set to “yes” (refer to Section 3.5 on page 65)
### Table 3-1: TSI Parameters for “main” Definition (Cont’d)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxBufSize</td>
<td>1024 + TSI overhead</td>
<td>1–64000</td>
<td>An integer value specifying the maximum size of each buffer in the TSI buffer pool. This user-supplied value does not include TSI overhead; the TSI overhead value is calculated by TSI and supplied to the user via the <code>usOverhead</code> parameter (page 77). See Section 2.3 on page 31 for details on buffer management.</td>
</tr>
<tr>
<td>MaxConns</td>
<td>1024</td>
<td>1–1024</td>
<td>An integer value defining the maximum number of connections the TSI can manage at one time.</td>
</tr>
<tr>
<td>ServerName</td>
<td>“Freeway”</td>
<td>string</td>
<td>A string of characters specifying the name of a TSI “server” application that runs in the VxWorks environment. This parameter applies only to TSI “server” applications using the shared-memory interface; the purpose is to identify themselves to server-resident applications (SRAs) operating as shared-memory clients. The <code>Server</code> parameter in a shared-memory “client” connection’s configuration definition must match the <code>ServerName</code> parameter in the “server’s” configuration definition. Also see the <code>Server</code> parameter (page 57)</td>
</tr>
<tr>
<td>StackSize</td>
<td>10240</td>
<td>1–64000</td>
<td>An integer value specifying the value of the stack to be used by the TSI for spawning its internal tasks. This parameter applies only to the VxWorks environment.</td>
</tr>
<tr>
<td>TraceLev</td>
<td>0</td>
<td>0–31</td>
<td>An integer value defining the level of tracing (or the sum of several levels) which the TSI performs for this session. This parameter can be overridden by the connection definitions following the “main” section. See also Appendix C.</td>
</tr>
<tr>
<td>TraceName</td>
<td>“tsitrace”</td>
<td>string</td>
<td>A string of characters defining the name (path) of the file for storing the TSI tracing information. If the path is not included, the current directory is assumed.</td>
</tr>
<tr>
<td>TraceSize</td>
<td>0</td>
<td>512–1048576</td>
<td>An integer value specifying the size of the internal trace buffer. The default is zero (tracing is not performed). The smallest allowable size is 512.</td>
</tr>
</tbody>
</table>
### 3.3.2 Parameters for the Connection Definition (Non-transport Specific)

Each additional connection definition specifies unique TSI connections; that is, each definition has a unique connection-name. The TSI parameters are shown alphabetically in Table 3–2, along with the defaults. You need to include only those parameters whose values differ from the defaults.

**Table 3-2: TSI Parameters for Non-Transport Specific Connection**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsyncIO</td>
<td>“no” a</td>
<td>boolean</td>
<td>Boolean value specifying the use of blocking or non-blocking I/O. A value of “no” specifies blocking I/O.</td>
</tr>
<tr>
<td>LogLev</td>
<td>0</td>
<td>0–7</td>
<td>An integer value defining the level of logging the TSI performs for this connection. A higher level specifies more detailed logging, while 0 specifies no logging. This value overrides the LogLev defined in the “main” section.</td>
</tr>
<tr>
<td>MaxBufSize</td>
<td>MaxBufSize defined in “main”</td>
<td>1 to MaxBufSize defined in “main”</td>
<td>An integer value specifying the maximum data size of the TSI buffers for this connection only. The value must be less than or equal to the “main” entry. The default value is the size specified in the “main” section.</td>
</tr>
<tr>
<td>MaxErrors</td>
<td>10</td>
<td>10–100</td>
<td>An integer value specifying the number of I/O errors the TSI can tolerate before declaring the connection is unusable.</td>
</tr>
<tr>
<td>MaxInQ</td>
<td>10</td>
<td>2–1000</td>
<td>An integer value specifying the number of entries in the TSI internal input queue. Make sure MaxBuffers defined in the “main” section (page 54) is adequate for your requirements, especially if your application uses non-blocking I/O. Use caution when changing the queue size parameters. TSI allocates buffers for each connection based on this parameter, and increasing the queue size could cause a buffer allocation problem on the server.</td>
</tr>
<tr>
<td>MaxOutQ</td>
<td>10</td>
<td>2–1000</td>
<td>An integer value specifying the number of entries in the TSI internal output queue. See MaxInQ above.</td>
</tr>
</tbody>
</table>

*a For Protogate's Freeway server TSI, AsyncIO must be set to "yes" (refer to the Freeway User Guide)*
Table 3-2: TSI Parameters for Non-Transport Specific Connection (Cont’d)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>none</td>
<td>string</td>
<td>A string of characters identifying the TSI “server” application with which to connect. This parameter is used differently for each transport interface:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(δ 20)</td>
<td>• For “tcp-socket” connections, this parameter specifies the host machine on which the TSI “server” resides. The string can be either the machine’s host name or its IP address in “dot” notation (for example, “freeway2” or “192.52.107.100”). This parameter is used in conjunction with the WellKnownPort parameter (page 58) to establish “tcp-socket” connections. Note that your application may be blocked while TSI searches for the server’s name if your network has Domain Name Server (DNS) setup.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• For “shared-memory” connections, this parameter identifies the TSI peer “server” application or process. The client connection’s Server parameter must match the server application’s ServerName parameter (page 55) defined in the “main” entry of the server application’s configuration file. The Server parameter is used in conjunction with the ShmPeerName parameter (page 59) to establish “shared-memory” connections.</td>
</tr>
<tr>
<td>Timeout</td>
<td>60</td>
<td>0–63999</td>
<td>An integer value specifying the number of seconds the TSI uses to time activities within this connection.</td>
</tr>
<tr>
<td>TraceLev</td>
<td>0</td>
<td>0–31</td>
<td>An integer value defining the level of tracing (or the sum of several levels) which the DLI performs for this session. If specified, this value overrides the “main” TraceLev parameter. See also Appendix C.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 = no trace 1 = read only 2 = write only 4 = interrupt only 8 = application IOCH 16 = user’s data</td>
</tr>
<tr>
<td>Transport</td>
<td>no default</td>
<td>string</td>
<td>A string of characters specifying the transport interface to be used by this connection. There are no defaults. Supported transport interfaces include “tcp-socket” for TCP/IP sockets and “shared-memory” for VxWorks shared-memory (server-resident applications).</td>
</tr>
<tr>
<td></td>
<td>(must be specified)</td>
<td>(δ 20)</td>
<td></td>
</tr>
</tbody>
</table>

DC 900-1386D  57
3.3.3 Parameters for Connection Definition (TCP/IP Socket Transport)

The TSI configuration parameters required for a TCP/IP socket transport connection are shown alphabetically in Table 3–3, along with the defaults. You need to include only those parameters whose values differ from the defaults. See Section 3.4.1 for an example.

**Table 3-3: TSI Parameters for TCP/IP Socket Transport Connection**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPKeepAlive</td>
<td>“no”</td>
<td>Boolean</td>
<td>A Boolean value specifying whether or not TCP/IP should enable periodic transmission to keep a connected link active while there are no user data transmissions on the link.</td>
</tr>
<tr>
<td>TCPNoDelay</td>
<td>“no”</td>
<td>Boolean</td>
<td>A Boolean value specifying whether or not TCP/IP should send a small packet as soon as possible.</td>
</tr>
<tr>
<td>WellKnownPort</td>
<td>0x2010</td>
<td>5001, 32676</td>
<td>An integer value (usually specified in hexadecimal) specifying the TCP/IP port to be used by this connection. A “server” connection (tListen) will bind to this port, while a “client” connection (tConnect) will attempt to connect to this port on the server's host machine (specified by the client connection's Server parameter, page 57).</td>
</tr>
<tr>
<td>LocalPort</td>
<td>0x0000</td>
<td>1024–65535 (0x0400–0xFFFF)</td>
<td>An integer value (usually specified in hexadecimal) specifying the local IP port used to connect to a DLI/TSI server, such as a Freeway. The default value of 0 allows the operating system to select any unused local IP port. Since the client application uses the specified local IP port for each connection attempt using a given TSI connection definition, this parameter can be used to force a Freeway to terminate a connection which was not terminated normally (for example, if the client machine crashed without properly closing its sockets, or if the client machine closed its sockets while the network was physically disconnected).</td>
</tr>
</tbody>
</table>
3.3.4 Parameters for Connection Definition (Shared-Memory Transport)

The TSI configuration parameters required for a VxWorks shared-memory transport connection (server-resident applications) are shown alphabetically in Table 3-4, along with the defaults. You need to include only those parameters whose values differ from the defaults. See Section 3.4.2 for an example.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Valid Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShmKey</td>
<td>n/a</td>
<td>n/a</td>
<td>Reserved.</td>
</tr>
<tr>
<td>ShmMaxInQ</td>
<td>10</td>
<td>2–1000</td>
<td>An integer value specifying the size of the TSI internal shared-memory input queue. It is recommended that this parameter value match the value of the connection's MaxInQ parameter (page 56).</td>
</tr>
<tr>
<td>ShmMaxOutQ</td>
<td>10</td>
<td>2–1000</td>
<td>An integer value specifying the size of the TSI internal shared-memory output queue. It is recommended that this parameter value match the value of the connection's MaxOutQ parameter (page 56).</td>
</tr>
<tr>
<td>ShmPeerName</td>
<td>no default</td>
<td>string (ð 20)</td>
<td>A string of characters specifying the connection-name of a TSI &quot;server&quot; connection (tListen) to which this &quot;client&quot; connection wishes to connect. This parameter is used in conjunction with the connection's Server parameter (page 57) in establishing a &quot;shared-memory&quot; connection.</td>
</tr>
</tbody>
</table>
3.4 Example TSI Configurations

This section describes the supported transport interfaces. Because TSI was intended to shield the application from any transport interface dependencies, the same application can be run over different transport interfaces; the user needs to modify only the application’s TSI configuration file. The application might need to make modifications due to operating system dependencies (especially under VxWorks); however, the interface to TSI should remain basically constant. Recall that TSI applications “connect” using the tConnect and tListen calls which take a connection-name (associated with a connection definition) as a parameter. The transport-specific parameters within each connection definition control this connection process.

3.4.1 TCP/IP Socket Transport Interface

To configure a TSI connection to use the TCP/IP socket transport interface, the connection definition’s Transport parameter (page 57) must have the value “tcp-socket”. The TCP/IP socket-specific parameters (see Table 3–3 on page 58) include:

- WellKnownPort
- LocalPort
- TCPKeepAlive
- TCPNoDelay

The WellKnownPort parameter along with the Server parameter (page 57) specify a “connection point.” Assume that we have two TSI applications executing on two machines connected by a TCP/IP network. An example would be the two applications labeled Client1 and Server or the two applications labeled Client2 and Server in Figure 1–5 on page 27.

A possible TSI configuration for the two applications is shown in Figure 3–2. “Server” application X (connection-name “TCPserver”) wishes to listen for an incoming connection, while “client” application Y (connection-name “TCPclient”) wishes to connect to “server” application X.
Notice the following points regarding the two TSI configuration files in Figure 3-2:

- The “server” application performing the `tListen` call (application X) needs to specify only the `WellKnownPort` parameter in the TSI configuration file, not the `Server` parameter. This is because the “server” TSI software automatically uses the address of the machine on which it is running.

- The “client” application performing the `tConnect` call (application Y) specifies the `Server` parameter as the host name (or IP address) of the remote machine as well.
as the WellKnownPort parameter to which the remote application is bound. If the “client” application Y needs to connect to a similar “server” application running on a different machine, only the Server and WellKnownPort parameters of the “TCPclient” connection definition must be modified.

Using the Figure 3–2 connection definitions, the “server” application X would call tListen using the connection-name “TCPserver”:

```
ID = tListen ("TCPserver", funcptr);
```

The “client” application Y would call tConnect using the connection-name “TCPclient”:

```
ID = tConnect ("TCPclient", funcptr);
```

### 3.4.2 Shared-Memory Transport Interface (VxWorks Only)

To configure a TSI connection to use the shared-memory transport interface, the connection definition’s Transport parameter (page 57) must have the value “shared-memory”. The shared-memory-specific parameters (see Table 3–4 on page 59) include:

- ShmMaxInQ
- ShmMaxOutQ
- ShmPeerName
- ShmKey

The ShmPeerName parameter along with the ServerName “main” parameter (page 55) specify a “connection point.” Assume that we have two TSI applications (“server” and “client”) executing as in the previous TCP/IP example; however, shared-memory applications must be running on the same machine under VxWorks. An example would be the two applications labeled **Client3** and **Server** in Figure 1–5 on page 27.

A possible TSI configuration for each application is shown in Figure 3–3. “Server” application X (connection-name “SHM server”) wishes to listen for an incoming connection, while “client” application Y (connection-name “SHM client”) wishes to connect to “server” application X.
Notice the following points regarding the two TSI configuration files in Figure 3-3:

- The **Server** parameter is used to match a peer TSI shared-memory application, and its value must match the value of the peer's **ServerName** parameter.
• The ShmPeerName parameter is used to match a particular connection definition (connection-name) in the peer’s configuration file.

Using the Figure 3-3 connection definitions, the “server” application X would call tListen using the connection-name “SHM server”:

    ID = tListen ("SHMserver", funcptr);

The “client” application Y would call tConnect using the connection-name “SHM client”:

    ID = tConnect ("SHMclient", funcptr);
3.5 Protogate's Freeway Server TSI Configuration

During the software installation procedures described in the Freeway User Guide, the default Protogate server TSI configuration file named MuxCfg (see Figure 3–4) was installed in the freeway/boot directory on the boot server. Freeway uses the information in the MuxCfg file to configure the Protogate server-resident TSI software so it can communicate (using the Freeway message multiplexor) with the client TSI software. Refer back to Figure 1–5 on page 27 to see the server TSI software relative to the Freeway message multiplexor.

There is one critical difference in the Protogate server TSI software, namely that it must use TSI non-blocking I/O support (that is, the AsyncIO parameter must be set to “yes”, as shown in Figure 3–4). Changing the AsyncIO parameter to “no” or omitting it, will prevent the Protogate server TSI software from operating as designed.

---

**Caution**

Before modifying the MuxCfg file for the Protogate server TSI software, you should be familiar with the parameter descriptions in Table 3–1 through Table 3–4 (page 54 through page 59). Of particular importance are those parameters that control server resources, such as the TSI buffer pool size (MaxBuffers parameter on page 54) or message size (MaxBufSize parameter on page 55). Improper values could adversely affect server operation.

Also keep the following points in mind if you must modify the MuxCfg file:

- The Transport parameter (page 57) has no default and must be defined.
- Unlike a client TSI configuration file (such as shown in Figure 3–2 on page 61), the Server parameter (page 57) is not required for MuxCfg because the server TSI software automatically uses the address of the machine on which it is running.
- The parameters can appear in any order in the configuration file and can be upper-case, lower-case, or a mixture.
- If a parameter is not explicitly contained in the file, the default is used (defaults are identified in Figure 3–4).
// Default TSI configuration file for the Protogate server-resident software:
//
// Date  Initials  Abstracts
// 12may94  pmt  Original coding...
//
// Note that the parameters commented as default could have been omitted
//
main
{
    AsyncIO = "yes"; // must be specified as yes
    LogLev = 0; // default
    MaxBuffers = 2048;
    MaxConns = 128; // default
    MaxBufSize = 1024; // default
    StackSize = 10240; // default
    TraceName = "/ram1/msgmux.trc";
    TraceSize = 64000;
    TraceLev = 3;
}

server1 // connection-name
{
    AsyncIO = "yes"; // must be specified as yes
    LogLev = 0;
    MaxBufSize = 1024; // default
    MaxErrors = 10; // default
    MaxInQ = 10; // default
    MaxOutQ = 10; // default
    TCPKeepAlive = "no"; // default
    TCPNoDelay = "no"; // default
    Timeout = 63999;
    TraceLev = 3;
    Transport = "tcp-socket"; // must be specified (no default)
    WellKnownPort = 0x'2010'; // default
}

Figure 3-4: TSI Configuration File (MuxCfg) for Protogate Server-Resident TSI
3.6 Miscellaneous TSI Configuration Details

After you are familiar with the fundamentals of working with the tsicfg preprocessor program, the additional details described in this section might be of interest.

3.6.1 TSI Configuration Error Messages

The TSI configuration preprocessor, tsicfg, can display one of the following error or warning messages:

Invalid type specified — STRING expected  Your parameter value does not match the expected type. Action: Review your configuration source code for errors, and try again.

Invalid type specified — BOOLEAN expected  You must use a Boolean value (“YES” or “NO”) for this parameter. Action: Review your configuration source code for errors and try again.

Invalid type specified — DEC/HEX/OCT expected  The expected type is decimal, hexadecimal, or octal data format. Action: Review your configuration source code for errors and try again.

Invalid type specified — FLOAT expected  The expected type is floating point format. Action: Review your configuration source code for errors and try again.

Invalid range specified  The provided parameter value is out of range. Action: Review your configuration source code for errors and try again.

Internal error!  This is an internal error in the tsicfg program. Action: Rerun tsicfg with your source file. If this error consistently occurs, save your configuration source code and contact Protogate for further assistance.

No “main” — Default is used  This is a warning message that your configuration source code does not have the “main” section specified as the first entry in the
configuration source code. Action: None if you do not wish to define the “main” section yourself. Otherwise, consider adding the “main” section as the very first section in the text configuration file.

**Redefined “main” — Definition ignored**  This is a warning message that either you defined the “main” section twice or that you did not code the “main” section as the very first entry in your text configuration file. Action: Review your text configuration file, correct the problem, and rerun tsicfg.

