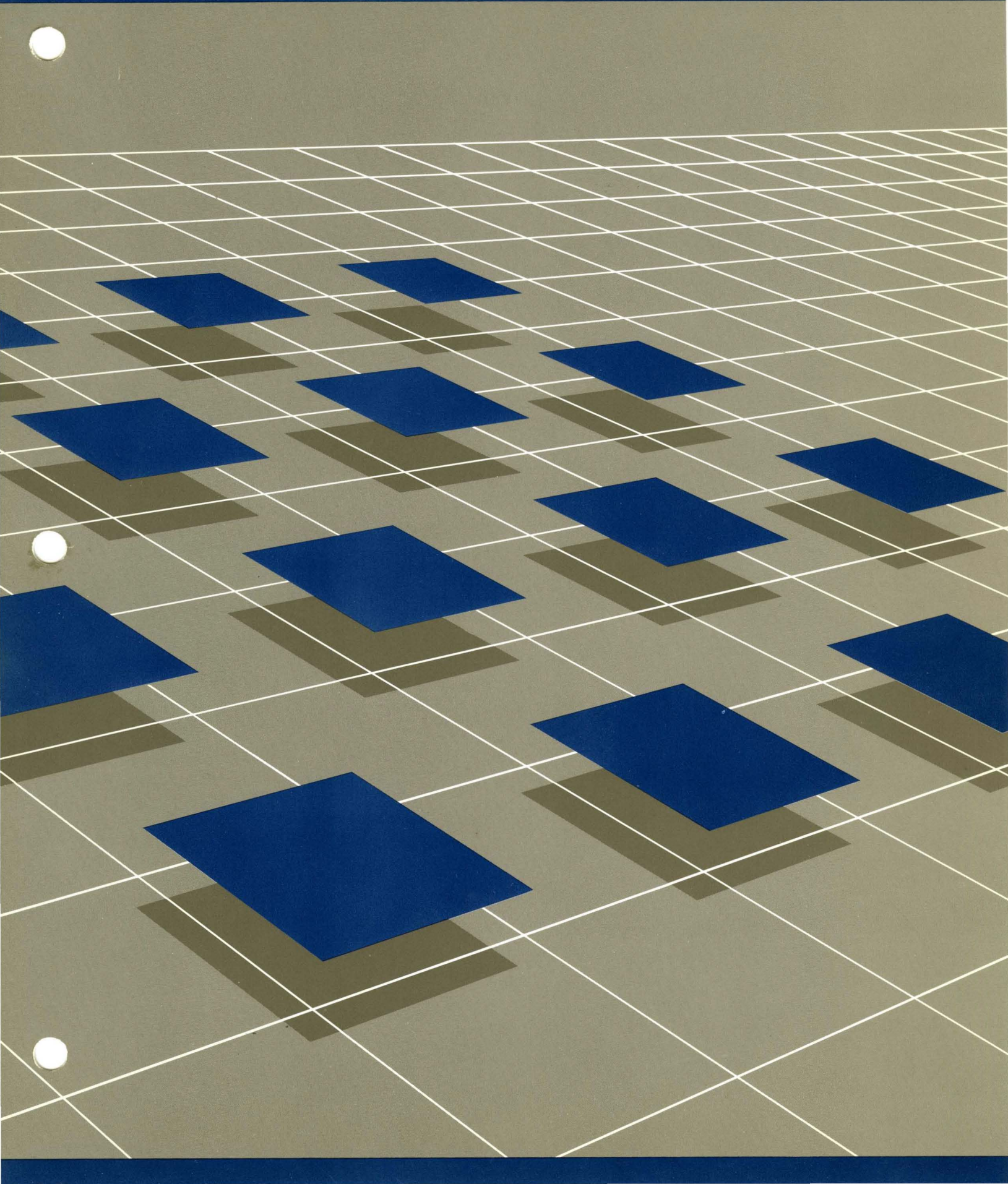
Bridge
Communications
Inc.

Ethernet System Product Line
Software Technical Reference Manual
Volume Two -- Protocols



BRIDGE COMMUNICATIONS, INC.

ETHERNET SYSTEM PRODUCT LINE

SOFTWARE TECHNICAL REFERENCE MANUAL

VOLUME TWO -- PROTOCOLS

09-0017-00

July, 1983

Bridge Communications, Inc., reserves the right to revise this publication, and to make changes in content from time to time without obligation on the part of Bridge Communications to provide notification of such revision or change.

Comments on this publication or its use are invited and should be directed to:

Bridge Communications, Inc.
Attn: Technical Publications
10440 Bubb Road
Cupertino, CA 95014

PUBLICATION CHANGE RECORD

This page records all revisions to this publication, as well as any Publication Change Notices (PCNs) posted against each revision. The first entry posted is always the publication's initial release. Revisions and PCNs subsequently posted are numbered sequentially and dated, and include a brief description of the changes made. The part numbers assigned to revisions and PCNs use the following format:

aa-bbbb-cc-dd

where "aa-bbbb" identifies the publication, "cc" identifies the revision, and "dd" identifies the PCN.

PCN Number	Date	Description	Affected Pages
09-0017-00	07/83	First Release	All

PREFACE

This manual provides the Bridge Communications customer with the information necessary to add software to a Bridge ESPL product.

The manual was prepared based on the following assumptions of reader knowledge:

1. The reader should be familiar with the information provided in the Bridge Communications Ethernet System Product Line Overview and CS/1 User's Guide.

2. The reader should be familiar with the Ethernet Specification, Version 1.0 (see reference [4]).

3. The reader should be familiar with the Xerox Network System high-level protocols (see references [5], [6] and [7]).

4. The reader should have some familiarity with the UNIX* operating system (see reference [8]).

5. The reader should be familiar with the "C" language (see reference 9), or other high-level structured languages.

6. The reader should be familiar with the material presented in Volume One of this manual (particularly Section 5.0, the Kernel Interface).

The ESPL Software Reference Manual is divided into three volumes. The information in Volume Two is grouped in five major sections whose contents are as follows:

Section 1.0 - Introduction: Provides an overview of the Bridge Communications Ethernet System Product Line (ESPL), and describes the purpose and scope of this manual.

Section 2.0 - Internetwork Datagram Service: Describes the Internetwork Datagram Service, which provides a means of sending packets compatible with the Xerox Network System Internet Datagram Protocol. The interfaces with other protocol layers are defined.

Section 3.0 - Level Two Support Protocols: Describes the Error Protocol and the Echo Protocol as implemented in the ESPL.

*UNIX is a Trademark of Bell Laboratories.

Section 4.0 - Packet Stream and Byte Stream Services: Describes the Packet and Byte Stream Services, which provide reliable packet transmission to and from a specified destination, and their interfaces to other protocol layers.

Section 5.0 - Virtual Terminal Connection Service: Describes the session-level protocol processing provided by the Connection Service and the interfaces between the Connection Service and other protocol layers.

Volume One of this manual describes the ESPL software architecture, software development environment, MCPU monitor, operating system and floppy disk I/O interface. Volume Three describes the ESPL drivers and firmware.

REFERENCES

The following publications describe the Bridge Communications Ethernet System Product Line (ESPL):

[1] Ethernet System Product Line Overview, Bridge Communications, Inc.

[2] ESPL Communications Server/1 User's Guide, Bridge Communications, Inc.

[3] ESPL Software Reference Manual, Volumes One and Three, Bridge Communications, Inc.

The following publications describe Ethernet and the Xerox Network System products:

[4] The Ethernet, A Local Area Network; Data Link Layer and Physical Layer Specifications, Version 1.0 (Digital Equipment Corporation, Intel Corporation, and Xerox Corporation, 1980)

[5] Internet Transport Protocols, XSIS 028112 (Xerox Corporation, 1981)

[6] Courier: The Remote Procedure Call Protocol, XSIS 038112 (Xerox Corporation, 1981)

[7] D. Oppen, Y. Dalal, The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment (Xerox Corporation, 1981)

The following publications describe other related specifications:

[8] UNIX Programmer's Manual, Seventh Edition, Virtual VAX-11 Version, (University of California, Berkeley, 1981)

[9] B. Kernighan, D. Ritchie, The C Programming Language (Prentice Hall, Inc., 1978)

[10] MC68000 Microprocessor User's Manual, Second Edition MC68000UM(AD3) (Motorola Corporation, 1982)

[11] MC68000 Educational Computer Board User's Manual, Second Edition

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

The ESPL Software Technical Reference Manual provides the OEM-level Bridge Communications customer with the information necessary to add software to an Ethernet System Product Line product. In addition, it provides information about the existing ESPL software modules for the sophisticated user (e.g., the Network Manager).