**Invalid transport name** You specified a protocol name that is not supported by the TSI. Action: Review your text configuration file and this manual for the supported transport protocols. Correct the error and try again.

**Undefined parameter name** The provided parameter name is not defined. Action: Review your configuration source code for errors and try again.

**Invalid parameter for specified protocol** This parameter does not belong to this protocol. Action: Review your configuration source code for errors and try again.

**Failed processing file** tsicfg failed to complete processing your configuration file. Action: Review your configuration source code for errors and try again.

**syntax error - cannot backup** This is an internal LEX/YACC error. Action: Retry the operation.

**out of memory** This is an internal LEX / YACC error. Action: Retry the operation.

**yacc stack overflow** This is an internal YACC error. Action: Retry the operation.

**syntax error** A syntax error was encountered in your configuration source code. Action: Locate and correct the error and try the operation again.
3.6.2 Protogate Definition Language (PDL) Grammar

The following extended BNF metalanguage describes the language used to create the TSI text configuration file. The following is a brief description of the symbols used:

1. A string inside of <> is a non-terminal symbol. Its definition is located somewhere down the list.

2. Strings inside of {} separated by a vertical bar (|) make up a list of options. You can select one or none of the options.

3. A string inside of [] is an optional string.

4. Terminal symbols are those not surrounded by <>.

Context Free Grammar

```
<config_entry> ::= <connection_name> <leftbr> <config_stmt_list> <rightbr>
<connection_name> ::= <identifier>
<config_stmt_list> ::= <config_stmt>{<config_stmt_list>}
<config_stmt> ::= [<parameter_name> <equal><parameter_value>;
<parameter_name> ::= <identifier>
<parameter_value> ::= {<string> | 0x<hex> | <decimal> | 0<octal>
0b<binary> | <float> }
<string> ::= <doublequote><str><doublequote>
<str> ::= [<char>{<str> }]
<decimal> ::= <decdigit>[<decimal>]
<octal> ::= <octdigit>[<octal>]
<binary> ::= <bindigit>[<binary>]
<hex> ::= <hexdigit>[<hex>]
<float> ::= <decimal>.<decimal>
<equal> ::= =
<leftbr> ::= {
<rightbr> ::= }
<doublequote> ::= "
```
<char> ::= 1…255
<decdigit> ::= 0…9
<hexdigit> ::= <decdigit>, a–f
<octdigit> ::= 0…7
<bindigit> ::= 0…1
<alpha> ::= a–z, A–Z, _
<digit> ::= <decdigit>
<identifier> ::= <alpha>[<restid>]
<restid> ::= <alphadigit>[<restid>]
<alphadigit> ::= <alpha> | <digit>
In this document, the term “Freeway” can mean either a Freeway server or an embedded ICP. For the embedded ICP, also refer to the user guide for your ICP and operating system (for example, the ICP2432 User Guide for Windows NT).

4.1 Overview of TSI Functions

This chapter describes the transport subsystem interface (TSI) functions used by your application program to interface to Protogate’s Freeway communications server. The TSI is provided as a C library to be linked with your application program. Section 4.1.3 describes TSI data structures that the application programmer uses with the TSI. Table A–1 on page 137 summarizes the error codes related to the TSI functions.

4.1.1 TSI Error Handling

The tserrno variable is globally available to your application and offers similar services to errno provided in the C language. The TSI uses tserrno to store all its error codes. Your application should check this value on all returns from TSI function calls. Applicable error codes are listed with each function call described in this chapter. Appendix A gives a complete list of TSI error codes.

While developing your TSI application, if a particular error occurs consistently, contact Protogate for further assistance.
4.1.2 Categories of TSI Functions

The TSI library can be categorized as shown in Table 4–1.

Table 4–1: TSI Function Groups

<table>
<thead>
<tr>
<th>Category</th>
<th>TSI Functions</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation and termination</td>
<td>tInit, tTerm</td>
<td>Initialize and terminate TSI services</td>
</tr>
<tr>
<td>Connection handling</td>
<td>tConnect, tListen, tDisconnect</td>
<td>Establish and terminate a connection with a peer TSI application</td>
</tr>
<tr>
<td>Data transfer</td>
<td>tRead, tWrite, tPoll, tPost(^1), tSyncSelect</td>
<td>Exchange data with a peer TSI application and obtain specific information related to your connection</td>
</tr>
<tr>
<td>Buffer management</td>
<td>tBufAlloc, tBufFree</td>
<td>Obtain and release fixed-size TSI buffers</td>
</tr>
</tbody>
</table>

\(^1\) Server-resident application only
4: TSI Functions

4.1.2.1 Summary of TSI Functions

The TSI functions used in writing a client application are presented alphabetically in Section 4.2 through Section 4.13. For easy reference after you are familiar with the details of each function call, Table 4-2 summarizes the TSI function syntax and parameters, listed in the most likely calling order.

**Caution**

When using non-blocking I/O, there must always be at least one tRead request queued to avoid loss of data or responses from the ICP.

An overview of using the TSI functions is:

- Start up communications (tInit, tConnect, tBufAlloc)
- Send requests and data using tWrite
- Receive responses using tRead
- For blocking I/O, use tSyncSelect to query read availability status for multiple connections
- For non-blocking I/O, handle I/O completions at the software interrupt level in the completion handler established by the tInit or tConnect function, or by periodic use of tPoll to query the I/O completion status.

Shut down communications (tBufFree, tDisconnect, tTerm)
Table 4-2: TSI Functions: Syntax and Parameters (Listed in Typical Call Order)

<table>
<thead>
<tr>
<th>TSI Function</th>
<th>Parameter(s)</th>
<th>Parameter Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>int tInit</td>
<td>(char *cCfgFile, char *pUsrCB, int (*fUsrIOCH)(char *pUsrCB));</td>
<td>TSI binary configuration file name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional I/O complete control block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional IOCH and parameter</td>
</tr>
<tr>
<td>int tConnect</td>
<td>(char *cTransportName, int (*fUsrIOCH)(char *pUsrCB, int iConnID));</td>
<td>TSI connection definition entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional I/O completion handler</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameters for IOCH</td>
</tr>
<tr>
<td>int tPost</td>
<td>(void);</td>
<td></td>
</tr>
<tr>
<td>int tListen</td>
<td>(char *cTransportName, int (*fUsrIOCH)(char *pUsrCB, int iConnID));</td>
<td>TSI connection definition entry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional I/O completion handler</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameters for IOCH</td>
</tr>
<tr>
<td>int tPoll</td>
<td>(int iConnID, int iPollType, char **ppBuf, int *piBufLen, char *pStat);</td>
<td>Connection ID (tConnect/tListen)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Request type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Poll-type dependent parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size of I/O buffer (bytes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Status or configuration buffer</td>
</tr>
<tr>
<td>char *tBufAlloc</td>
<td>(void);</td>
<td></td>
</tr>
<tr>
<td>int tRead</td>
<td>(int iConnID, char **ppBuf, int iBufLen);</td>
<td>Connection ID (tConnect/tListen)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer to receive data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum bytes to be returned</td>
</tr>
<tr>
<td>int tWrite</td>
<td>(int iConnID, char *pBuf, int iBufLen, int iWritePriority);</td>
<td>Connection ID (tConnect/tListen)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Source buffer for write</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Number of bytes to write</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Normal or expedite write</td>
</tr>
</tbody>
</table>
Table 4-2: TSI Functions: Syntax and Parameters (Listed in Typical Call Order)

<table>
<thead>
<tr>
<th>TSI Function</th>
<th>Parameter(s)</th>
<th>Parameter Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>int tSyncSelect</td>
<td>(int iNbrConnID, int connIDAry[], int readStatArray[])</td>
<td>Number of connection IDs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Packed array of connection IDs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Array containing read status for IDs</td>
</tr>
<tr>
<td>char *tBufFree</td>
<td>(char *pBuf);</td>
<td>Buffer to return to pool</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>int tDisconnect</td>
<td>(int iConnID, int iDiscType)</td>
<td>Connection ID (tConnect/tListen)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mode (normal or force)</td>
</tr>
<tr>
<td>int tTerm</td>
<td>(void);</td>
<td></td>
</tr>
</tbody>
</table>
4.1.3 TSI Data Structures

This section describes the following TSI data structures that your application can access using the tPoll function.

- TSI system configuration structure
- TSI connection status structure
- TSI connection definition list

4.1.3.1 TSI System Configuration

After initializing the TSI services (tInit), your application can obtain system configuration parameters from TSI by calling tPoll with the TSI_POLL_GET_SYS_CFG option (Section 4.8). The information includes the size of buffers in the TSI data pool and the overhead TSI uses for header storage. This information is useful if your application uses its own buffers instead of TSI buffer management's. Your application receives the system configuration information in the data structure shown in Figure 4-1. Table 4-3 describes the fields.

```c
typedef struct _TSI_SYS_CFG {
    int iMaxBufSize;
    unsigned short usMaxConns;
    unsigned short usMaxBufs;
    unsigned short usNumActiveConns;
    unsigned short usNumBufsUsed;
    unsigned short usNumBufsAvail;
    unsigned short usOverhead;
    unsigned short usVersion;
    unsigned short usRevision;
    BOOLEAN tfAsyncIO;
    unsigned char cTraceFileName[TSI_MAX_NAME];
} TSI_SYS_CFG;
```

**Figure 4-1:** TSI System Configuration Data Structure
Table 4-3: TSI System Configuration Data Structure Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iMaxBufSize</td>
<td>The maximum area available for user data in buffers from the TSI buffer pool.</td>
</tr>
<tr>
<td>usMaxConns</td>
<td>The maximum number of connections that can be active simultaneously. This value is configurable using the MaxConns TSI configuration parameter (page 55) in the “main” configuration section.</td>
</tr>
<tr>
<td>usMaxBufs</td>
<td>The maximum number of buffers available for your application. This value is configurable using the MaxBuffers TSI configuration parameter (page 54) in the “main” configuration section.</td>
</tr>
<tr>
<td>usNumActiveConns</td>
<td>The number of connections currently in use. This number should be less than or equal to usMaxConns.</td>
</tr>
<tr>
<td>usNumBufsUsed</td>
<td>The number of buffers currently in use. This number should be less than or equal to usMaxBufs.</td>
</tr>
<tr>
<td>usNumBufsAvail</td>
<td>The number of buffers currently available for use. This number plus usNumBufsUsed should be equal to usMaxBufs.</td>
</tr>
<tr>
<td>usOverhead</td>
<td>The number of additional bytes that must precede your data area in a buffer that your application requests the TSI to read or to write. Your application needs to be aware of this value only if it does not wish to use the TSI buffer management scheme. This value, usOverhead + iMaxBufSize, must be equal to the MaxBufSize TSI configuration parameter (page 55).</td>
</tr>
<tr>
<td>usVersion</td>
<td>The current version of the TSI. You must know this number when you call Protogate's customer support.</td>
</tr>
<tr>
<td>usRevision</td>
<td>The current revision of the TSI.</td>
</tr>
<tr>
<td>tfAsyncIO</td>
<td>A BOOLEAN value indicating whether or not the TSI was configured to use non-blocking I/O (zero specifies blocking I/O).</td>
</tr>
<tr>
<td>cTraceFileName</td>
<td>The name of the file the TSI uses to trace the data that flows through the layer.</td>
</tr>
</tbody>
</table>
4.1.3.2 TSI Connection Status

After establishing a connection, your application can obtain connection-related information by calling tPoll with the TSI_POLL_GET_CONN_STATUS option (Section 4.8). This information includes the negotiated buffer size (usMaxConnBufSize), which is the actual data size available for this connection’s user data. Your application receives the connection status information in the TSI_CONN_STAT data structure shown in Figure 4–2. Table 4–4 describes the fields.

```
typedef struct _TSI_CONN_STAT
{
    short iQReadSize;
    short iQWriteSize;
    short iQNumRead;
    short iQNumWrite;
    short iQNumReadDone;
    short iQNumWriteDone;
    short iConnStatus;
    short iNumErrors;
    short iMaxErrors;
    short usMaxConnBufSize
    short iNumCalls;
    BOOLEAN tfOverflow
} TSI_CONN_STAT;
```

**Figure 4–2: TSI Connection Status Data Structure**

### 4.1.3.3 TSI Connection Definition List

The connection definition list is returned to your application if it invokes tPoll with the TSI_POLL_GET_CFG_LIST option. The configuration list is returned through the ppBuf argument, and the number of the connection definitions is given by the piBufLen argument. Refer to tPoll, Section 4.8, for information on this list of connection definitions.

*Note* Refer to the Freeway Data Link Interface Reference Guide for an example program to request DLI session status. Requesting TSI connection status is similar.
### Table 4-4: TSI Connection Status Data Structure Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iQReadSize</td>
<td>The size of the input queue. This value can be configured using the MaxInQ TSI configuration parameter (page 56) in the connection definition section.</td>
</tr>
<tr>
<td>iQWriteSize</td>
<td>The size of the output queue. This value can be configured using the MaxOutQ TSI configuration parameter (page 56) in the connection definition section.</td>
</tr>
<tr>
<td>iQNumRead</td>
<td>The current number of read requests in the read queue. This value is less than or equal to iQReadSize.</td>
</tr>
<tr>
<td>iQNumWrite</td>
<td>The current number of write requests in the write queue. This value is less than or equal to iQWriteSize.</td>
</tr>
<tr>
<td>iQNumReadDone</td>
<td>The current number of read requests that are complete or timed out in the input queue. This value is less than or equal to iQReadSize.</td>
</tr>
<tr>
<td>iQNumWriteDone</td>
<td>The current number of write requests that are complete or timed out in the output queue. This value is less than or equal to iQWriteSize.</td>
</tr>
<tr>
<td>iConnStatus</td>
<td>The current status of the connection. The valid connection status values are: TSI_STATE_NOT_CONNECTED  The current TSI connection is being ended. TSI_STATE_NOT_IN_USE  The current TSI connection is not available for use because it failed to establish a connection or has finished disconnecting. A cleanup is underway. TSI_STATE_NOT_CONNECTED  The TSI is still trying to establish a connection with Freeway or the peer TSI application. TSI_STATE_CONNECTED  The TSI successfully established a connection with Freeway or the peer TSI application. Your application can now read or write to this connection.</td>
</tr>
<tr>
<td>iNumErrors</td>
<td>The current number of I/O errors that has occurred for this connection since the connection was established. Your application can monitor this value to check the health of an active connection.</td>
</tr>
<tr>
<td>iMaxErrors</td>
<td>The maximum number of errors this connection can tolerate before it rejects I/O requests from your application. This value can be defined using the MaxErrors TSI configuration parameter (page 56) in the connection definition section.</td>
</tr>
<tr>
<td>usMaxConnBufSize</td>
<td>The maximum buffer area available to the user for the transfer of data. The value can vary between connections, and the value is valid only after the connection is established. This size does not include any TSI overhead requirements.</td>
</tr>
<tr>
<td>iNumCalls</td>
<td>Reserved</td>
</tr>
<tr>
<td>tfOverflow</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
4.2 tBufAlloc

The tBufAlloc function allocates a fixed-size buffer that is maintained by TSI services. Your application can use these buffers to exchange data with peer TSI applications or for any other purpose. To avoid a buffer depletion problem, your application must return all unused buffers to TSI using tBufFree (Section 4.3).

Though you are not required to use tBufAlloc, you should consider using it for all TSI I/O operations for the following reasons:

- The TSI buffer services account for buffer overhead requirements.
- The TSI allocates all buffers up front, resulting in better real-time performance than the normal C malloc and free functions
- The number of TSI buffers is configurable for operating environments with limited system resources (MaxBuffers, page 54)

To enhance performance, the TSI implementation uses the memory area just before the data area to store its headers. Due to this implementation, if your application does not wish to use the TSI buffer management, it must allocate sufficient memory for not only its data but also the TSI headers. To obtain the amount of memory required for the TSI overhead, your application can invoke tPoll with the TSI_POLL_GET_SYS_CFG option (usOverhead field on page 77). Your application can also use tPoll with the TSI_POLL_GET_SYS_CFG option to obtain the status of buffer management (see Table 4-3 on page 77). Refer to Section 2.3 on page 31 for more information on buffer management issues.

Synopsis

```c
char *tBufAlloc ( void );
```

Parameters

None
Returns

If the `tBufAlloc` function completes successfully, it returns an address that points to the buffer area to be used by your application. Note that the address returned is the address of the data area of the TSI buffer; this address must be used by any further manipulations on the buffer such as `tRead`, `tWrite`, `tPoll`, and `tBufFree`. Otherwise it returns `NULL`, and `tserrno` contains one of the following error codes (listed alphabetically):

- **TSI_BUFA_ERR_NEVER_INIT**  The TSI was never initialized; that is, `tInit` was never called.
  
  **Action:** Review your application and try again.

- **TSI_BUFA_ERR_NO_BUFS**  The TSI exhausted buffers. Possible problems are:
  (1) your application did not release buffers after use, or (2) your application is configured for non-blocking I/O, and it has too many outstanding connections with a large queue size (for example, `MaxBuffers = 1000`, 10 outstanding connections, and each has 100 entries in the queue).

  **Action:** Severe error; consider increasing the number of buffers in the TSI configuration file (`MaxBuffers` on page 54). Review your application and make sure it releases unused buffers to the TSI.

- **TSI_BUFA_ERR_SEVERE_ERR**  The TSI internal call to `QAdd` failed.