The manual makes no attempt to present tutorial-level material aimed at the end user; please refer to the appropriate User's Guide for tutorial material.

The Software Technical Reference Manual is divided into three volumes. Volume One describes the ESPL overall software architecture, the software development environment, the kernel and various support software. Volume Two (this manual) describes the high-level, packet-processing protocols used in the ESPL. Volume Three describes the ESPL drivers and firmware.

2.0 INTERNETWORK DATAGRAM SERVICE

This section describes the Internetwork Datagram Service, including an overview of the implementation of the Internetwork Datagram Protocol (IDP) and its client interface. This service is used to send datagrams to an addressable destination which may be on another network.

2.1 Overview

The IDP software resides on the MCPU processor. It exists as a single process, and is created by the parent process "init" at system initialization time. Communication between IDP and its clients (level two protocol processes, usually SPP) is accomplished primarily via IPC messages. Communication between IDP and level zero service providers (e.g., the Data Link Service) uses a procedure call interface for transmission and IPC messages for reception. Communication between IDP and the Level 2 service protocols (Error and Echo, which are implemented as a part of Network Management and are described in Section 8.0) uses primarily a procedure call interface.

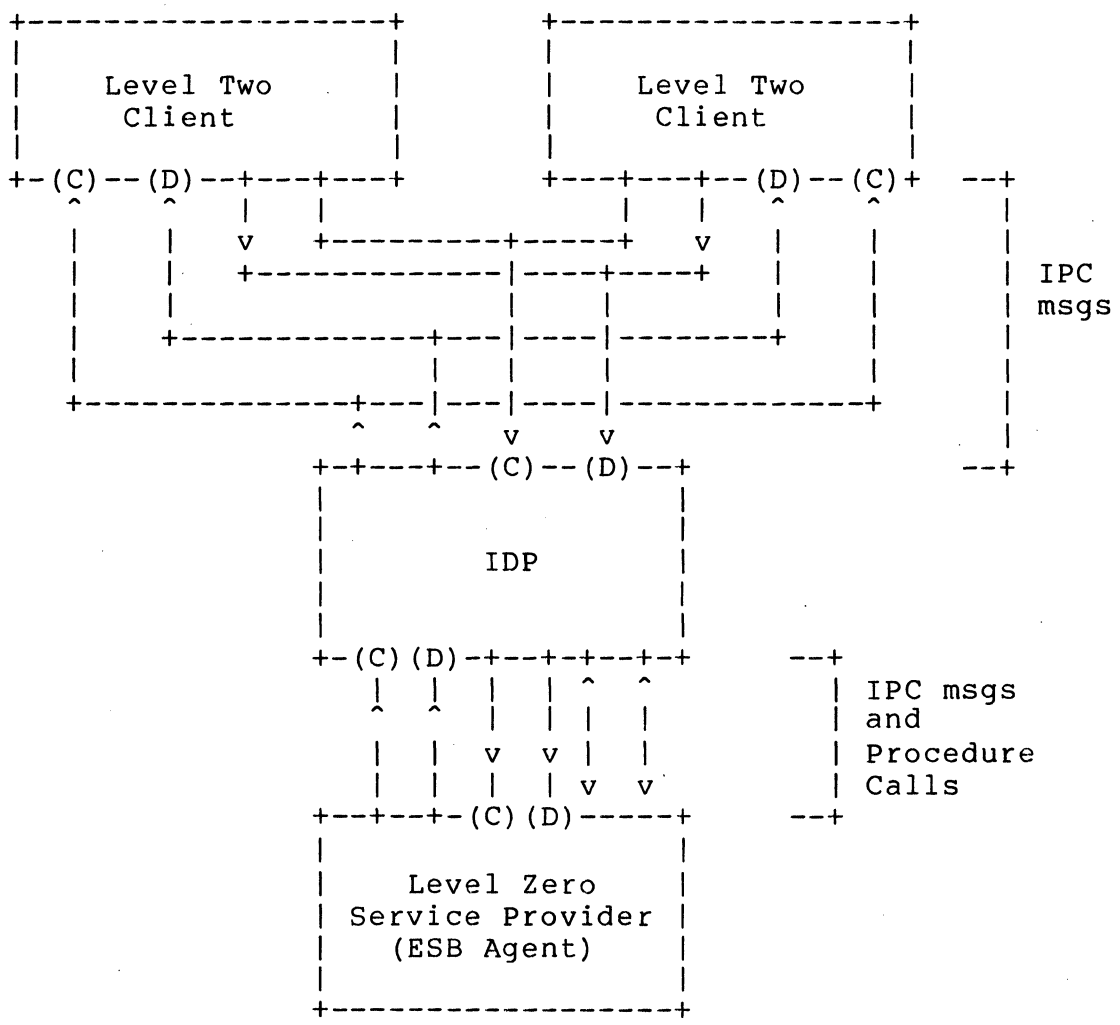
2.1.1 Initialization

IDP is created at system initialization. The parent process "init" creates a well-known control mailbox and registers the mailbox name of "IDP". The parent then makes IDP runnable, and transfers control to the entrypoint under normal kernel scheduling.

For initialization of the interface to the ESB agent, IDP is responsible for creating a mailbox to be used for receiving incoming packets, as well as for issuing a procedure call to the ESB agent initialization routine "eainit" as described in Section 6.3.1. One procedure call is issued for each ESB agent residing in the local node.

2.1.2 IDP Interfaces

Communication between a client and IDP is via the kernel's IPC message facility. Initial establishment requests originate from the client using the well-known mailbox created and registered on IDP's behalf. Therefore, no explicit action is required on the part of IDP to make this interface available. IDP's interaction with other processes is illustrated in Figure 2-1.



Notes:

(C) = Control mailbox
(D) = Data mailbox

Figure 2-1 IDP Interfaces

IDP initially blocks waiting for messages from either a client or the ESB agent. Then IDP services level zero and level two receive queues, processing each packet completely.