  **Action:** If this error occurs consistently, contact Protogate for assistance.

For additional error codes, refer to Appendix A.
4.3 tBufFree

Your application must use tBufFree to release the TSI buffers that it allocated using tBufAlloc. It must also release any read buffer that TSI allocated in tRead (Section 4.10). The buffer is returned to the TSI internal free buffer pool. It is the responsibility of your application to prevent buffer depletion by releasing the unused TSI buffers.

Synopsis

```c
char *tBufFree ( char *pBuf ); /* Buffer to return to buffer pool */
```

Parameters

- `char *pBuf` This field contains the data address of the TSI buffer that was returned by tBufAlloc (or that was allocated by tRead).

Returns

If the tBufFree function completes successfully, it returns the value of pBuf. Otherwise it returns NULL, and tserrno contains one of the following error codes (listed alphabetically):

- **TSI_BUFF_ERR_INVALID_BUF** The buffer pointer provided to tBufFree is either NULL or -1.
  
  Action: Review your application and try again.

- **TSI_BUFF_ERR_NEVER_INIT** The TSI was never initialized; that is, tInit is never called.
  
  Action: Review your program and try again.

- **TSI_BUFF_ERR_NOT_ALLOCATED** The TSI invoked tBufFree to free a buffer and the buffer is not allocated.
Action: Make sure that your application frees only a TSI buffer and that it frees it only once. Review your application and try again.

**TSI_BUFF_ERR_SEVERE_ERR**  The TSI invoked `tBufFree` to free a buffer and either the buffer pool is empty or the buffer does not belong to the TSI.

Action: Review your application and try again.

For additional error codes, refer to Appendix A.
4.4 tConnect

The tConnect function establishes a connection to a peer TSI application. The connection parameters are provided through the TSI configuration file. The TSI searches its configuration file for a match of the connection name provided as the first parameter of this function call. Once found, the TSI loads the connection parameters into memory and begins to establish a connection with the peer TSI application. The destination is part of the TSI connection parameters.

For non-blocking I/O, tConnect returns as soon as it detects a potential blocking condition. Your application is not blocked while the TSI attempts to complete the connection request. When the connection is made, the TSI calls one or both of the IOCH functions provided through the tInit and tConnect calls. If no IOCH functions are provided, none is called. Either tInit or tConnect can be used to supply the IOCH; however, the tConnect IOCH requires a connection ID, and is called for that particular connection only.

Your application uses the iConnID returned from this call for all other TSI calls (tRead, tWrite, and so on). If this connection is configured for non-blocking I/O, you must ensure that the connection is fully established before you call the tRead, tWrite, or tDisconnect functions.

---

Note

If you need to request connection status to obtain the maximum buffer size (which may change due to negotiation procedures during tConnect), your application should wait until after a successful tConnect before calling tPoll with the TSI_POLL_GET_SESS_STATUS option (Section 4.1.3.2 on page 78). See Section 2.3.2.3 on page 39 for details of the negotiation process.

If your application did not previously call tInit, tConnect makes the tInit call. The default values are used.
Synopsis

int tConnect (char *cTransportName, /* TSI connection definition entry name */
         int (*fUsrIOCH) (char *pUsrCB, int iConnID));
/* Optional IOCH for specific connection */

Parameters

char *cTransportName  A string of characters that specifies the name of the desired connection definition entry in the TSI binary configuration file. The associated configuration entry defines the characteristics of the connection you are about to make.

int (*fUsrIOCH) (char *pUsrCB, int iConnID)  This field should contain the address of the I/O completion handler (IOCH) that the TSI invokes immediately after it services an I/O condition for this connection. The IOCH is invoked by the TSI only if this connection is configured for non-blocking I/O. You must write the IOCH yourself. When an I/O condition occurs for this connection, the TSI invokes this IOCH with the pUsrCB value (that you provided through tInit) and the connection ID returned by this call. You can provide a different fUsrIOCH for each connection ID or use the same fUsrIOCH for all IDs. If your application does not wish the TSI to invoke your IOCH, this parameter should be NULL.

Returns

The tConnect function returns a connection ID or ERROR. The connection ID can have a value from zero to the maximum number of connections (MaxConns parameter, page 55) minus one. If the connection definition for the function parameter cTransportName specifies blocking I/O, a returned connection ID indicates that the connection has completed successfully. If the connection definition specifies non-blocking I/O and a connection ID is returned, the application must examine tserrno and invoke tPoll with the TSI_POLL_GET_CONN_STATUS option to determine when the connection has successfully completed. If the connection has not completed upon return, tserrno is set to TSI_EWOULDBLOCK.
If the tConnect function returns ERROR, the connection failed, and your application must check tserrno which contains one of the following error codes (listed alphabetically):

- **TSI_CONN_ERR_CLOSE_FAILED**  The TSI failed to close its transport-dependent connection when it failed to complete the connection request.
  
  **Action:** If this error occurs consistently, contact Protogate for assistance.

- **TSI_CONN_ERR_CONN_INIT_FAILED**  TSI failed to initialize a connection entry for your application.
  
  **Action:** Check the TSI connection configuration.

- **TSI_CONN_ERR_INVALID_PROT**  The transport parameter in the connection definition of the TSI configuration file is not valid.
  
  **Action:** Check whether your configuration program, tsicfg, is at the same level as your library.

- **TSI_CONN_ERR_INVALID_STATE**  The TSI encountered an invalid state in its state processing machine.
  
  **Action:** If this error occurs consistently, contact Protogate for assistance.

- **TSI_CONN_ERR_OPEN_FAILED**  The TSI failed to establish a transport-dependent connection to the peer TSI application.
  
  **Action:** Terminate your application and review your TSI configuration file.

- **TSI_CONN_ERR_QADD_FAILED**  The TSI failed to insert its internal request in the internal I/O queue.
  
  **Action:** Severe error; terminate your application and try again.
TSI_CONN_ERR_RETRY_EXCEEDED  The TSI retry count exceeded its limit.

Action: Terminate your application and review your configuration parameter for this connection.

TSI_CONN_ERR_SOCK_ALLOC_FAILED  The TSI failed to allocate the resources necessary to support the connection.

Action: Review your connection configuration parameters. You can also review your error log for additional information.

TSI_CONN_ERR_TINIT_FAILED  The TSI failed to initialize its services. This error occurs only if your application does not explicitly call the tInit function.

Action: Check your binary configuration file. If the default binary configuration file (tsicfg.bin) was used by the TSI, verify its existence.

TSI_EWOULDBLOCK  The requested action could not be completed immediately. The TSI would have blocked this operation if your connection was using blocking I/O.

Action: Use tPoll to check whether your request completed. You can program your application to be notified by one of the I/OCH routines that you provided when you invoked the tInit, tConnect, or tListen function. Refer to Appendix B for information on managing TSI applications using non-blocking I/O.

For additional error codes, refer to Appendix A.
4.5 tDisconnect

The tDisconnect function terminates an active connection between your application and the peer TSI application.

If this connection is configured for non-blocking I/O, the application must examine tserrno and invoke tPoll with the TSI_POLL_GET_CONN_STATUS option to determine when the connection has been successfully disconnected.

It is suggested that your application perform I/O cleanup and then issue a Normal tDisconnect request to the TSI. Issuing tTerm while active connections exist should be the last option.

Synopsis

```c
int tDisconnect (  
  int iConnID, /* Connection ID from tConnect|tListen */  
  int iDiscType ); /* Disconnect mode (normal or force) */
```

Parameters

- **int iConnID**  The connection ID is provided by the TSI through the tConnect or tListen function call. This ID uniquely identifies a TSI connection between your application and the peer TSI application.

- **int iDiscType**  This parameter allows your application to request the TSI to terminate an active connection in either of the following close modes (whether the connection is using blocking or non-blocking I/O):
  
  TSI_DISC_FORCE  When your application issues a Force disconnect for an active connection, the TSI empties the I/O queues and proceeds with the disconnect process without considering the status of the I/O queues. Note that when your application issues a tTerm while active connections exist, the TSI itself issues a Force disconnect request before it frees the TSI service structure.
TSI_DISC_NORMAL  The TSI rejects a Normal disconnect request if its internal input and output queues contain outstanding I/O requests. Your application should consider checking the I/O queue prior to calling tDisconnect with this mode. You can obtain the status of I/O queues using the tPoll call.

Returns

If the tDisconnect function completes successfully, it returns OK. Otherwise, it returns ERROR, and tserrno contains one of the following error codes (listed alphabetically):

TSI_DISC_ERR_CLOSE_FAILED  The TSI failed to close the listener.

Action: Severe error; if this error occurs consistently, contact Protogate for assistance.

TSI_DISC_ERR_DISC_FAILED  The disconnect failed and the connection has been returned to the connected state.

Action: If this error occurs consistently, contact Protogate for assistance.

TSI_DISC_ERR_INVALID_ID  The provided connection ID is invalid.

Action: Review your program's logic. If this error occurs consistently, contact Protogate for assistance.

TSI_DISC_ERR_INVALID_STATE  The TSI encountered an invalid state in its state processing machine.

Action: Review the TSI trace and error logs. If this error occurs consistently, contact Protogate for assistance.

TSI_DISC_ERR_INVALID_TYPE  Invalid disconnect type requested (use TSI_DISC_NORMAL or TSI_DISC_FORCE).

Action: Review your program's logic.
TSI_DISC_ERR_NEVER_INIT  The TSI was never initialized. You must invoke tInit before using this function.

Action: Correct your program's logic and try again.

TSI_DISC_ERR_Q_NOT_EMPTY  Your application requested a normal disconnect on a given connection, and the internal I/O queues for that connection are not empty.

Action: Review your program's logic, and try again.

TSI_DISC_ERR_QADD_FAILED  The TSI failed to add the disconnect packet to its internal queue.

Action: If this error occurs consistently, contact Protogate for assistance.

TSI_DISC_ERR_SEVERE_ERROR  The TSI failed to access its internal I/O queues.

Action: Severe error; terminate your application and try again.

TSI_EWOULDBLOCK  The requested action could not be completed immediately. The TSI would have blocked this operation if your connection was using blocking I/O.

Action: Use tPoll to check whether your request completed. You can program your application to be notified by one of the IOCH routines that you provided when you invoked the tInit, tConnect, or tListen function. Refer to Appendix B for information on managing TSI applications using non-blocking I/O.

For additional error codes, refer to Appendix A.
4.6 tInit

The tInit function is usually the first TSI function your application calls. It initializes the TSI services based upon the “main” definition in the TSI binary configuration file (described in Section 3.3.1 on page 54) provided through the first parameter, cCfgFile. This function call is optional; if your application does not explicitly call tInit, the tListen or tConnect function calls it implicitly, in which case the default values are used.

Even though the TSI does not require your application to call tInit prior to its being used, it is suggested that your application always call tInit and tTerm. If tTerm is not called before your application ends, connection and system resources might not be released properly.

Synopsis

```c
#include <tsi.h>

int tInit (char *cCfgFile, /* TSI binary configuration file name */
          char *pUsrCB, /* I/O complete control block */
          int (*fUsrIOCH)(char *pUsrCB)); /* Optional IOCH */
```

Parameters

- **char *cCfgFile**  
The binary configuration file that contains all TSI run-time parameters. This file results from execution of the TSI configuration program, tsicfg, upon a TSI text configuration file. If this parameter is a NULL pointer, the default file, tsicfg.bin, is used. Whether or not you supply the configuration file name, the binary configuration file must exist for the TSI to operate. An optional on-line configuration method is described in Section 3.2.4 on page 52.

- **char *pUsrCB**  
This field should contain the address of the data area that is accessible by your supplied I/O completion handler (IOCH), fUsrIOCH, below. When the TSI is configured for non-blocking I/O, it invokes your supplied IOCH function after it services an I/O condition. The TSI passes this field as the first parameter to your supplied fUsrIOCH function.
int (*fUsrIOCH) (char *pUsrCB)  This field should contain the address of the I/O completion handler (IOCH) that TSI invokes immediately after it processes all active connections. You must write the IOCH yourself. The IOCH is called only if the TSI is configured for non-blocking I/O. If your application does not wish the TSI to invoke your IOCH, this parameter should be NULL. Either \texttt{tInit} or \texttt{tConnect} can be used to supply the IOCH; however, the \texttt{tConnect} IOCH requires a connection ID, and is called for that particular connection only.

**Returns**

If \texttt{tInit} completes successfully, it returns \texttt{OK}. Otherwise it returns \texttt{ERROR}, and \texttt{tserrno} contains one of the following error codes (listed alphabetically):

- **TSI_INIT_ERR_ACT_ADD_REM_FAILED**  The TSI failed to add its internal active connection queue.
  
  Action: Check your system resources. Refer to Section 2.4 on page 44 to calculate system resources required by the TSI.

- **TSI_INIT_ERR_ACT_QINIT_FAILED**  The TSI failed to initialize its internal active connection queue.
  
  Action: Check your system resources. Refer to Section 2.4 on page 44 to calculate system resources required by the TSI.

- **TSI_INIT_ERR_ALREADY_INIT**  Your application already issued \texttt{tInit}.
  
  Action: Review your program logic and try again.

- **TSI_INIT_ERR_BUF_ADD_REM_FAILED**  The TSI failed to set up its buffer pool queue.
  
  Action: Severe error; terminate your application and try again.
TSI_INIT_ERR_BUF_QINIT_FAILED  The TSI failed to initialize its buffer pool queue.

Action: Check your system resources. Refer to Section 2.4 on page 44 to calculate system resources required by the TSI.

TSI_INIT_ERR_CFG_LOAD_FAILED  The TSI failed to load the system configuration parameters from the provided binary configuration file.

Action: Check the binary configuration file used by the TSI. If your application calls tInit directly, make sure the binary configuration file containing the configuration your application provides exists. If your application does not call tInit directly, the TSI calls this function for you; make sure the default configuration file (tsicfg.bin) exists. Verify the binary file name includes a ‘.’ character. If a text file is supplied (as discussed in Section 3.2.4 on page 52), verify the file name and its existence in the current directory (where the application program is executing). Review your program logic and try again.

TSI_INIT_ERR_LOG_INIT_FAILED  The TSI failed to initialize its internal logging and tracing facility.

Action: Check your logging-related and tracing-related parameters in the currently used TSI configuration file.

TSI_INIT_ERR_NO_MEM  The TSI memory initialization failed.

Action: Make sure your operating environment provides sufficient memory resources for your application.

TSI_INIT_ERR_NO_RESOURCE  No memory resource is available for the TSI to start its services.

Action: Make sure your operating environment provides sufficient memory resources for your application.
TSI_INIT_ERR_NO_TRACE_BUFFER  No memory is available for the TSI trace buffer.

Action: Check your system resources. Refer to Section 2.4 on page 44 to calculate system resources required by the TSI.

TSI_INIT_ERR_SETRLIMIT_FAILED  The TSI call to `setrlimit` failed. SUNOS, AIX, and SOLARIS only.

Action: Check your system resources.

TSI_INIT_ERR_TASK_VAR_FAILED  The TSI failed to establish its internal task variables (VxWorks only).

Action: If this error occurs consistently, contact Protogate for assistance.

TSI_INIT_ERR_TSCCB_ALLOC_FAILED  The TSI failed to allocate memory for its internal system control block.

Action: Check your system resources. Refer to Section 2.4 on page 44 to calculate system resources required by the TSI. If this error occurs consistently, contact Protogate for assistance.

For additional error codes, refer to Appendix A.
4.7 tListen

The tListen function waits for a connection request from a peer TSI application. It is similar to the tConnect function, except that the TSI waits for the peer TSI application to make a connection request (via tConnect). As with tConnect, the TSI searches the configuration file for a match of the connection name provided as the first parameter of this function call, then loads the parameters into memory. Once a connection request is received and validated, a connection is then made available to the application. Your application must issue another tListen call if it wishes to receive any additional connection requests.

For non-blocking I/O, tListen returns immediately as soon as it detects a potential blocking condition. Your application is not blocked while the TSI is waiting for a connection request. When the connection is made, the TSI calls one or both of the IOCH functions provided through the tInit and tListen calls. If no IOCH functions are provided, none is called. When using non-blocking I/O, the application can have up to TSI_MAX_LISTEN (defined in tsdefs.h file) outstanding “listens” for each connection definition in the application’s TSI configuration file. For example, at startup the application can call tListen multiple times to have several “listening connections” waiting for incoming connection requests.

Your application uses the iConnID returned from this call for all other TSI calls (tRead, tWrite, and so on). If this connection is configured for non-blocking I/O, you must ensure that the connection is fully established before you call the tRead, tWrite, or tDisconnect functions.

**Synopsis**

```c
int tListen (
    char *cTransportName, /* Transport name in TSI config file */
    int (*fUsrIOCH) (char *pUsrCB, int iConnID)); /* Optional IOCH for specific connection */
```
Parameters

char *cTransportName  A string of characters that specifies the name of the desired transport definition entry in the TSI binary configuration file. The associated configuration entry defines the characteristics of the connection to be completed when a connection request arrives.

int (*fUsrIOCH) (char *pUsrCB, int iConnID)  This field should contain the address of the I/O completion handler (IOCH) that TSI invokes immediately after it services an I/O condition for this connection. The TSI invokes the IOCH only if this connection is configured for non-blocking I/O. You must write the IOCH yourself. When an I/O condition occurs for this connection, the TSI invokes passes the pUsrCB value (that you provided with tInit) and the connection ID returned by this call. You can provide a different fUsrIOCH for each connection or use the same fUsrIOCH for all IDs. If your application does not wish the TSI to invoke your IOCH, this parameter should be NULL. The tInit function (Section 4.6) can also be used to define a general-purpose IOCH that is not restricted to one particular connection.

Returns

The tListen function returns a non-negative connection ID or ERROR. The connection ID can have a value from zero to the maximum number of connections (MaxConns parameter, page 55) minus one. If the connection definition for the function parameter cTransportName specifies blocking I/O, a returned connection ID indicates that the connection has completed successfully. If the connection definition specifies non-blocking I/O and a connection ID is returned, the application must examine terrno and invoke tPoll with the TSI_POLL_GET_CONN_STATUS option to determine when the connection has successfully completed. If the connection has not completed upon return, terrno is set to TSI_EWOULDBLOCK.
If this function returns ERROR, the connection failed and tserrno contains one of the following error codes (listed alphabetically):

**TSI_EWOULDBLOCK**  The requested action could not be completed immediately. The TSI would have blocked this operation if your connection was using blocking I/O.

Action: Use tPoll to check whether your request completed. You can program your application to be notified by one of the IOCH routines that you provided when you invoked the tInit, tConnect, or tListen function. Refer to Appendix B for information on managing TSI applications using non-blocking I/O.

**TSI_LSTN_ERR_CLOSE_FAILED**  The TSI failed to close its transport-dependent connection when it failed to complete the connection request.

Action: If this error occurs consistently, contact Protogate for assistance.

**TSI_LSTN_ERR_CONN_INIT_FAILED**  The TSI failed to initialize a connection for the listener.

Action: Check whether your connection name (first argument of tListen) is properly defined in the TSI configuration. Check your error log for additional error codes.

**TSI_LSTN_ERR_INVALID_PROT**  The value of the transport parameter in your configuration file is invalid.

Action: Check whether configuration program, tsicfg, is at the same level as your library.

**TSI_LSTN_ERR_INVALID_STATE**  The TSI encountered an invalid state in its state processing machine.

Action: If this error occurs consistently, contact Protogate for assistance.
TSI_LSTN_ERR_LSTN_FAILED  The TSI failed to establish a transport-dependent connection to the peer TSI application.

Action: Review your TSI configuration file.

TSI_LSTN_ERR_NO_LISTENERS  The TSI failed to remove a connection from the internal listening queue.

Action: If this error occurs consistently, contact Protogate for assistance.

TSI_LSTN_ERR_NOT_SERVER  A non-listener TSI connection tried to execute a transport-dependent listen.

Action: If this error occurs consistently, contact Protogate for assistance.

TSI_LSTN_ERR_QADD_FAILED  The TSI failed to add a connection entry into its internal queue.

Action: Severe error; if this error occurs consistently, contact Protogate for assistance.

TSI_LSTN_ERR_QFULL  The internal listening queue for this connection definition (cTransportName) is full and is not able to accept more listening requests.

Action: Your application can post another listening request later, when one of the queued listeners completes a connection.

TSI_LSTN_ERR_QINIT_FAILED  The TSI failed to allocate an internal listening queue for the listener.

Action: Severe error; terminate your application and try again.

TSI_LSTN_ERR_SETOPT_FAILED  This error is specific to the TCP/IP socket interface. The TSI failed in setting socket options on a newly accepted socket.
Action: If this error occurs consistently, contact Protogate for assistance.

**TSI_LSTN_ERR_SOCK_ALLOC_FAILED**  The TSI failed to allocate a TCP/IP socket for the listener.

Action: Review the TSI log for additional information.

**TSI_LSTN_ERR_TINIT_FAILED**  The TSI failed to initialize its services. This error can occur only if your application did not explicitly call the `tInit` function.

Action: Check your binary configuration file. If the default binary configuration file (`tsicfg.bin`) was used by the TSI, verify its existence.

For additional error codes, refer to Appendix A.
4.8 tPoll

The tPoll function queries information related to I/O operations, connection status, or system configuration status. The poll type parameter specifies the type of query; the most common uses are to acquire the status of a TSI connection and to poll for I/O completions (only for those connections using non-blocking I/O). Your application can call tPoll as often as necessary.

Synopsis

```c
int tPoll (  
    int iConnID,  /* Connection ID from tConnect/tListen */  
    int iPollType,  /* Request type */  
    char **ppBuf,  /* Poll-type dependent parameter */  
    int *piBufLen,  /* Size of I/O buffer in bytes */  
    char *pStat );  /* Status or configuration buffer */
```

Parameters

- **iConnID**  
The connection ID (returned from a tConnect or tListen function) that uniquely identifies an active connection serviced by the TSI. If a connection ID is not relevant to the iPollType parameter specified (TSI_POLL_GET_SYS_CFG, TSI_POLL_GET_CFG_LIST, TSI_POLL_TRACE_OFF, TSI_POLL_TRACE_ON, TSI_POLL_TRACE_WRITE), this parameter should be set to zero.

- **iPollType**  
This parameter specifies the type of poll request to the TSI. Valid poll types are:

  TSI_POLL_GET_CFG_LIST  
  Request the TSI to get a list of all TSI connection definition names defined in the application's TSI configuration file. The list is returned through the ppBuf parameter in NULL-terminated string lists (see Section 4.1.3.3 on page 78). The number of connection definition names in the list is returned in the piBufLen parameter. The list does not contain the definition of the “main” section.
**TSI_POLL_GET_CONN_STATUS**  Request the TSI to get the current connection status of the connection ID, iConnID, given. The connection status is returned through the TSI_CONN_STAT structure (see Section 4.1.3.2 on page 78). The pointer to the TSI_CONN_STAT structure must be provided through the pStat parameter.

**TSI_POLL_GET_SYS_CFG**  Request the TSI to get the TSI system configuration. The system configuration is returned through the TSI_SYS_CFG structure (see Section 4.1.3.1 on page 76). The pointer to the TSI_SYS_CFG structure must be provided through the pStat parameter.

**TSI_POLL_READ_CANCEL**  Request the TSI to remove a read request from this connection's (iConnID parameter) input queue regardless of the completion status of the read request. If the content of the ppBuf (*ppBuf) parameter is NULL, the TSI removes the first entry in the input queue regardless of its completion status. If the content of ppBuf is not NULL, the TSI searches through its input queue for a matching address pointer. If it finds a match, it removes that request regardless of its completion status.

**TSI_POLL_READ_COMPLETE**  Request the TSI to remove the first read request from this connection's input queue if it is either complete, timed-out, or a read error occurred. The address and length of the buffer are returned through the ppBuf and piBufLen parameters. This request removes the first entry in the input queue only if the request is marked “read complete,” “read timed-out,” or “read error.” In all cases, the application is responsible for freeing the returned buffer.

**TSI_POLL_TRACE_OFF**  Request the TSI to disable tracing services. See Appendix C.

**TSI_POLL_TRACE_ON**  Request the TSI to enable tracing services. See Appendix C.
TSI_POLL_TRACE_STORE Use this poll type to write your own information into the TSI trace buffer. Use the pStat parameter to indicate the area of memory to be copied to the trace buffer and the piBufLen parameter to indicate the length of the area to be copied. The length of your trace area must be less than or equal to the size of the trace buffer (TraceSize on page 55). Otherwise, your trace area will be truncated when copied into the TSI trace buffer.

TSI_POLL_TRACE_WRITE Request the TSI to write the contents of the trace buffer to the trace file.

TSI_POLL_WRITE_CANCEL Request the TSI to remove a write request from this connection’s (iConnID parameter) output queue regardless of the completion status of the write request. If the content of the ppBuf (*ppBuf) parameter is NULL, the TSI removes the first entry in the output queue regardless of its completion status. If the content of ppBuf is not NULL, the TSI searches through the output queue for a matching address pointer. If it finds a match, it removes that request regardless of its completion status.

TSI_POLL_WRITE_COMPLETE Request the TSI to remove the first write request from this connection’s output queue if it is either complete or timed-out. The address and length of the buffer are returned through the ppBuf and piBufLen parameters. This request removes the first entry in the output queue only if the request is marked “write complete,” “write timed-out,” or “write error.” In any case, the application is responsible for freeing the returned buffer.

char **ppBuf This parameter specifies an address of a pointer to a buffer area. This parameter must not be NULL when the poll type is TSI_POLL_READ_COMPLETE, TSI_POLL_READ_CANCEL, TSI_POLL_WRITE_COMPLETE, or TSI_POLL_WRITE_CANCEL.
int *piBufLen

This field is used to return the length of the buffer pointed to by the content of the ppBuf parameter (for I/O completes and cancels) or the number of entries in the configuration list for the TSI_POLL_GET_CFG_LIST option.

char *pStat

This field can be a pointer to a connection status or a system configuration structure. If the request is for the connection status, this field is the address of the user-supplied TSI_CONN_STAT structure (Section 4.1.3.2 on page 78). Otherwise, it points to the user-supplied TSI_SYS_CFG structure (Section 4.1.3.1 on page 76).

Returns

If the tPoll function completes successfully, it returns OK. Otherwise it returns ERROR, and tserrno contains one of the following error codes (listed alphabetically):

- TSI_POLL_ERR_BAD_PTR The ppBuf parameter supplied to tPoll was a NULL pointer. This error pertains only to the TSI_POLL_READ_COMPLETE, TSI_POLL_WRITE_COMPLETE, TSI_POLL_READ_CANCEL, and TSI_POLL_WRITE_CANCEL options.

  Action: Modify your application to supply the address of a pointer.

- TSI_POLL_ERR_BUF_NOT_FOUND The TSI could not find the buffer pointed to by *ppBuf in its I/O queues. This error pertains only to the TSI_POLL_READ_CANCEL and TSI_POLL_WRITE_CANCEL options.

  Action: Severe error; if this error occurs consistently, contact Protogate for assistance.

- TSI_POLL_ERR_GETLIST_FAILED The TSI failed to get a list of connection definition entries from the TSI configuration file.

  Action: Verify the configuration file.

- TSI_POLL_ERR_INTERNAL An internal TSI error occurred.
Action: Note the events leading to the error, obtain the TSI log, and contact Protogate for assistance.

**TSI_POLL_ERR_INVALID_ID**  Your connection is no longer valid.

Action: Review your error log, terminate your application, and try again.

**TSI_POLL_ERR_INVALID_IOQ**  The TSI could not access the connection’s I/O queues. This error pertains only to the TSI_POLL_READ_COMPLETE, TSI_POLL_WRITE_COMPLETE, TSI_POLL_READ_CANCEL, and TSI_POLL_WRITE_CANCEL options.

Action: Severe error; if this error occurs consistently, contact Protogate for assistance.

**TSI_POLL_ERR_INVALID_REQ_TYPE**  The iPollType parameter did not contain a valid value.

Action: Review your application to ensure that a valid poll option is used.

**TSI_POLL_ERR_NEVER_INIT**  The TSI was never initialized.

Action: Revise your program and try again.

**TSI_POLL_ERR_OVERFLOW**  The buffer returned in the ppBuf parameter was too small to handle the incoming TSI packet. The overflow portion of the incoming packet was discarded. Note that this buffer must still be freed by the application. This error pertains only to the TSI_POLL_READ_COMPLETE option.

Action: Increase the buffer size supplied to tRead up to the maximum defined for the connection.

**TSI_POLL_ERR_QEMPTY**  The pertinent I/O queue is empty. This error pertains only to the TSI_POLL_READ_COMPLETE,
TSI POLL_WRITE_COMPLETE, TSI POLL_READ_CANCEL, and TSI POLL_WRITECANCEL options.

Action: This can be a normal condition if the application is attempting to empty the corresponding queue.

TSI POLL_ERR_QREM_FAILED The TSI failed to remove a buffer from a non-empty I/O queue. This error pertains only to the non-specific (no buffer pointer was supplied) TSI POLL_READ_CANCEL and TSI POLL_WRITE_CANCEL options.

Action: Severe error; if this error occurs consistently, contact Protogate for assistance.

TSI POLL_ERR_READ_NOT_COMPLETE The head entry in the TSI read queue is still pending (but has not timed out or incurred an error in processing). This error code pertains only to the TSI POLL_READ_COMPLETE option.

Action: This is a normal condition when polling for I/O completions.

TSI POLL_ERR_READ_TIMEOUT The head entry in the TSI read queue has timed out. Note that the buffer returned in the ppBuf parameter must still be freed by the application. This error code pertains only to the TSI POLL_READ_COMPLETE option.

Action: Consider raising the timeout parameter value in the connection definition of the TSI configuration file.

TSI POLL_ERR_SOCK_CLOSED A fatal error occurred on this buffer's attempted I/O operation. The connection between the client application and Freeway has been closed.

Action: Cancel all outstanding read and write requests, and free the buffers. Close the connection. After a successful close, another connection can be attempted. Examine the error logs for details of the failure.
TSI_POLL_ERR_UNBIND  The connection between the client application and Freeway has been closed. The system has performed “Unbind” processing. The connection was closed either because of a “Force Unbind” received from the peer entity (Freeway or client), or because of a failure with the I/O connection.

Action: Cancel all outstanding read/write requests and free the buffers. Close the connection. After a successful close, another connection can be attempted. Examine the message logs on the peer system (if the error occurred in the client application, examine the Freeway log).

TSI_POLL_ERR_WRITE_NOT_COMPLETE  The head entry in the TSI write queue is still pending (but has not timed out or incurred an error in processing). This error code pertains only to the TSI_POLL_WRITE_COMPLETE option.

Action: This is a normal condition when polling for I/O completions.

TSI_POLL_ERR_WRITE_TIMEOUT  The head entry in the TSI write queue has timed out. Note that the buffer returned in the ppBuf parameter must still be freed by the application. This error code pertains only to the TSI_POLL_WRITE_COMPLETE option.

Action: Consider raising the timeout parameter value in the connection definition of the TSI configuration file.

For additional error codes, refer to Appendix A.
4.9 tPost

The tPost function operates only in the VxWorks environment where the basic non-blocking I/O system services are not provided. It signals the TSI to begin processing I/O requests queued by your application.

Currently, tPost implements a controlled task switch environment for VxWorks through the use of binary semaphore mechanisms. Your application must call this function as the last operation before it relinquishes task control to the operating system (for example, sleeping or taking semaphores). Your application must take special consideration to operate in a VxWorks environment. Refer to Appendix B for designing and implementing server-resident applications under a VxWorks environment.

Synopsis

```c
int tPost ( void );
```

Parameters

None

Returns

If the tPost function completes successfully, it returns OK. Otherwise it returns ERROR, and tserrno contains one of the following error codes (listed alphabetically):

- **TSI_POST_ERR_CLIENT_INVALID_SEM**  The TSI call to the VxWorks semGive function failed because the semaphore ID is not valid.
  
  Action: Internal error. If this error occurs consistently, contact Protogate for assistance.

- **TSI_POST_ERR_NEVER_INIT**  The TSI was never initialized (tInit).
  
  Action: Correct your program and try again.
TSI_POST_ERR_SERVER_INVALID_SEM  The TSI call to the VxWorks semGive function failed because the semaphore ID is not valid.

Action: Internal error. If this error occurs consistently, contact Protogate for assistance.

For additional error codes, refer to Appendix A.
4.10 tRead

The tRead function requests the TSI to receive data from the peer TSI application over the specified connection ID.

The use of tRead varies slightly when using non-blocking I/O as opposed to blocking I/O. If the read is for a connection using non-blocking I/O, note the following:

- If tRead returns ERROR and t_errno is set to TSI_READ_ERR_QFULL, the input queue for this connection is full and cannot accept any more requests at this time.

- If tRead returns ERROR and t_errno is set to TSI_EWOULDBLOCK, the read could not be completed immediately, but was queued to the input queue. Upon completion, the read can be retrieved from the queue by calling tPoll with the TSI_POLL_READ_COMPLETE option.

Synopsis

```c
int tRead ( 
    int iConnID, /* Connection ID from tConnect/tListen */
    char **ppBuf, /* Buffer to receive data */
    int iBufLen ); /* Maximum bytes to be returned */
```

Parameters

- int iConnID  The connection ID uniquely identifies an active connection serviced by the TSI. This ID is returned from the tConnect or tListen function call.

- char **ppBuf  This field contains the address of a pointer to a read buffer. The buffer can be allocated using the tBufAlloc function or a similar C function. However, if the buffer is allocated by a function other than tBufAlloc, your application must provide sufficient header space for the TSI control information (usOverhead field on page 77). If its content is NULL, TSI allocates a buffer for your application.

This parameter must not be NULL, but it can be the address of a NULL pointer (*ppBuf = NULL), which instructs the TSI to allocate a TSI read buffer for the
application. Upon return, the TSI fills in this pointer with the address of the allocated TSI buffer. If you let TSI allocate the read buffer, your application is still responsible for releasing that buffer when it no longer needs it, using tBufFree.

int iBufLen  This field specifies the maximum number of bytes to be read. This value must be greater than zero and less than the maximum buffer size configured (iMaxBufSize, page 77, as reported by the tPoll call with the TSI_POLL_GET_SYS_CFG option). The TSI is a message-based transport service, meaning that data is not transferred as a continuous stream of bytes; the data is broken up into discrete packets (messages) that also contain TSI control information. When an application requests a read, it is returned in the next incoming packet. Because the application does not know the data size of the incoming packet, it is suggested that the application always request to read the maximum buffer size allowed and supply an appropriate sized buffer (or let TSI allocate the buffer). Incoming packets are handled as follows:

- If the data length of the next incoming packet is less than or equal to iBufLen, tRead returns the length of the incoming packet.

- If the data length of the next incoming packet is greater than iBufLen, tRead returns ERROR and tserrno is set to TSI_READ_ERR_OVERFLOW. The buffer is truncated to the number of bytes requested, and the remaining data is discarded.