Outbound data is treated differently. A procedure call is used to transfer a buffer to the ESB agent. If a packet is rejected, there is no attempt to queue, retry or otherwise perform error recovery. IDP is not a reliable protocol; reliability is the responsibility of IDP's clients and (to some extent) of the Data Link Service.

2.1.3 Addressing

The ability of an IDP client to address or identify a destination device on another network is provided by a network address. The address consists of the following:

- 32-bit network number,
- 48-bit level zero address, and
- 16-bit socket number.

With the addition of a packet type field, the network address constitutes the level one or IDP address. The packet type is supplied by the IDP client at the service interface, but it is not used in determining the recipient of the packet at the target system.

In order to route a packet to the destination level two socket, IDP provides the Data Link Service with a 48-bit level zero address consisting of a 32-bit host number and a 16-bit type field. This address identifies an immediate destination to the level zero service provider.

An IDP client can be uniquely identified by its host number and socket number. If internetwork routing is not required, IDP is able to provide communication based on this information. The network number is only required for internetwork routing; in a single network environment, it is not required for identification of IDP clients.

IDP supports the broadcast and multicast ability specified in reference [5]. This ability consists of delivery of broadcast packets inbound and internetwork forwarding of directed broadcast packets (by definition, directed broadcast is to a distant network).

Please see reference [5] for more detailed information on addressing.

2.1.4 Socket Establishment

IDP functions as a slave or passive listener in the establishment of communication paths to level two users. A client desiring a path to/from IDP sends a socket-establishment message to IDP's control mailbox. Before asking IDP to open a socket, the client must have created a mailbox to be used for reception of data and obtained a socket number by invoking the socket management procedure call. When a socket has been established and opened, IDP returns a message to the client including a pointer to the data mailbox to be used by that client.

2.1.5 Data Transfer To/From Client

The function of IDP is to enable clients to exchange data. IDP has no knowledge of a data stream or connection; each packet is unrelated to every other.

The mailbox in which the client receives data is defined by the client, therefore queue depth is its responsibility. IDP makes no assumption about the depth of this queue. No internal queuing takes place within IDP; if a send queue is full, IDP records error statistics and discards the packet.