Returns

The tRead function returns the number of bytes transferred if the TSI successfully completes the I/O request. For both blocking and non-blocking I/O connections, a positive return value signifies a successful completion of the read. Otherwise, the return code is ERROR, and tserrno contains one of the following error codes (listed alphabetically):
TSI_EWOULDBLOCK  The requested action could not be completed immediately. The TSI would have blocked this operation if your connection was using blocking I/O.

   Action: Use tPoll to check whether your request completed. You can program your application to be notified by one of the IOCH routines that you provided when you invoked the tInit, tConnect, or tListen function. Refer to Appendix B for information on managing TSI applications using non-blocking I/O.

TSI_READ_ERR_INTERNAL  A TSI internal error was detected during read processing. The TSI log will indicate the specific error.

   Action: Obtain the TSI error log, and contact Protagate for assistance.

TSI_READ_ERR_INVALID_BUF  This function was invoked with a NULL ppBuf pointer.

   Action: Correct your application and try again.

TSI_READ_ERR_INVALID_ID  Your connection ID is no longer valid.

   Action: Review your application and try again.

TSI_READ_ERR_INVALID_LENGTH  Your requested read length (iBufLen) must be greater than zero and less than or equal to the maximum buffer length allowed by the TSI.

   Action: Use tPoll to obtain the maximum buffer size allowed by the TSI (iMaxBufSize). Review your program and try again.

TSI_READ_ERR_INVALID_STATE  This connection is not in a proper state to accept a read request.

   Action: Review your program and try again.
TSI_READ_ERR_LIMIT_EXCEEDED  This connection has a large number of I/O errors that exceeded the maximum number of errors allowed.

Action: Consider increasing the maximum I/O errors parameter in the connection definition entry. Review your operating environments.

TSI_READ_ERR_NEVER_INIT  The TSI was never initialized. Your application must initialize the TSI (tInit) before calls to other TSI functions (except tConnect and tListen) are made.

Action: Review your application.

TSI_READ_ERR_NO_BUFS  Your application requested that the TSI allocate a buffer, but the allocation failed.

Action: Review your program and make sure you free unused buffers. Also, check to see if you configured enough buffers for your application.

TSI_READ_ERR_OVERFLOW  The TSI encountered an overflow of data in your read request. The read length specified was smaller than the size of the incoming TSI packet. See the discussion under the Description heading.

Action: Increase the buffer size supplied to tRead up to the maximum defined for the connection.

TSI_READ_ERR_QADD_FAILED  The TSI failed to add your request to its internal I/O queues for this connection.

Action: Severe error; terminate your application and try again.

TSI_READ_ERR_QFULL  The TSI cannot accept more read requests because its input queue is full.

Action: Your application must remove complete or timed-out read requests, using the tPoll call, before it can request more reads. Review your program and handle this error accordingly.
TSI_READ_ERR_READ_TIMEOUT Your request was not completed in a timely manner.

Action: Consider increasing the timeout value in the configuration parameters for this connection.

TSI_READ_ERR_SELECT The TSI received an error from a system select call during a read attempt using blocking I/O.

Action: Retry the operation. If the error persists, review your application.

TSI_READ_ERR_SOCK_CLOSED A system read attempt failed, and TSI has closed the connection.

Action: Cancel all outstanding read and write requests, and free the buffers. Close the connection. After a successful close, another connection can be attempted. Examine the log files for abnormal conditions prior to the read failure.

TSI_READ_ERR_UNBIND This connection between Freeway and the client application has been closed. The system has performed "Unbind" processing. The connection was closed either because of a "Force Unbind" received from the peer entity (Freeway or client), or because of a failure with the I/O connection.

Action: Cancel all outstanding read/write requests and free the buffers. Close the connection. After a successful close, another connection can be attempted. Examine the log files on the peer system (if the error occurred in the client application, examine the Freeway log).

For additional error codes, refer to Appendix A.
4.11 tSyncSelect

The tSyncSelect function queries a set of connection IDs for a read data available condition. This feature is available only for clients in a Freeway server environment (it is not supported in an embedded ICP environment) using blocking I/O. The client application can query a connection(s) for read data, and if available, perform the read operation without blocking. This operation does not block; it interrogates the system for read data available and immediately returns this status to the user.

The user builds a connection ID array (connIDArray) containing the list of connections for which read availability is requested. The number of connections can be from 1 to the defined maximum number of connections (see the TSI MaxConns parameter on page 55). Connection IDs must begin at position 0 in the array (first position in the array), and be packed (no non-used positions). The contents of this array are not modified by the interface. The number of connection IDs packed in this array is passed in iNbrConID. In addition, a result array is passed which will contain the returned read availability status of the connections in the corresponding array position of the connection ID array. A connection’s availability status is either TRUE (data available) or FALSE (data not available).

Synopsis

```c
int tSyncSelect (  
  int iNbrConnID,  /* # of connection ids in connIDArray */  
  int connIDArray[], /* packed array of connection ids for requested read data status */  
  int readStatArray[]); /* array containing read data status for connections in connIDArray */
```

Parameters

- **int iNbrConnID**  The number of connection IDs to be queried in the following connection ID array. If a value of 0 is passed (no connection IDs to be queried), the function returns zero (0).
int connIDArray[] An array containing the connection IDs whose read availability status is requested. The connection IDs are those returned from tConnect. Connection ids must begin at position 0 (the first array element), and be packed (no non-used positions). These values are not modified by the call.

int readStatArray[] An array passed to the interface for the returned TRUE/FALSE read availability status for connections in the corresponding positions of the connection ID array (connIDArray). This array is modified by the interface. If an error occurs in the call, the contents of this array are indeterminable; all elements should be ignored.

Returns

If the tSyncSelect function completes successfully, it returns the number of connections in the connection ID array that have read data available (if 3 of 7 connections in the array have read data available, a value of 3 is returned). If no connections have data available, 0 is returned.

Successful completion also returns the readStatArray with a TRUE/FALSE value in each position corresponding to the connection ID in the connection ID array. TRUE means that connection has data available; FALSE means data is not currently available. If the function returns a 0 or ERROR, values in this array are indeterminable; they should be ignored. If this function is successful, it modifies iNbrConnID positions in this array.

For any error condition, the tSyncSelect return code is ERROR, and errno contains one of the following error codes (listed alphabetically):

TSI_SYNCSELECT_ERR_INVALID_STATE A connection(s) in the connection ID array (connIDArray) is not in the proper state to accept this request. con-
Connections must be “opened” (in the “ready” state) before this operation can be performed.

Action: Ensure all connections in the connection ID array have successfully opened.

TSI_SYNCSELECT_ERR_SELECT_ERROR  An error was returned from the system select function.

Action: Review the TSI log file for the specific error, and take corrective action.

Example

One connection is open, tConnect returned with a connection ID of 4.

```c
connIDArray[0] = 4;
if ( (nbrReads = tSyncSelect( 1, connIDArray, readStatArray ) ) == ERROR )
{
    error processing
}
if ( nbrReads ) /∗ with only one read in array, we need not look further */
{
    if ( readStatArray[0] == TRUE )
    {
        /∗ process read available for connection connIDArray[0] – tRead */
    }
}
```

With multiple connections in array, go through readStatArray iNbrConnID times or until nbrReads of TRUE are found.
4.12 tTerm

The tTerm function closes all connections and frees all TSI-related system resources. Under normal conditions your application should call tTerm to close all active connections before it exits to the operating system. You should also make an effort to call tTerm when your application ends abnormally.

The tTerm function can be invoked at any time during the life of your application. To use the TSI again, you must call tInit to re-establish the TSI operating environment. It is not recommended that you call tTerm too often in your application because of the timing cost associated with it. However, in some applications this capability might be essential if your system and network resources are scarce and your application is not time-critical. If you call tTerm while there are active connections, the TSI issues a forced tDisconnect on the active connections before it brings down its service structure. Issuing tTerm while active connections exist should be the last option.

----

**Note**

The successful writing of client trace files to the client file system requires successful completion of the tTerm function. When the client application abnormally terminates, TSI trace files are not written.

---

**Synopsis**

```c
int tTerm ( void );
```

**Parameters**

None
Returns

If this function completes successfully, it returns OK. Otherwise it returns ERROR, and `tserrno` contains one of the following error codes (listed alphabetically):

- **TSI_TERM_ERR_ACT_REM_FAILED**  The TSI failed to terminate its internal active connection queue.
  
  Action: Severe error; terminate your application and try again.

- **TSI_TERM_ERR_ACT_TERM_FAILED**  The TSI failed to terminate its internal active connection queue.
  
  Action: Severe error; terminate your application and try again.

- **TSI_TERM_ERR_BUF_FREE_FAILED**  The TSI failed in removing buffers used by the application.
  
  Action: Severe error; terminate your application and try again.

- **TSI_TERM_ERR_BUFM_TERM_FAILED**  The TSI failed to terminate buffer management services.
  
  Action: Review your TSI configuration services and TSI error log.

- **TSI_TERM_ERR_DISC_FAILED**  The TSI failed to disconnect an active connection.
  
  Action: Severe error; terminate your application and try again.

- **TSI_TERM_ERR_LOG_END_FAILED**  The TSI failed to terminate its internal logging and tracing facility.
  
  Action: Check your logging-related and tracing-related parameters in the currently used TSI configuration file.

- **TSI_TERM_ERR_NEVER_INIT**  The TSI was never initialized with a call to tinit.
Action: Review your program's logic, and try again.

**TSI_TERM_ERR_RES_FREE_FAILED**  The TSI failed to free connection-related resources.

Action: Review the TSI connection log, terminate your application and try again.

For additional error codes, refer to Appendix A.
4.13 tWrite

The tWrite function requests the TSI to send data to a peer TSI application over the specified connection ID.

For both blocking and non-blocking I/O connections, a positive return value signifies a successful completion of the write. If the write is for a connection using non-blocking I/O, note the following:

- If tWrite returns ERROR and tserrno is set to TSI_WRIT_ERR_QFULL, the output queue for this connection is full and cannot accept any more requests at this time.

- If tWrite returns ERROR and tserrno is set to TSI_EWOULDBLOCK, the write could not be completed immediately, but was queued to the output queue. Upon completion, the write can be retrieved from the queue by calling tPoll with the TSI_POLL_WRITE_COMPLETE option.

Synopsis

```c
int tWrite( int iConnID, /* Connection ID from tConnect/tListen */
            char *pBuf,  /* Source buffer for transfer */
            int iBufLen, /* Number of bytes to transfer */
            int iWritePriority ); /* Normal or expedite queueing */
```

Parameters

**int iConnID**  This field uniquely identifies an active connection serviced by the TSI. This ID is returned from a tConnect or tListen function call.

**char *pBuf**  This field contains the address of the write buffer to be sent to the peer TSI application. This parameter must not be NULL. The buffer can be allocated using the tBufAlloc function or a similar C function. However, if the buffer is allocated by a function other than tBufAlloc, your application must provide sufficient header space for the TSI control information (usOverhead field on page 77).
Caution

Normally, if the client application allocates the write buffer using \texttt{tBufAlloc}, it must release the buffer using \texttt{tBufFree} when the write request completes. A server-resident application is an exception (see Section 2.3.7 on page 43).

\begin{verbatim}
int iBufLen  This field specifies the number of bytes to be sent. This value must be greater than zero and less than the maximum buffer size configured (\texttt{iMaxBufSize}, page 77, as reported by the \texttt{tPoll} call with the \texttt{TSI_POLL_GET_SYS_CFG} option).

int iWritePriority  This field specifies the priority of the write operation. Your application can use this field to expedite a request to the peer TSI application. The default type is a normal write operation. Note that this field has no meaning for a connection using blocking I/O, as only one write can be pending at a time. Valid types are:

\begin{itemize}
  \item \texttt{TSI_WRITE_EXPEDITE}  If this type is used, your current request is inserted before any output requests whose actual output operation has not started and after any output request that has already started or that was issued with \texttt{TSI_WRITE_EXPEDITE}. This exercises the priority queue concept.
  \item \texttt{TSI_WRITE_NORMAL}  If this type is used, your output request is added to the end of the connection internal output queue. This exercises the FIFO concept of queue.
\end{itemize}

Returns

The \texttt{tWrite} function returns the transfer count if the TSI successfully completes the I/O. Otherwise, the return code is \texttt{ERROR}, and \texttt{tserrno} contains one of the following error codes (listed alphabetically):
\end{verbatim}
TSI_EWOULD_BLOCK  The requested action could not be completed immediately. The TSI would have blocked this operation if your connection was using blocking I/O.

Action: Use tPoll to check whether your request completed. You can program your application to be notified by one of the IOCH routines that you provided when you invoked the tInit, tConnect, or tListen function. Refer to Appendix B for information on managing TSI applications using non-blocking I/O.

TSI_WRIT_ERR_INVALID_BUF  Your application invoked this function with a NULL pBuf pointer.

Action: Correct your application and try again.

TSI_WRIT_ERR_INTERNAL  A TSI internal error was detected during write processing. The TSI log will indicate the specific error.

Action: Have the TSI error log available, and contact Protogate.

TSI_WRIT_ERR_INVALID_ID  Your connection ID is no longer valid.

Action: Review your log, terminate your program, and try again.

TSI_WRIT_ERR_INVALID_LENGTH  The buffer length (iBufLen) must be greater than zero and less than the maximum buffer length allowed by the TSI.

Action: Use tPoll to obtain the maximum buffer length allowed by the TSI (iMaxBufSize). Correct your application and try again.

TSI_WRIT_ERR_INVALID_STATE  This connection is not in a proper state to accept a write request.

Action: Review your program and try again.
TSI_WRIT_ERR_INVALID_WRITE_TYPE  

*tWrite* allows either TSI_WRITE_NORMAL or TSI_WRITE_EXP.

Action: Review your program's logic, and try again.

TSI_WRIT_ERR_LIMIT_EXCEEDED  

This connection has a large number of I/O errors that exceeded the maximum number of errors allowed.

Action: Consider increase the maximum I/O errors parameter in the connection definition entry. Review your operating environments.

TSI_WRIT_ERR_NEVER_INIT  

The TSI was never initialized. Your application must initialize the TSI (*tInit*) before it can use it.

Action: Review your application.

TSI_WRIT_ERR_QADD_FAILED  

The TSI failed to add your write request to its internal I/O queues for this connection.

Action: Severe error; terminate your application and try again.

TSI_WRIT_ERR_QFULL  

The TSI cannot accept more write requests because its output queue is full.

Action: Your application must remove complete or timed-out write requests, using the *tPoll* call, before it can request more writes. Review your program and handle this error accordingly.

TSI_WRIT_ERR_SELECT  

The TSI received an error from a system select call during a write attempt using blocking I/O.

Action: Retry the operation. If the error persists, review your application.

TSI_WRIT_ERR_SOCK_CLOSED  

A system write attempt failed, and TSI has closed the connection.
Action: Cancel all outstanding read and write requests, and free the buffers. Close the connection. After a successful close, another connection can be attempted. Examine the log files for abnormal conditions prior to the write failure.

**TSI_WRIT_ERR_UNBIND** This connection between Freeway and the client application has been closed. The system has performed “Unbind” processing. The connection was closed either because of a “Force Unbind” received from the peer entity (Freeway or client), or because of a failure with the I/O connection.

Action: Cancel all outstanding read/write requests and free the buffers. Close the connection. After a successful close, another connection can be attempted. Examine the log files on the peer system (if the error occurred in the client application, examine the Freeway log).

**TSI_WRIT_ERR_WRITE_TIMEOUT** Your request was not completed in timely manner.

Action: Increase the timeout value in the configuration for this connection.

For additional error codes, refer to Appendix A.
Appendix A

TSI Common Error Codes

This chapter describes the internal and command-specific TSI error codes.

Note
While developing your TSI application, if a particular error occurs consistently, contact Protogate for further assistance.

A.1 Internal Error Codes

To assist you in debugging your application, the following codes (listed alphabetically) describe internal error conditions of TSI services. These codes are returned in the global variable tserrno.

TSI_CINIT_ERR_CFG_LOAD_FAILED  TSI failed to load the configuration entry from the TSI configuration file.

Action: Ensure that your tsicfg program and your TSI library is up-to-date. Make sure your application uses a correct connection entry name through the first parameter of the tConnect or tListen function call. Note that the connection entry name is case-sensitive.

TSI_CINIT_ERR_DEQ_FAILED  TSI failed to dequeue its internal active connection entry.

Action: If this error occurs consistently, it is considered a severe error; contact Protogate for further assistance.
TSI_CINIT_ERR_GET_ENTRY_FAILED  TSI failed to add the newly created connection entry to its internal connection queue.

Action: Severe error. Terminate your application and try again.

TSI_CINIT_ERR_QFULL  TSI failed to add a new connection entry to its internal connection queue.

Action: Consider increasing the number of connections (MaxConns parameter) through the TSI configuration file. If this error occurs consistently even though the maximum number of connections is adequately defined, contact Protogate for further assistance.

TSI_CINIT_ERR_RESA_FAILED  TSI failed to allocate system resources for this newly created connection.

Action: Review more error messages to pinpoint the problems. Recalculate your system resource requirements and adjust your configuration file accordingly.

TSI_CLNT_ERR_EVT_PROC_FAILED  TSI failed to complete its internal event processing routine.

Action: If this error occurs consistently, contact Protogate for further assistance. VxWorks only.

TSI_CLNT_ERR_TASK_VAR_FAILED  TSI failed to add task variables to the VxWorks system. Severe error.

Action: Contact Protogate for further assistance. VxWorks only.