IDP maintains one mailbox for reception of client data messages. Although the socket-establishment/socket-open facility allows the definition of multiple outbound data mailboxes, IDP uses only one data mailbox for all open sockets. Once a packet has been dequeued, it is processed completely.

2.1.6 Incoming Packet Reception

On receipt of a packet from the ESB agent, the IDP process examines the level one header information and attempts to pass on the packet to the intended level two protocol entity, or to forward the packet to another host. The level one destination address must be either for the local host or for another host on a network different than that on which the packet reached this station. Packets reaching the IDP level must have a level zero destination address matching this host.

Address decapsulation is accomplished through the level zero/level one interface. No explicit action is required on the part of IDP. The packet presented to IDP is an internetwork packet. IDP determines if the packet is intended for a host on one of IDP's directly attached networks. If it is, a best-effort attempt is made to transfer the packet to the appropriate socket. If the packet is intended for a host on a remote (i.e., not directly attached) network, a best-effort attempt is made to route the packet to the gateway attached to the destination network.

If the packet is destined for the local host, a demultiplexing operation is performed on socket number. The socket must have previously been established by a level two client. The indicated socket-to-mailbox mapping is performed, and IDP transfers the packet to the indicated mailbox. If the destination socket is not identifiable, IDP delivers the packet to the Error protocol.

2.1.7 Packet Transmission

On reception of an outbound client packet, IDP examines the level one destination address fields. At this point a decision is made to route the packet to a client in the local host, or forward it to another host. This decision is based on the matching the level one destination host address to the local host address. A destination of broadcast is acceptable.

If the two values do not match, the packet is passed to the transmission/encapsulation logic.

IDP supports directed multicast and broadcast addressing at level one, but does not support global network addressing (e.g., a network address of "all").

Very few actions are performed on an outbound packet. The host address and socket/protocol type fields have previously been filled in by the client.

2.2 Client Interface to IDP

This section describes the interfaces between IDP and its clients, which is accomplished via one procedure call and several IPC messages.

Note that communication between IDP and the ESB Agent is also accomplished partly via IPC messages and partly via procedure calls; both are described in Volume Three, Section 2.0, the Data Link Service.

2.2.1 Socket Management Procedure Call

This procedure call is issued by the client to obtain a socket number. A socket number must be obtained before a socket can be opened or used for transmission/reception. No parameters are passed with the call.

"C" Declaration:

```
SOCKADD   idsktget()
```

Input Parameters: None

Output Parameter: Next available socket number.

2.2.2 The Socket Establishment Message

This message is sent from the client to IDP to request that IDP open a socket previously obtained via the socket management procedure call.

"C" Declaration:

```
typedef struct idl2_skest
{
    MSG     idsk_m;
    MBID    idsk_dmbox;
    SOCKADD idsk_sock;
}  IDL2_SKEST;
```

Message Parameters:

idsk_m Message header (message type MIDEST).

idsk_dmbox
Pointer to requestor's data mailbox.

idsk_sock
Socket number.

2.2.3 The Socket Close Message

This message is sent by the client to IDP to request that IDP close the specified socket.

"C" Declaration:

```
typedef struct idl2_sclose
{
   MSG      idsc_m;
   SOCKADD  idsc_ssk;
} IDL2_SCLOSE;
```

Message Parameters:

idsc_m Message header (message type MIDCLS).

idsc_ssk Source socket number.

2.2.4 The Socket Open Message

This message is sent by IDP to the client to advise the client that a requested socket has been established and opened.

"C" Declaration:

```
typedef struct idl2_socopen
{
   MSG     idso_m;
   SOCKADD idso_sk;
   MBID    idso_mbid;
}  IDL2_SOCOPEN ;
```

Message Parameters:

idso_m Message header (message type MIDOPEN).

idso_sk Socket number of socket being opened.

idso_mbid
Mailbox to which to send data.

2.2.5 The Receive Data Message

This message is sent from IDP to the client to advise that IDP has a packet for the client.

"C" Declaration:

```
typedef struct idl2_rdatamsg
{
   MSG              idrd_m;
   Ll_ADDR          idrd_dadd;
   Ll_ADDR          idrd_sadd;
   unsigned short   idrd_ptype;
} IDL2_RDATAMSG;
```

Message Parameters:

idrd_m Message header (message type MIDRDATA).

idrd_dadd
Destination network address (this network).

idrd_sadd
Source address of data.

idrd_ptype
Protocol type.

2.2.6 The Send Data Message

This message is sent by the client to request that IDP transmit a packet.

"C" Declaration:

```
typedef struct idl2_sdatamsg
{
   MSG              idsd_m;
   L1_ADDR          idsd_sadd;
   L1_ADDR          idsd_dadd;
   unsigned short   idsd_ptype;
} IDL2_SDATAMSG;
```

Message Parameters:

idsd_m — Message header (message type MIDSDATA).

idsd_sadd — Source address (this network address).

idsd_dadd — Destination address of data.

idsd_ptype — Protocol type.

2.2.7 The IDP Error Message

This message is sent from IDP to the client when an error is detected.

"C" Declaration:

```
typedef struct idl2_errmsg
{
   MSG       idem_m;
   SOCKADD   idem_sk;
   short     idem_mtype;
   short     idem_errcd;
} IDL2_ERRMSG;
```

Message Parameters:

idem_m Message header (message type MIDERR).

idem_sk Socket from which error originated.

idem_mtype
Message type of message in error.

idem_errcd
Error code.

Error Codes:

IDEM_SKTFULL
Socket table full (1).

IDEM_SKOPN
Attempt to open an already open socket (2)

IDEM_CLNOPN
Attempt to close socket that is not open (3).

IDEM_LEN Send data request with length > 546 (4).

IDEM_UNDEF
Undefined other error (5).

IDEM_UEST Undefined socket establishment problem (6).

2.3 Peer Protocol

This version of IDP supports all the required features described in reference [5].

Two Level 2 service protocols are described in reference [5] (the Error Protocol and the Echo Protocol). While IDP does make use of the services of Error in the Bridge implementation, both Level 2 service protocols exist as separate processes and are documented separately in Section 3.Ø of this volume.

2.4 IDP Sysgen Parameters

This section describes the Sysgen parameters which apply to IDP. Since the default values of the parameters and the recommended value ranges vary depending on the user's specific configuration, these values and ranges are not included in this manual; refer to the appropriate Sysgen menu for guidelines.

2.4.1 Mailbox Depth Level Ø

This parameter specifies the depth of the mailbox used for messages from the Ethernet Agent to IDP.

2.4.2 Mailbox Depth Level 2

This parameter specifies the depth of the mailbox used for messages from IDP to the Ethernet Agent.

2.4.3 Maximum Number of Sockets

This parameter specifies the maximum number of sockets IDP will allow; the value varies with the number of sessions permitted per port and the number of ports in the system.

2.4.4 Number of Attached Networks

This parameter specifies the number of networks attached to this host (e.g., the number of level zero agents with which IDP needs to communicate).

2.4.5 Level Ø Service Access Point Table

The Level Ø Service Access Point Table (lØsaptab) contains an entry for every attached network specified by the Number of Attached Networks parameter (Section 2.4.5), including information on how to route packets to it. The current network is always the first entry in the table.

2.4.6 Level Ø Remote Network Directory Table

The Level Ø Remote Network Directory Table (lØrmtdir) contains an entry for every known remote (i.e., not directly attached) network. Each entry contains the network ID, the host address of the gateway, and the number of hops needed to reAch the destination. The table is ordered by network ID.

3.0 LEVEL TWO SUPPORT PROTOCOLS

The level two support protocols available in the ESPL products include the Error Protocol and the Echo Protocol.

3.1 The Error Protocol

This section briefly describes the implementation of the XNS Error Protocol in the Bridge ESPL.

3.1.1 Overview

The Error Protocol is provided as an XNS level two protocol. The Error protocol defined in reference [5] is used by level two processes to inform a correspondent of a perceived error condition. Error packets are returned to their originating network address, if it can be determined.

Level two processes that wish to use the Error protocol must include the ability to handle the Error packet type. The current implementation of SPP can handle incoming Error packets. The only use of the Error process is to generate or receive information about errors detected by IDP. The error conditions usually occur when it is not possible to deliver to or even determine the identity of the destination level two process (e.g., the destination socket does not exist).