TSI_EVTG_ERR_ILL_CMD_TYPE  A TSI command packet with an invalid command field was processed. Severe error.

Action: Terminate your application and try again.
TSI_INTERN_ERR_01  A TSI error occurred during write processing without being correctly identified. TSI_WRIT_ERR_INTERNAL is returned to the application.

Action: If this error occurs consistently, contact Protogate for further assistance.

TSI_INTERN_ERR_02  A TSI error occurred during read processing without being correctly identified. TSI_READ_ERR_INTERNAL is returned to the application.

Action: If this error occurs consistently, contact Protogate for further assistance.

TSI_INTERN_ERR_03  A TSI_INTERN_ERR_01 or TSI_INTERN_ERR_02 error has been recognized in returning a buffer from a poll request for read/write completions. TSI_POLL_ERR_INTERNAL is returned to the application.

Action: If this error occurs consistently, contact Protogate for further assistance.

TSI_INT_ERR_SELECT_FAILED  TSI failed to perform a select on its active socket file descriptors. Severe error.

Action: Terminate your application and try again. Non-blocking I/O with socket interface only.

TSI_INTR_ERR_CLIENT_SPAWN_FAILED  TSI failed to spawn its internal IOClient task. Severe error.

Action: Ensure that your VxWorks system resources are properly configured. Make sure that your linkage completes successfully. VxWorks only.

TSI_INTR_ERR_SEM_FAILED  TSI failed to create a binary semaphore for its internal usage. Severe error.

Action: Ensure that your VxWorks system resource is properly configured. VxWorks only.
TSI_INTR_ERR_SERVER_SPAWN_FAILED  TSI failed to spawn its internal IOServer task. Severe error.

Action: Ensure that your VxWorks system resources are properly configured. Make sure that your linkage completes successfully. VxWorks only.

TSI_INTR_ERR_SIG_ERR  TSI failed to install its SIGIO signal to UNIX or a UNIX-like system. Severe error.

Action: Make sure your TSI application does not use SIGIO signal. UNIX or UNIX-like only.

TSI_INTR_ERR_SOCKET_SPAWN_FAILED  TSI failed to spawn its internal IOSocket task. Severe error.

Action: Ensure that your VxWorks system resources are properly configured. Make sure that your linkage completes successfully. VxWorks only.

TSI_IO_ERR_INVALID_EXEC_STATE  The TSI has gone through an invalid state transition. Severe error.

Action: Make sure that your network is functioning properly. Terminate your application and try again.

TSI_LOGI_ERR_LOG_OPEN_FAILED  TSI failed to open its internal error log.

Action: Check access authority for the current directory.

TSI_LOGI_ERR_TRC_OPEN_FAILED  TSI failed to open the trace file for input.

Action: Check access authority for the current directory.

TSI_MEMI_ERR_CALLOC_FAILED  TSI failed to obtained adequate memory resources to operate.
A: TSI Common Error Codes

TSI_MEMI_ERR_NEVER_INIT  TSI is not initialized correctly.

Action: Review your program logic and try again. VxWorks with shared-memory package only.

TSI_RECV_PACK_ERR_QADD_FAILED  The TSI failed to queue a read for an incoming control packet. Severe error.

Action: Ensure that your TSI configuration file defines an adequate queue size.

TSI_RECV_PACK_ERR_NO_BUFS  The TSI failed to obtain a buffer for an internal control packet.

Action: Ensure that your TSI configuration file includes an adequate number of buffers.

TSI_RESA_ERR_BUFIO_FAILED  TSI failed to obtain two internal IO buffers for the newly created connection. The connection request will fail.

Action: Ensure that your TSI configuration file includes adequate buffer configuration.

TSI_RESA_ERR_INVALID_PROT  TSI failed to recognize the protocol specified in the configuration file.

Action: Ensure that your tsicfg and your TSI library are up-to-date.
TSI_RESA_ERR_QIO_FAILED  TSI failed to create its internal IO queue for the newly requested connection.

Action: Recalculate your system resource requirements. If needed, re-configure your TSI to match your limited system resources and try again.

TSI_SEND_PACK_ERR_NO_BUFS  The TSI failed to obtain a buffer for an internal control packet.

Action: Ensure that your TSI configuration file includes an adequate number of buffers.

TSI_SEND_PACK_ERR_QADD_FAILED  The TSI failed to queue a write for an outgoing control packet. Severe error.

Action: Ensure that your TSI configuration file defines an adequate queue size.

TSI_SHM_ERR_CTL_PROC_QTERM_FAIL  An error occurred while terminating the TSI shared-memory queues.

Action: If this error occurs consistently, contact Protogate for assistance.

TSI_SHM_ERR_CTRL_PROC_FAILED  An error occurred while the TSI was processing a shared-memory control packet.

Action: If this error occurs consistently, contact Protogate for assistance.

TSI_SHM_ERR_INVALID_STATUS  An invalid status was reached while executing a shared-memory open. Severe error.

Action: If this error occurs consistently, contact Protogate for assistance.

TSI_SHM_ERR_KILL_SIG1_FAILED  The TSI failed to send SIGUSR1 to process a shared-memory open.
Action: Make sure that your TSI application does not use SIGUSR1 on VxWorks. Terminate your application and try again. If this error occurs consistently, contact Protogate for assistance.

TSI_SHM_ERR_NOT_OPEN   The TSI attempted to close a shared-memory transport that was not open.

Action: Review your application logic. Terminate your application and try again.

TSI_SHM_ERR_NO_BUFS   The TSI failed to obtain a buffer for an internal control packet.

Action: Ensure that your TSI configuration file includes an adequate number of buffers.

TSI_SHM_ERR_NO_RESOURCE   The TSI failed to initialize the shared-memory queues for a connection.

Action: Your application probably has inadequate memory resources; check the TSI configuration.

TSI_SHM_ERR_OPEN_PROC_NO_BUFS   The TSI failed to obtain a buffer for an internal control packet.

Action: Ensure that your TSI configuration file includes an adequate number of buffers.

TSI_SHM_ERR_OPEN_PROC_NO_LSTN   A shared-memory open request was rejected. No outstanding listens were posted to the requested Transport Name.

Action: Ensure that your TSI configuration file includes an adequate number of buffers.

TSI_SHM_ERR_PIPE_OPENFAILED   The TSI failed to open its peer's shared-memory open request pipe.
Action: Ensure that the client connection definition's server parameter matches the server main definition's servername parameter in the corresponding TSI configuration files.

TSI_SHM_ERR_PIPE_READ_FAILED A TSI shared-memory listening connection incurred an error attempting to read from the shared-memory request pipe.

Action: Severe error. Terminate your application and try again.

TSI_SHM_ERR_PIPE_SEL_FAILED A TSI shared-memory listening connection incurred an error attempting to perform a select on the shared-memory request pipe.

Action: Severe error. Terminate your application and try again.

TSI_SHM_ERR_PIPE_WRITE_FAILED A TSI shared-memory client connection incurred an error attempting to write an open request to the shared-memory request pipe.

Action: Severe error. Terminate your application and try again.

TSI_SHM_ERR_QREM_AT_CTLBUF_FAIL The TSI incurred an error attempting to remove a TSI control packet from the connection's queue.

Action: Severe error. Terminate your application and try again.

TSI_SOCK_ERR_ACCEPT_ERR An accept on a TSI listening socket failed.

Action: Check system resources to ensure that the system limit for open file descriptors has not been reached.

TSI_SOCK_ERR_ALREADY_CLOSED The TSI attempted to close a socket that was not open.

Action: Internal logic error. Contact Protogate to report problem.

TSI_SOCK_ERR_CLOSE_ERR The TSI incurred an error while closing an open socket.
A: TSI Common Error Codes

Action: Terminate your application and try again.

TSI_SOCK_ERR_CONNECT_ERR The TSI failed to connect a client socket.

Action: Severe error. Terminate your application and try again.

TSI_SOCK_ERR_GET_HNAME_FAILED The TSI failed to convert the server host
name to an IP address.

Action: Ensure that the TSI configuration file contains a valid host name for the
serverName parameter.

TSI_SOCK_ERR_INVALID_IOQ_STATUS The IO status field of a TSI socket packet
contains an invalid status. Severe error.

Action: Terminate your application and try again.

TSI_SOCK_ERR_INVALID_RTN_VALUE The TSI generated an invalid internal value.

Action: If this error occurs consistently, contact Protogate for further assistance.

TSI_SOCK_ERR_NO_SOCKET The TSI did not have a valid socket file descriptor to
perform a socket open or listen.

Action: Terminate your application and try again.

TSI_SOCK_ERR_READ_FAILED The TSI encountered an error performing a read
from a socket.

Action: Check the TSI log file just prior to this error for more detailed informa-
tion.

TSI_SOCK_ERR_SELECT_ERR The TSI encountered an error performing a select on
a socket.

Action: Terminate your application and try again.
TSI_SOCK_ERR_SEM_CREATE_FAILED  TSI failed to create an internal semaphore for its internal socket task.

Action: Recalculate your system resource requirements. If needed, reconfigure your TSI configuration file. VxWorks only.

TSI_SOCK_ERR_SOCKET_CLOSED  A socket that the TSI was attempting to perform IO upon closed.

Action: Check the peer software and attempt to discover why the peer socket closed.

TSI_SOCK_ERR_TASK_VAR_FAILED  TSI failed to add task variables to the VxWorks system. Severe error.

Action: Contact Protogate for further assistance. VxWorks only.

TSI_SOCK_ERR_WRITE_FAILED  The TSI encountered an error performing a write to a socket.

Action: Check the TSI log file just prior to this error for more detailed information.

TSI_SOCKETA_ERR_BIND_FAILED  TSI failed to bind a well-known port to its listening socket.

Action: Make sure that your well-known port is not being used by any other application within your system. Terminate your application and try again.

TSI_SOCKETA_ERR_FCNTL_FAILED  TSI failed to set the socket file descriptor to a non-blocking I/O mode.

Action: Ensure that your TCP/IP software package supports non-blocking I/O and that your application is configured to use non-blocking I/O. Otherwise, contact Protogate for further assistance.
TSI_SOCKA_ERR_LISTEN_FAILED  TSI failed to listen for a connection request through a well-known port specified in the TSI configuration file.

Action: If this error occurs consistently, contact Protogate for further assistance.

TSI_SOCKA_ERR_SETOPT_FAILED  TSI failed to set TCP/IP TCP_NODELAY and KEEP_ALIVE options.

Action: Ensure that your TCP/IP socket interface package supports these socket options. If your TCP/IP software does not support these options, consider turning them off using the TSI configuration file. Otherwise, contact Protogate for further assistance.

TSI_SOCKA_ERR_SOCKET_FAILED  TSI failed to obtain a new socket for the newly requested connection.

Action: Consider increasing the maximum number of file descriptors through system resource services (UNIX: getrlimit function).

TSI_SRVR_ERR_EVT_PROC_FAILED  TSI failed to complete its internal event processing routine.

Action: If this error occurs consistently, contact Protogate for further assistance. VxWorks only.

TSI_SRVR_ERR_PIPE_CREATE_FAILED  TSI failed to create a named pipe for its internal shared-memory protocol package.

Action: If you develop a server-resident application, make sure your linkage completes successfully. VxWorks only.

TSI_SRVR_ERR_PIPE_OPEN_FAILED  TSI failed to open its internal named pipe.

Action: Terminate your application and try again. VxWorks only.
TSI_SRVR_ERR_SIG_ERR  TSI failed to install an interrupt handler for SIGUSR1 signal.

Action: Ensure that your TSI application in the VxWorks environment does not use the SIGUSR1 signal. VxWorks only.

TSI_SRVR_ERR_TASK_VAR_FAILED  TSI failed to add task variables to VxWorks system. Severe error.

Action: Contact Protogate for further assistance. VxWorks only.
### A.2 Command-Specific Error Codes

Table A-1 lists alphabetically all the error codes related to specific TSI commands described in Chapter 4. These codes are returned in the global variable `tserrno`.

**Table A-1: TSI Command-specific Error Codes**

<table>
<thead>
<tr>
<th>Command(s)</th>
<th>Error Code</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tBufAlloc</td>
<td>TSI_BUFA_ERR_NEVER_INIT</td>
<td>page 81</td>
</tr>
<tr>
<td></td>
<td>TSI_BUFA_ERR_NO_BUFS</td>
<td>page 81</td>
</tr>
<tr>
<td></td>
<td>TSI_BUFA_ERR_SEVERE_ERR</td>
<td>page 81</td>
</tr>
<tr>
<td>tBufFree</td>
<td>TSI_BUFF_ERR_INVALID_BUF</td>
<td>page 82</td>
</tr>
<tr>
<td></td>
<td>TSI_BUFF_ERR_NEVER_INIT</td>
<td>page 82</td>
</tr>
<tr>
<td></td>
<td>TSI_BUFF_ERR_NOT_ALLOCATED</td>
<td>page 82</td>
</tr>
<tr>
<td></td>
<td>TSI_BUFF_ERR_SEVERE_ERR</td>
<td>page 83</td>
</tr>
<tr>
<td>tConnect</td>
<td>TSI_CONN_ERR_CLOSE_FAILED</td>
<td>page 86</td>
</tr>
<tr>
<td></td>
<td>TSI_CONN_ERR_CONN_INIT_FAILED</td>
<td>page 86</td>
</tr>
<tr>
<td></td>
<td>TSI_CONN_ERR_INVALID_PROT</td>
<td>page 86</td>
</tr>
<tr>
<td></td>
<td>TSI_CONN_ERR_INVALID_STATE</td>
<td>page 86</td>
</tr>
<tr>
<td></td>
<td>TSI_CONN_ERR_OPEN_FAILED</td>
<td>page 86</td>
</tr>
<tr>
<td></td>
<td>TSI_CONN_ERR_QADD_FAILED</td>
<td>page 86</td>
</tr>
<tr>
<td></td>
<td>TSI_CONN_ERR_RETRY_EXCEEDED</td>
<td>page 87</td>
</tr>
<tr>
<td></td>
<td>TSI_CONN_ERR_SOCK_ALLOC_FAILED</td>
<td>page 87</td>
</tr>
<tr>
<td></td>
<td>TSI_CONN_ERR_TINIT_FAILED</td>
<td>page 87</td>
</tr>
<tr>
<td>tDisconnect</td>
<td>TSI_DISC_ERR_CLOSE_FAILED</td>
<td>page 89</td>
</tr>
<tr>
<td></td>
<td>TSI_DISC_ERR_DISC_FAILED</td>
<td>page 89</td>
</tr>
<tr>
<td></td>
<td>TSI_DISC_ERR_INVALID_ID</td>
<td>page 89</td>
</tr>
<tr>
<td></td>
<td>TSI_DISC_ERR_INVALID_STATE</td>
<td>page 89</td>
</tr>
<tr>
<td></td>
<td>TSI_DISC_ERR_INVALID_TYPE</td>
<td>page 89</td>
</tr>
<tr>
<td></td>
<td>TSI_DISC_ERR_NEVER_INIT</td>
<td>page 90</td>
</tr>
<tr>
<td></td>
<td>TSI_DISC_ERR_Q_NOT_EMPTY</td>
<td>page 90</td>
</tr>
<tr>
<td></td>
<td>TSI_DISC_ERR_QADD_FAILED</td>
<td>page 90</td>
</tr>
<tr>
<td></td>
<td>TSI_DISC_ERR_SEVERE_ERROR</td>
<td>page 90</td>
</tr>
<tr>
<td>Command(s) Causing Error</td>
<td>Error Code</td>
<td>Reference Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>tConnect</td>
<td>TSI_EWOULDBLOCK</td>
<td>page 87</td>
</tr>
<tr>
<td>tDisconnect</td>
<td></td>
<td>page 90</td>
</tr>
<tr>
<td>tListen</td>
<td></td>
<td>page 97</td>
</tr>
<tr>
<td>tRead</td>
<td></td>
<td>page 111</td>
</tr>
<tr>
<td>tWrite</td>
<td></td>
<td>page 122</td>
</tr>
<tr>
<td>tInit</td>
<td>TSI_INIT_ERR_ACT_ADD_REM_FAILED</td>
<td>page 92</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_ACT_QINIT_FAILED</td>
<td>page 92</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_ALREADY_INIT</td>
<td>page 92</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_BUF_ADD_REM_FAILED</td>
<td>page 92</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_BUF_QINIT_FAILED</td>
<td>page 93</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_CFG_LOAD_FAILED</td>
<td>page 93</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_LOG_INIT_FAILED</td>
<td>page 93</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_NO_MEM</td>
<td>page 93</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_NO_RESOURCE</td>
<td>page 93</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_NO_TRACE_BUFFER</td>
<td>page 94</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_SETRLIMIT_FAILED</td>
<td>page 94</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_TASK_VAR_FAILED</td>
<td>page 94</td>
</tr>
<tr>
<td></td>
<td>TSI_INIT_ERR_TSICB_ALLOC_FAILED</td>
<td>page 94</td>
</tr>
</tbody>
</table>
### Table A-1: TSI Command-specific Error Codes (Cont’d)