The Error process is a single, independent process which is started up by the parent process "init" at the time of system initialization. The default mailbox established by the parent process is called "ERROR". Because all communication between Error and its clients is via IPC messages, it is possible to compile an ESPL system without the Error process by omitting the Error process entry from the sysinit table (refer to Volume One, Section 5.2.1 for a description of the table).

3.1.2 Error Protocol Packets

The level two portion of an Error protocol packet contains four octets of header information prepended to the user data. User data is defined as the entire packet in which an error has been detected, including level one headers. Error receives the entire packet in the form of a buffer descriptor, prepends the two Level Two header fields (error number and parameter), and passes the packet back to IDP via the IDP Send Data message (refer to Section 2.2.6).

The current version of the Error protocol makes no use of error protocol packets addressed to the well-known "error protocol" socket. It receives the packet on that socket from level one, and releases the associated memory resources.

3.1.3 Client Interface

All communication is accomplished via a single IPC message. The client uses the erequest message structure to request generation of an Error protocol packet.

"C" Declaration:

```
typedef struct erequest
{
    MSG     errmsgx;
    short   errcodex;
    short   errparm;

}  EREQUEST;
```

Message Parameters:

errmsgx Message header (message type MEREQUEST).

errcodex Error type code.

errparm Additional error information.

Error Codes:

ERcodunspec
: Unspecified error detected at destination (ØØ).

ERcodcksum
: Checksum error detected at destination (Ø1).

ERcodnosock
: Destination socket does not exist (Ø2).

ERcodblckd
: Resource error at destination (Ø3).

ERcodiunspec
: Unspecified error occurred before reaching destination (Ø1ØØØ).

ERcodicksum
: Checksum error occurred before reaching destination (Ø1ØØ1).

ERcodiunrch
: Destination host unreachable from here (Ø1ØØ2).

ERcodiaged
: Packet passed through 15 Internet Routers without reaching destination (Ø1ØØ3).

ERcodilen
: Invalid length, packet too long to be forwarded through an intermediate network. The errparm field contains the maximum allowable length (Ø1ØØ4).

3.2 Echo Protocol

The Echo Protocol is a level two service protocol as defined in reference [5]. Its main function is to verify the existence, accessibility and operation of a remote host. Echo can also be used to test the operation of level one and level zero servers used to connect two hosts.

3.2.1 Overview

The Bridge ESPL implementation consists of a distinct, independent process. Echo is created by the parent process "init" at the time of system initialization time, and has a default mailbox called "ECHO". An ESPL system can be compiled without the Echo process by omitting the Echo entry from the sysinit table.

Echo performs the following functions:

1. On initialization, Echo opens a well-known socket.

2. Echo communicates with its client as follows:

   - Echo receives requests for service on the well-known mailbox registered as "ECHO".

   - Echo can only process one user request at a time; a reply must be received or a timeout must expire before Echo can accept a subsequent request.

   - Echo informs the client when the expected reply is received or the timeout has expired.

3. After receiving a request, Echo does the following:

   - Sends an Echo protocol packet to the specified destination network address; the socket will be the Echo protocol's well-known socket.

   - Keeps track of the outstanding request.

4. Echo processes and responds to protocol packets received at the well-known socket.

   - If a packet is identified as an echo request, Echo issues a reply.

   - If a packet is identified as an echo reply, Echo informs the client that a reply was received.

   - If Echo can not locate the appropriate client, it discards the reply packet.

Echo processes incoming packets as follows: IDP messages with a message type of MIDRDATA are checked for the correct protocol type and Echo operation (either Request or Reply). Request messages are from a remote peer, and are simply turned into reply messages. This entails swapping the source and destination network addresses, and changing the operation from Request to Reply.

A reply message is associated with an outstanding request; the returned message is sent to the client originating the request, and the data structure containing the message is cleared.

Outgoing packets are sent to IDP using the MIDSDATA message type.

3.2.2 Client Interface

Communication between Echo and its clients is accomplished by the use of two IPC messages. The following subsections describe these messages.

3.2.2.1 The Ecrequest Message

The ecrequest message is used by a client process to instruct Echo to send an Echo packet.