<table>
<thead>
<tr>
<th>Command(s) Causing Error</th>
<th>Error Code</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tListen</td>
<td>TSI_LSTN_ERR_CLOSE_FAILED</td>
<td>page 97</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_CONN_INIT_FAILED</td>
<td>page 97</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_INVALID_PROT</td>
<td>page 97</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_INVALID_STATE</td>
<td>page 97</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_LSTN_FAILED</td>
<td>page 98</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_NO_LISTENERS</td>
<td>page 98</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_NOT_SERVER</td>
<td>page 98</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_QADD_FAILED</td>
<td>page 98</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_QFULL</td>
<td>page 98</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_QINIT_FAILED</td>
<td>page 98</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_SETOPT_FAILED</td>
<td>page 98</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_SOCK_ALLOC_FAILED</td>
<td>page 99</td>
</tr>
<tr>
<td></td>
<td>TSI_LSTN_ERR_TINIT_FAILED</td>
<td>page 99</td>
</tr>
<tr>
<td>tPoll</td>
<td>TSI_POLL_ERR_BAD_PTR</td>
<td>page 103</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_BUF_NOT_FOUND</td>
<td>page 103</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_GETLIST_FAILED</td>
<td>page 103</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_INTERNAL</td>
<td>page 103</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_INVALID_ID</td>
<td>page 104</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_INVALID_IOCTL</td>
<td>page 104</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_INVALID_REQ_TYPE</td>
<td>page 104</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_NEVER_INIT</td>
<td>page 104</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_OVERFLOW</td>
<td>page 104</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_QEMPTY</td>
<td>page 104</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_QREM_FAILED</td>
<td>page 105</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_READ_NOT_COMPLETE</td>
<td>page 105</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_READ_TIMEOUT</td>
<td>page 105</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_SOCK_CLOSED</td>
<td>page 105</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_UNBIND</td>
<td>page 106</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_WRITE_NOT_COMPLETE</td>
<td>page 106</td>
</tr>
<tr>
<td></td>
<td>TSI_POLL_ERR_WRITE_TIMEOUT</td>
<td>page 106</td>
</tr>
</tbody>
</table>
Table A-1: TSI Command-specific Error Codes (Cont’d)

<table>
<thead>
<tr>
<th>Command(s) Causing Error</th>
<th>Error Code</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tPost</td>
<td>TSI_POST_ERR_CLIENT_INVALID_SEM</td>
<td>page 107</td>
</tr>
<tr>
<td></td>
<td>TSI_POST_ERR_NEVER_INIT</td>
<td>page 107</td>
</tr>
<tr>
<td></td>
<td>TSI_POST_ERR_SERVER_INVALID_SEM</td>
<td>page 108</td>
</tr>
<tr>
<td>tRead</td>
<td>TSI_READ_ERR_INTERNAL</td>
<td>page 111</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_INVALID_BUF</td>
<td>page 111</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_INVALID_ID</td>
<td>page 111</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_INVALID_LENGTH</td>
<td>page 111</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_INVALID_STATE</td>
<td>page 111</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_LIMIT_EXCEEDED</td>
<td>page 112</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_NEVER_INIT</td>
<td>page 112</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_NO_BUFS</td>
<td>page 112</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_OVERFLOW</td>
<td>page 112</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_QADD_FAILED</td>
<td>page 112</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_QFULL</td>
<td>page 112</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_READ_TIMEOUT</td>
<td>page 113</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_SELECT</td>
<td>page 113</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR.SOCK_CLOSED</td>
<td>page 113</td>
</tr>
<tr>
<td></td>
<td>TSI_READ_ERR_UNBIND</td>
<td>page 113</td>
</tr>
<tr>
<td>tSyncSelect</td>
<td>TSI_SYNCSELECT_ERR_INVALID_STATE</td>
<td>page 115</td>
</tr>
<tr>
<td></td>
<td>TSI_SYNCSELECT_ERR_SELECT_ERROR</td>
<td>page 116</td>
</tr>
<tr>
<td>rTerm</td>
<td>TSI_TERM_ERR_ACT_REM_FAILED</td>
<td>page 118</td>
</tr>
<tr>
<td></td>
<td>TSI_TERM_ERR_ACT_TERM_FAILED</td>
<td>page 118</td>
</tr>
<tr>
<td></td>
<td>TSI_TERM_ERR_BUF_FREE_FAILED</td>
<td>page 118</td>
</tr>
<tr>
<td></td>
<td>TSI_TERM_ERR_BUFM_TERM_FAILED</td>
<td>page 118</td>
</tr>
<tr>
<td></td>
<td>TSI_TERM_ERR_DISC_FAILED</td>
<td>page 118</td>
</tr>
<tr>
<td></td>
<td>TSI_TERM_ERR_LOG_END_FAILED</td>
<td>page 118</td>
</tr>
<tr>
<td></td>
<td>TSI_TERM_ERR_NEVER_INIT</td>
<td>page 118</td>
</tr>
<tr>
<td></td>
<td>TSI_TERM_ERR_RES_FREE_FAILED</td>
<td>page 119</td>
</tr>
</tbody>
</table>
### Table A-1: TSI Command-specific Error Codes (Cont’d)

<table>
<thead>
<tr>
<th>Command(s) Causing Error</th>
<th>Error Code</th>
<th>Reference Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tWrite</td>
<td>TSI_WRIT_ERR_INTERNAL</td>
<td>page 122</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_INVALID_BUF</td>
<td>page 122</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_INVALID_ID</td>
<td>page 122</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_INVALID_LENGTH</td>
<td>page 122</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_INVALID_STATE</td>
<td>page 122</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_INVALID_WRITE_TYPE</td>
<td>page 123</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_LIMIT_EXCEEDED</td>
<td>page 123</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_NEVER_INIT</td>
<td>page 123</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_QADD_FAILED</td>
<td>page 123</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_QFULL</td>
<td>page 123</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_SELECT</td>
<td>page 123</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_SOCK_CLOSED</td>
<td>page 123</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_UNBIND</td>
<td>page 124</td>
</tr>
<tr>
<td></td>
<td>TSI_WRIT_ERR_WRITE_TIMEOUT</td>
<td>page 124</td>
</tr>
</tbody>
</table>
B.1 UNIX Environment

In a UNIX environment, the TSI gains access to non-blocking I/O services through the use of the UNIX signal delivery mechanism. At initialization, TSI installs a signal handler (or IOCH) to be executed upon delivery of the signal. When a signal is delivered to a TSI application, the TSI IOCH immediately suspends the delivery of that signal again until it completes its I/O services through the IOCH function. TSI exits the IOCH either when it runs out of system resources to accept additional I/O, or when it has no additional I/O to accept. In either case, system resources are tied up by TSI while it is in the IOCH function unless it is interrupted by another system service request (that is, another signal delivery) with a higher priority than its own. A TSI application using non-blocking I/O can provide TSI with its own IOCH; the TSI IOCH subsequently invokes the application IOCH after it completes its I/O services.

In short, non-blocking I/O operation is not only complex but also expensive. Therefore, it requires careful planning and design so that your application uses the system resources wisely.

B.1.1 Blocking I/O operations

Blocking I/O operation requires no IOCHs. Blocking I/O does not use any signal delivery mechanism to handle the delivery of data. Blocking I/O allows the orderly execution of your application and requires far less system resources than non-blocking I/O. Blocking I/O is also easier to debug and troubleshoot than non-blocking I/O. Careful design through the isolation of system and protocol dependency allows your applica-
tion to work using either blocking I/O or non-blocking I/O. The TSI services allow your application to switch from blocking to non-blocking I/O, and vice versa, without the recompilation of your application code.

It is not efficient to handle multiple connections under blocking I/O operation, because your application is blocked until the data arrives or TSI times out while waiting. While your application is waiting for I/O in one connection, data from other connections is blocked.

B.1.2 Non-Blocking I/O Operations

The TSI uses the SIGIO signal for its non-blocking BSD socket interface. Therefore, your application should not block the delivery of SIGIO signals (for example, sigprocmask) at any time, especially when expecting data from the network.

If you use non-blocking I/O, design your application with robust IOCH function(s). Also, the application IOCH should perform as little work as possible and, before it exits, use some notification techniques to awaken the main routines to perform the remaining tasks. Some possible notification techniques are system semaphores, sleep and wakeup calls using the SIGALRM signal, and so on.

B.1.3 SOLARIS use of SIGALRM

The use of a default signal handler through SIGALRM signal can cause a system core dump inside the SOLARIS internal SIGALRM signal handler. You can work around it by providing your own signal handler for SIGALRM. The following code segment assists you in setting up a SIGALRM handler for the SIGALRM signal:
void genSigHdlr ( int signal )
{
    return;
}

void main ( )
{
    struct sigaction       SigAction;

    SigAction. sa_handler = genSigHdlr;
    sigfillset (&SigAction. sa_mask);
    SigAction. sa_flags = 0;

    if (sigaction (SIGALRM, &SigAction, (struct sigaction *)NULL) ==
        ERROR)
    {
        fprintf (stderr, "sigaction failed %d\n", errno);
        return ERROR;
    }
    ..... return OK;
}

Notice that genSigHdlr does nothing but return to the system.

B.1.4 Polling I/O Operations

Your application can implement polling I/O operations if it uses TSI with non-blocking
I/O but provides no IOCH functions. Since your application provides no mechanism
for TSI to notify it when an I/O condition occurs, your application must poll TSI for the
completion of I/O requests that it posts to TSI. Polling I/O operations involve the tPoll
function (Section 4.8 on page 100). Polling I/O is helpful if your application manages
multiple connections, data arrives at a predictable rate, and the timing of data is not
critical.

B.2 VxWorks Environment

The TSI operates only in a VxWorks environment that is similar to that of the Freeway
server. VxWorks has several features similar to UNIX; however, it has a unique operat-
ing environment and a real-time operating system. The use of the TSI by an application
that runs on the Freeway server is often called a server-resident application (SRA). The SRA can be configured to interact with Protogate's message multiplexor subsystem through the shared-memory transport mechanism supported by TSI, or it can be configured to interact with other systems using the BSD socket interface which is also supported by TSI. Whichever transport your SRA program uses, you should understand not only the VxWorks operating system but also the way the Freeway server is configured and how Protogate implements TSI under VxWorks. For more information on SRAs, see Section 2.3.7 on page 43 and the Freeway Server-Resident Application and Server Toolkit Programmer Guide.

B.2.1 Blocking I/O Operations

Blocking I/O in VxWorks is similar to that of the UNIX environment.

B.2.2 Non-Blocking I/O Operations

Non-blocking I/O in VxWorks with Protogate's Freeway server requires your application to cooperate with other tasks. VxWorks on Freeway is configured to operate in a cooperative manner. This means that VxWorks operates as a non-preemptive multitasking environment. When your application does not have data to be processed, it must relinquish the CPU so that other tasks can run. You can use your own interrupt service routine to notify or resume your application when its data arrives.

The TSI uses a binary semaphore to support non-blocking I/O delivery from both network and shared-memory environments. Since VxWorks running on Freeway is configured for a cooperative environment, your application must also act cooperatively. Your application must call tPost immediately before relinquishing its control to VxWorks. Your application must relinquish the control through taskDelay, binary semaphores, or other means; otherwise, only your task has control of the CPU which prevents other important tasks from running.

Your application should not use global variables freely if multiple instances of the same application are running concurrently. VxWorks global variables are shared among all
tasks unless you define them as a particular task's variables (using taskVarAdd). Task variables are expensive to maintain by VxWorks and therefore should be used sparingly.

**B.3 VMS Environment**

The TSI uses the process-level Asynchronous System Trap (AST) for non-blocking data delivery from the network. Therefore, your application should not block the delivery of ASTs (using sys$setast) at any time, especially when expecting data from the network.
Appendix

C

TSI Logging and Tracing

To support debugging efforts, TSI provides tracing and logging services to troubleshoot both application and network problems.

C.1 TSI Logging

There are two kinds of TSI logging services: general logging and connection-related logging. As the name implies, general logging includes error or information that is not related to any particular connection. Connection-related logging indicates error or information related to a specific connection. To monitor data, you must use the TSI tracing services described in Section C.2.

General logging is defined in the “main” section of the TSI text configuration file. The LogLev parameter (page 54) specifies the level of logging your application needs and can be from 0 to 7, with level 0 being no logging, level 1 being the most severe error, and 7 being the least severe. In the “main” section, the LogName parameter (page 54) defines the log file name where your logging information is to reside. The default file name is “tsilog”. If you wish logging information to be output to the screen, define LogName as “stdout.” The number of entries to “stdout” is unlimited. A disk file is limited to 1000 entries, and this number is not configurable.

Connection-related logging can be defined in each individual TSI connection definition. You can log for some connections but not for the others; and different connections can log errors at different levels. All error codes are defined in Appendix A and in each individual function description (for example, tConnect in Section 4.4 on page 84).
The following is the format of the each log entry:

SessX: TSI_YYY_ZZZ_Information(terno/errno)

where:

X is connection ID. For general logging, X will be 999. Otherwise, it indicates a connection-related entry.

YYY is brief function name of TSI. For example, if ZZZ is CONN it indicates the log entry is from tConnect function.

ZZZ can be ERR or INFO. ERR indicates an error condition; INFO indicates information only.

terrno is a TSI error code for this entry; errno is the last encountered ‘C’ errno value.

C.2 TSI Tracing

C.2.1 Trace Definitions

The TSI tracing facility captures and stores real-time data in its internal wrap-around buffer. The size of this buffer is configurable up to 1 megabyte of memory. There are two kinds of TSI tracing: general tracing and connection-specific tracing. In general tracing, trace data has no connection-specific information, whereas connection-specific trace data pertains to only one specific connection ID.

To activate tracing, first specify the TSI “main” configuration parameters. Specify the TraceSize parameter (page 55) up to 1 megabyte of memory. The TraceName parameter (page 55) defines the file name where your trace information is to reside.

Specify the level of tracing using the TraceLev parameter (page 55). This parameter defaults to zero if not defined (no tracing). The TraceLev parameter can be defined in the “main” section for general tracing or in each individual connection definition (page 56). Each connection definition can have a different TraceLev value.
The TraceLev parameter can be the sum of one or more of the following values:

1 = trace the read (input) data

2 = trace the write (output) data

4 = trace the TSI interrupt services

8 = trace the application IOCH services

16 = trace the user’s data

For example, if you want to trace both read and write data, specify 3 for the TraceLev parameter. If you want to trace read, write, and user’s data, specify 19 for the TraceLev parameter.

The most commonly used trace level is for I/O passing through the TSI service layer (TraceLev = 3). TSI also provides the interrupt and application I/O completion handler (IOCH) trace levels within TSI to assist the application in troubleshooting the IOCH mechanism. The user data trace level allows the application to store its own data in the trace buffer.

---

**Note**

TSI does not decode user data with its tsidecode program (Section C.2.2).

---

You can turn tracing on or off at any time after TSI is initialized using tPoll with the TSI_POLL_TRACE_ON or TSI_POLL_TRACE_OFF options. Tracing is done internally with the TSI trace buffer. Trace data is not written to the trace file until tTerm is called or tPoll is called with the TSI_POLL_TRACE_WRITE option. Therefore, your application should always call tTerm before it exits to the operating system. If tracing is required and is defined in the TSI configuration file, it is automatically on when tInit is called. You can use tPoll with the TSI_POLL_TRACE_STORE option to store your own...
trace buffer inside the TSI trace buffer. Refer to \texttt{tPoll} (Section 4.8 on page 100) for more information. Since TSI tracing does not involve disk I/O until \texttt{tTerm} is executed, there is little or no performance impact.

\subsection*{C.2.2 Decoded Trace Layout}

You can run either the \texttt{tsidecode} or \texttt{dlidecode} program against the trace file produced by TSI (run \texttt{dlidecode} only if your application is using DLI). The decoded output is displayed on the screen for UNIX-like systems. In VMS, the decoded output is written to the file named \texttt{tsi.sum} (or \texttt{dli.sum}). You can run \texttt{dlidecode} against the trace file produced by either DLI or TSI. Refer to the Freeway Data Link Interface Reference Guide for details on DLI tracing.

The format of the decoded trace can be described as follows. See Section C.2.3 for an actual decoded trace example.

- line 1: Protogate 2000(C) TSI Trace Decoder
- line 2: Max buffer size: xyz
- line 3: TRACE SOURCE: yyy
- line 4: @@@@@ Actual Data offset aa Size = bb
- line 5: cc: hex data and printable ascii equivalent.
- line 6: @@@@@ Decode begins
- line 7: dd(desc) Conn ee: time and date
- line 8: TSI header info:
- line 9: iHdrLen = ff iDataLen = gg uiSeqNo = hh
- line 10: iPacketType (ii) = textii iCmd (jj) = textjj
- line 11: DATA : hex data and printable ascii equivalent.

Each line of the above format is explained as follows:

- line 1: indicates the copyright and the name of the \texttt{tsidecode} program. Note that \texttt{tsidecode} can run only against the TSI trace file, unlike \texttt{dlidecode} which can run on both DLI and TSI trace files.

- line 2: prints the currently used maximum buffer size that is defined in the TSI configuration file (\texttt{MaxBufSize} on page 55). Note that the size excludes the over-
head used by TSI; it describes the maximum number of actual data bytes allowed by TSI. See Section 2.3 on page 31.

line 3: prints the source of the trace.

line 4: describes the actual offset (aa) from the beginning of the trace file where this packet is stored and the number of bytes contained in this packet has (bb). Section C.2.4 describes how to read the TSI trace file in case you want to write your own decoder to decode your own trace data that you store in TSI trace buffer using tPoll with the TSI_POLL_TRACE_STORE option.

line 5: prints the actual hex values and their equivalent printable ASCII text. The offset (cc) is the actual offset from the beginning of the packet, based on 0. Each line contains up to 16 bytes from the trace packet. Note that line 5 can be repeated if the actual size of the trace packet is more than 16 bytes long.

line 6: indicates that the actual decoding begins. This is where the headers are broken into individual fields.

line 7: dd indicates the direction of the packet; dd can be =====> to indicate an outgoing packet or <===== to indicate an incoming packet. For non-I/O related packets (for example, user's data packet), dd is either ***** or ######. If the trace packet is for I/O, desc can be READ(1)/WRITE(2) n bytes. If the packet is a non-I/O packet, desc can be one of the following:

CONNECTION INTERRUPT BEGINS(4): indicates that TSI begins its interrupt handler to process I/O requests.

CONNECTION INTERRUPT ENDS(5): indicates that TSI ends its interrupt handler routine and is ready to return to TSI.

CONNECTION ISR(3): indicates that TSI is about to call the IOCH of a specific
connection ID. The address of this IOCH was provided to TSI through the \texttt{tConnect} function.

\textbf{APPLICATION ISR BEGINS(6)}: indicates that TSI is about to call the generic IOCH that was provided by the application to TSI through the \texttt{tInit} function.

\textbf{APPLICATION ISR ENDS(7)}: indicates that the generic IOCH routine returns control to TSI.

\textbf{AT(8)}: indicates that this trace buffer belongs to the application. The \texttt{tsidecode} program will not attempt to decode this packet. You have to write your own decode function to interpret your own data packet.

line 8: begins the TSI header information.

line 9 and line 10: show detailed information of the TSI header.

line 11: indicates the details of the data in both hex values and printable ASCII equivalent.

\textbf{C.2.3 Example \texttt{tsidecode} Program Output}

The following are example segments of decoded trace output from the TSI trace file using the \texttt{tsidecode} program (output using the \texttt{dlidecode} program would be virtually identical except for the first few lines):

\begin{verbatim}
Protogate 2000(C) TSI Trace Decoder
Max buffer size: 564
-------------------------------------------------------------------------------
@@@@@ Actual Data offset 8 Size = 0
@@@@@ Decoding begins
ERROR
  DATA : 00 00 00 00 00 00 00 00 00 2e a8 3d 6a
        ...........=j
\end{verbatim}
@@ @ @@ @ Actual Data offset 20 Size = 30
000000: 00 00 00 14 00 00 00 0a 00 00 00 00 00 00 01 00 01 ................
000016: 00 01 00 01 02 34 00 14 00 00 00 00 00 00 ....4........
@@ @ @@ @ Decoding begins

====>(WRITE 30 bytes)Conn 0: Fri Oct 21 15:15:07 1994

TSI header info:
iHdrLen = 20  iDataLen = 10  uiSeqNo = 0
iPacketType(1) = CONTROL  iCmd(1) = BIND

TSI BIND info:
iMaxBufSize = 564  iMaxHdrSize = 20
Segmenting = NO  Buffering = NO  Negotiable = NO

@@ @ @@ @ Actual Data offset 62 Size = 30
000000: 00 00 00 14 00 00 00 0a 00 00 00 00 00 00 00 00 01 00 01 ................
000016: 00 01 00 01 02 34 00 14 00 00 00 00 00 00 ....4........
@@ @ @@ @ Decoding begins


TSI header info:
iHdrLen = 20  iDataLen = 10  uiSeqNo = 0
iPacketType(1) = CONTROL  iCmd(1) = BIND

TSI BIND info:
iMaxBufSize = 564  iMaxHdrSize = 20
Segmenting = NO  Buffering = NO  Negotiable = NO

@@ @ @@ @ Actual Data offset 104 Size = 20
000000: 00 00 00 14 00 00 00 00 0a 00 00 00 00 00 00 00 00 01 00 01 ................
000016: 00 01 00 01 02 34 00 14 00 00 00 00 00 00 ....4........
@@ @ @@ @ Decoding begins

====>(READ 20 bytes)Conn 0: Fri Oct 21 15:15:07 1994

TSI header info:
iHdrLen = 20  iDataLen = 0  uiSeqNo = 6
iPacketType(1) = CONTROL  iCmd(6) = ACK
@@@ Actual Data offset 136 Size = 96
  000000: 00 00 00 14 00 00 00 4c 00 00 00 01 00 02 00 00 ......L......
  000016: 00 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 0a 00 16 ..........
  000032: 00 01 00 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .........
  000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......
  000064: 69 63 70 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 icp0......
  000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...........
@@@ Decoding begins

====>(WRITE 96 bytes)Conn 0: Fri Oct 21 15:15:07 1994

TSI header info:
iHdrLen = 20  iDataLen = 76 uiSeqNo = 1
iPacketType(2) = DATA  iCmd(0) =

  DATA : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 16 00 01 00 01 ............
  DATA : 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......
  DATA : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......
  DATA : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......
  DATA : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......

@@@ Actual Data offset 244 Size = 20
  000000: 00 00 00 14 00 00 00 00 00 00 00 00 00 06 00 01 00 06 ............
  000016: 00 01 00 01 ....
@@@ Decoding begins


TSI header info:
iHdrLen = 20  iDataLen = 0 uiSeqNo = 6
iPacketType(1) = CONTROL  iCmd(6) = ACK

@@@ Actual Data offset 276 Size = 96
  000000: 00 00 00 14 00 00 00 4c 00 00 00 01 00 02 00 00 ......L......
  000016: 00 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 0a 00 16 ..........
  000032: 00 01 00 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .........
  000048: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ......
  000064: 69 63 70 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 icp0......
  000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...........
@@@ Decoding begins

TSI header info:
iHdrLen = 20  iDataLen = 76  uiSeqNo = 1
iPacketType(2) = DATA  iCmd(0) =

DATA : 00 00 00 00 00 00 00 00 00 00 0a 00 16 00 01 00 01 ............
DATA : 00 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 ................
DATA : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...............
DATA : 00 00 00 00 00 00 00 00 00 00 00 00 69 63 70 30 ............icp0
DATA : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...............
DATA : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...............

-------------------------------------------------------------------------------

@@@@@ Actual Data offset 384 Size = 20

000000: 00 00 00 14 00 00 00 40 00 00 00 01 00 02 00 00 ......@....... 000016: 00 01 00 01 ....

@@@@@ Decoding begins

<==== (READ 20 bytes) Conn 0: Fri Oct 21 15:15:07 1994

TSI header info:
iHdrLen = 20  iDataLen = 64  uiSeqNo = 1
iPacketType(2) = DATA  iCmd(0) =

-------------------------------------------------------------------------------

@@@@@ Actual Data offset 416 Size = 64

000000: 00 00 00 00 00 00 00 00 0a 00 16 00 01 00 01 ............
000016: 00 00 00 00 00 03 00 00 00 00 00 00 00 00 00 00 ............
000032: 00 00 00 14 00 00 00 00 00 00 00 00 03 00 00 46 72 65 65 77 61 79 20 52 65 6c 65 61 73 65 20 32 2e 30 00 way Release 2.0.

@@@@@ Decoding begins

<==== (READ 64 bytes) Conn 0: Fri Oct 21 15:15:07 1994

DATA : 00 00 00 00 00 00 00 00 0a 00 16 00 01 00 01 ............
DATA : 00 00 00 00 00 03 00 00 00 00 00 00 00 00 00 00 ............
DATA : 00 00 00 14 00 00 00 00 00 00 00 03 00 00 46 72 65 65 77 61 79 20 52 65 6c 65 61 73 65 20 32 2e 30 00 way Release 2.0.

-------------------------------------------------------------------------------
C.2.4 Trace Binary Format

You can use the following information to write your own decoder if you need to provide your own trace information. The trace file format is shown in Figure C-1.

```
typedef struct _TRACE_FCB
{
    int iMaxBufSize;
    short iTraceSource;
    short iPadding;
} TRACE_FCB, *PTRACE_FCB;
```

Figure C-1: TSI Trace File Format

Figure C-2 shows the TRACE_FCB ‘C’ structure.

```
typedef struct _TRACE_FCB
{
    int iMaxBufSize;
    short iTraceSource;
    short iPadding;
} TRACE_FCB, *PTRACE_FCB;
```

Figure C-2: TRACE_FCB ‘C’ Structure
Figure C-3 shows the format of each trace packet in the TSI_TRACE_HDR ‘C’ structure.

typedef struct _TSI_TRACE_HDR
{
    unsigned short usTrcType; /* type of tracing */
    unsigned short usTrcConnID; /* current connection ID */
    int iTrcDataSize; /* size of the trace packet */
    time_t tTrcTime; /* time stamp */
} TSI_TRACE_HDR, *PTSI_TRACE_HDR;

**Figure C-3:** TSI_TRACE_HDR ‘C’ Structure

### C.3 Freeway Server Tracing

Tracing service is also provided from the Freeway server. Refer to the Freeway User Guide for more information. You can use the trace information from both the client application and the Freeway server to diagnose and troubleshoot your client application. The Freeway trace service is identical to that of the client application; however, the direction of the trace is the reverse of that of the client. For example, for the same data packet, the client would indicate a read packet while the server would indicate a write packet. Care therefore must be taken when translating the two traces.
Index

A
Asynchronous I/O  see Non-blocking I/O
AsyncIO TSI parameter  54, 56, 65
Audience  11

B
Binary configuration files
management  52
Bit numbering  15
Blocking I/O  30
    signal processing  45
    UNIX  143
    VxWorks  146
BSC msgBlkSize parameter  32, 33, 34
Buffer management  31
    allocation and release  40
    buffer size negotiation  39
    cautions  41
    client buffers  32
    client configuration  35
    client versus server-resident  43
    connection-specific buffers  38
    example calculation  33
    headers  41
    ICP buffers  32
    muxCfg file  41
    see Functions
    server buffers  33
    TSI buffer pool definition  36
    using your own buffers  41
Byte ordering  15

C
Caution

data loss  73
    signal processing  45
Client buffer configuration  35
Client buffers  32
    allocation and release  40
    connection-specific  38
Client-server environment  22
    summary of steps  23
Concepts of TSI  29
Configuration  29, 47
    binary files
        management  52
        client buffers  35
        error messages  67
        file
            example  60
            grammar (PDL)  69
            language  50
            on-line processing  52
            system  37, 76
TSI  47
    asyncIO parameter  54, 56, 65
    binary file  50
    dualAddress parameter  54
    interruptTrace parameter  54
    localPort parameter  58
    logLev parameter  54, 56, 149
    logName parameter  54, 149
    main parameters  54
    maxBuffers parameter  36, 41, 44, 54, 65, 77, 80
    maxBufSize parameter  33, 36, 37, 38, 39, 40, 41, 44, 55, 56, 65, 77, 152
    maxConns parameter  44, 55, 77, 85, 96
    maxErrors parameter  56, 79
maxInQ parameter 44, 56, 79
maxOutQ parameter 44, 56, 79
non-transport specific 56
server parameter 57, 60, 61, 63
serverName parameter 55, 62, 63
shared memory transport 59
example 62
shmKey parameter 59
shmMaxInQ parameter 59
shmMaxOutQ parameter 59
shmPeerName parameter 59, 62, 64
Simpact server 65
example 66
stackSize parameter 55
summary 47
TCP/IP socket transport 58
example 60
TCPKeepAlive parameter 58
TCPNoDelay parameter 58
timeout parameter 45, 57
traceLev parameter 55, 57, 150, 151
traceName parameter 55, 150
traceSize parameter 55, 150
transport parameter 57, 60, 62, 65
wellKnownPort parameter 58, 60, 61
tsicfg program 48
rules 51
Configuration parameters
connection definition
non-transport specific 56
shared memory transport 59
TCP/IP socket transport 58
main definition 54
Configuration preprocessor
see tsicfg program
Connection
definition 53
definition list 78
main definition 54
non-transport specific 56
shared memory transport 59
example 62
status 78
TCP/IP socket transport 58
example 60
Connection status 39, 40
Connection-specific buffers 38
Context free grammar 69
Customer support 16
D
Data
cautions, data loss 73
Data structures 76
connection definition 78
connection status 78
system configuration 76
Data transfer
see also Functions
DLI configuration parameters
BSC msgBlkSize 32, 33, 34
DLI header 38
DLITE embedded interface 18
Documents
reference 12
Download software 23
DualAddress TSI parameter 54
E
Electrical interface 21
Embedded ICP
overview 18
Error codes 125
command-specific 137
internal 125
see also Functions (return codes listed under each function)
tserro global variable 71, 125, 137
TSI_READ_ERR_OVERFLOW 41, 110
TSI_READ_ERR_QFULL 109
TSI_WRIT_ERR_INVALID_LENGTH 39, 41
TSI_WRIT_ERR_QFULL 120
Error handling
TSI logging 149
Error messages
tsicfg 67
Ethernet 21
Examples
calculation of buffer sizes 33
configuration file 60
F
Features
  product  21
  TSI  25
Files
  binary configuration
    management  52
  muxCfg  41
  on-line configuration  52
  see also Configuration, file
  server MuxCfg configuration  65
    example  66
Freeway
  client-server environment  22
  configuration  29
    overview  18
Functions
  categories  72
    dIO
      DLI_POLL_GET_SESS_STATUS
        option  38, 40
        syntax synopsis  74
tBufAlloc  40, 74, 80
        return codes  81, 137
tBufFree  41, 75, 82
        return codes  82, 137
tConnect  74, 84
        return codes  85, 137
tDisconnect  75, 88
        return codes  89, 137
    tlnit  74, 91
        return codes  92, 138
tListen  74, 95
        return codes  96, 139
tPoll  74, 100
        return codes  103, 139
    TSI_POLL_GET_CFG_LIST option  78
    TSI_POLL_GET_CONN_STATUS
        option  39, 40, 78
    TSI_POLL_GET_SESS_STATUS option  84
    TSI_POLL_GET_SYS_CFG option  37, 76, 80
tPost  74, 107
        return codes  107, 140
tRead  74, 109
  return codes  110, 140
tSyncSelect  75, 114
        return codes  115, 140
tTerm  75, 117
        return codes  118, 140
tWrite  74, 120
        return codes  121, 141
G
Grammar
  context free  69
  PDL  69
H
Headers
  DLI  38
  example  38
  TSI  37, 41
History of revisions  15
I
ICP buffers  32
Internet addresses  23
InterruptTrace TSI parameter  54
I/O
  blocking vs non-blocking  30
  completion handler (IOCH)  31, 143
  poll  145
  signal processing  45
  UNIX environment  143
  VxWorks environment  145
L
LAN interface processor  18
Library
  TSI reference  50
LocalPort TSI parameter  58
Logging services  149
LogLev TSI parameter  54, 56, 149
LogName TSI parameter  54, 149
M
MaxBuffers TSI parameter  36, 41, 44, 54, 65, 77, 80
MaxBufSize TSI parameter  33, 36, 37, 38, 39,
Freeway Transport Subsystem Interface Reference Guide

40, 41, 44, 55, 56, 65, 77, 152
MaxConns TSI parameter 44, 55, 77, 85, 96
MaxErrors TSI parameter 56, 79
MaxInQ TSI parameter 44, 56, 79
MaxOutQ TSI parameter 44, 56, 79
msgBlkSize BSC parameter 32, 33, 34
MuxCfg file 41
MuxCfg server configuration file 65
example 66
Files
server MuxCfg configuration 33, 40, 41

N
Negotiation of buffer size 39
Non-blocking I/O 30
I/O completion handler 31
signal processing 45
UNIX 144
VxWorks 146

O
On-line configuration file processing 52
Operating system
ProtoGate's real-time 18
Overview
embedded ICP 18
Freeway server 18
product 17
TSI 25

P
Poll I/O 145
UNIX 145
Product
features 21
overview 17
support 16

R
Reference documents 12
Resource requirements 44
Revision history 15
rlogin 21

S
Server buffers 33
Server processor 18
Server TSI parameter 57, 60, 61, 63
ServerName TSI parameter 55, 62, 63
Server-resident application 22, 43, 57, 59, 146
Session status 84
ShmKey TSI parameter 59
ShmMaxInQ TSI parameter 59
ShmMaxOutQ TSI parameter 59
ShmPeerName TSI parameter 59, 62, 64
SIGALRM 144
SIGIO 144
Signal processing 45
SNMP 21
Software
download 23
Solaris 144
StackSize TSI parameter 55
Status
connection 39, 40, 78
session 84
Support, product 16
Synchronous I/O
see Blocking I/O
System
configuration 37, 76
resource requirements 44

T
tBufAlloc (see also Functions) 80
tBufFree (see also Functions) 82
tConnect (see also Functions) 84
TCP/IP 21
TCPKeepAlive TSI parameter 58
TCPNoDelay TSI parameter 58
tDisconnect (see also Functions) 88
Technical support 16
telnet 21
Timeout TSI parameter 45, 57
tInit (see also Functions) 91
tListen (see also Functions) 95
tPoll (see also Functions) 100
tPost (see also Functions) 107
TraceLev TSI parameter 55, 57, 150, 151
TraceName TSI parameter 55, 150
TraceSize TSI parameter 55, 150
Tracing services 149, 150
  binary format 158
  example 154
  file layout 152
  Freeway server 159
Transport TSI parameter 57, 60, 62, 65
tRead (see also Functions) 109
Troubleshooting 149
terrno global variable 71, 125, 137
TSI
  concepts 29
  configuration 47
    see also Configuration, TSI
data structures 76
error codes 125
error handling 71
features 25
functions 71, 73
  see also Functions
  syntax synopsis 74
overview 25
reference library 50
TSI buffer pool definition 36
tsicfg program 47, 48
  error messages 67
tSyncSelect (see also Functions) 114
tTerm (see also Functions) 117
tWrite (see also Functions) 120

U
UNIX environment 143

V
VMS environment 147
VxWorks 18
VxWorks environment 145

W
WAN interface processor 18
WellKnownPort TSI parameter 58, 60, 61
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