"C" Declaration:

```
typedef struct ecrequest
{
   MSG         ecq_msg;
   NETADD      ecq_dnet;
   HOSTADD     ecq_dhost;
   short       ecq_id;
   MBID        ecq_mbox;
}  ECREQUEST;
```

Message Parameters:

ecq_msg Message header (message type MECREQUEST).

ecq_dnet Destination (requestor's) network address.

ecq_dhost Destination host address.

ecq_id Optional request identifier.

ecq_mbox Mailbox to which to send reply.

3.2.2.2 The Ecreply Message

The ecreply message is sent by Echo to the sender of an ecrequest message.

```
typedef struct ecreply
{
   MSG         ecr_msg;
   NETADD      ecr_snet;
   HOSTADD     ecr_shost;
   short       ecr_id;
   short       ecr_code;
}  ECREPLY;
```

Message Parameters:

ecr_msg Message header (message type MECREPLY).

ecr_snet Source (requestor's) network address.

ecr_shost Source host address.

ecr_id Optional request identifier.

ecr_code Error return code.

Error Codes:

ECR_RECD Echo received, no error detected (Ø).

ECR_TIME Timeout elapsed (-1).

ECR_SERR Undefined system error (-2).

4.0 PACKET STREAM AND BYTE STREAM SERVICES

This section describes the Packet Stream and Byte Stream Services provided by the Sequenced Packet Protocol (SPP). These services are used to send reliable streams of information (as packets or as bytes) to a destination process anywhere within the internetwork environment.

4.1 Overview

The SPP software resides on the MCPU. There is one Parent SPP process, plus one SPP process per connection, and a one-to-one mapping between connections and sockets. All SPP processes except the parent share the same code and use kernel stack-sharing.

The Parent SPP process is created at system initialization time by the "init" process, and has a default mailbox name called "PS". When a level three client process wants to establish an SPP connection, the client sends a message to the Parent SPP's default mailbox requesting that Parent SPP spawn a new SPP process for the connection. All requests from the client to SPP are sent via the kernel IPC facility as messages to this new SPP process, and all replies from SPP are sent back to the client as messages. When the client wants to delete the connection, it sends a delete message either to the Parent or to its SPP process, resulting in the deletion of the SPP process.

All communication between SPP and the IDP process is also done via IPC messages.

Of the three modes described in reference [5], the SPP process as implemented in the ESPL supports two: packet stream mode and byte stream mode. The third mode (reliable packet mode) is identical to packet stream mode except that the client process receives packets from the network as they arrive, possibly out of order but without duplicates. Reliable packet mode is not supported at this time because the ESPL includes no higher level protocols that use this interface.

The following subsections describe the major functions of the Parent SPP and the individual SPP processes.

4.1.1 SPP Mailbox Usage

The Parent SPP process has two mailboxes: a data mailbox for receiving packets from IDP, and a control mailbox for receiving all other messages. The control mailbox (called "PS") is the default mailbox established at initialization time.

Each SPP process has three mailboxes: two data mailboxes and one control mailbox. The control mailbox is the default mailbox obtained at the time Parent SPP created the SPP process via the

procreate call). Control messages from all sources (the client, the Parent SPP, IDP, and timeouts) are sent to SPP's default mailbox. One data mailbox is used for receiving data from the client, and one for receiving data from IDP.

All control messages from SPP to the client are sent to the control mailbox specified by the client (this would normally be the client's default mailbox). All data messages to the client are sent to the specified data mailbox. The client's data mailbox should only be used for data messages from a single SPP process.

4.1.2 Connection Creation

The client process creates a new SPP connection by sending either a "create packet stream connection" or a "create byte stream connection" message to the well-known mailbox of the Parent SPP process. If the client is a service listener agent, it sends an "accept packet stream connection" or "accept byte stream connection" message instead. Both create stream messages contain the following parameters:

Source address
Destination address
Control mailbox
Data mailbox
Mode (LISTEN/INITIATE)

The Parent SPP creates a new SPP process for each stream, and passes it the create stream parameters. This sequence of actions is illustrated in Figure 4-1.

If the specified source address contains a non-zero socket number, the new SPP process will use this socket number for the connection. If zero is specified, the Parent will obtain a unique socket number for the connection from socket management, and pass this number along to the new SPP process.

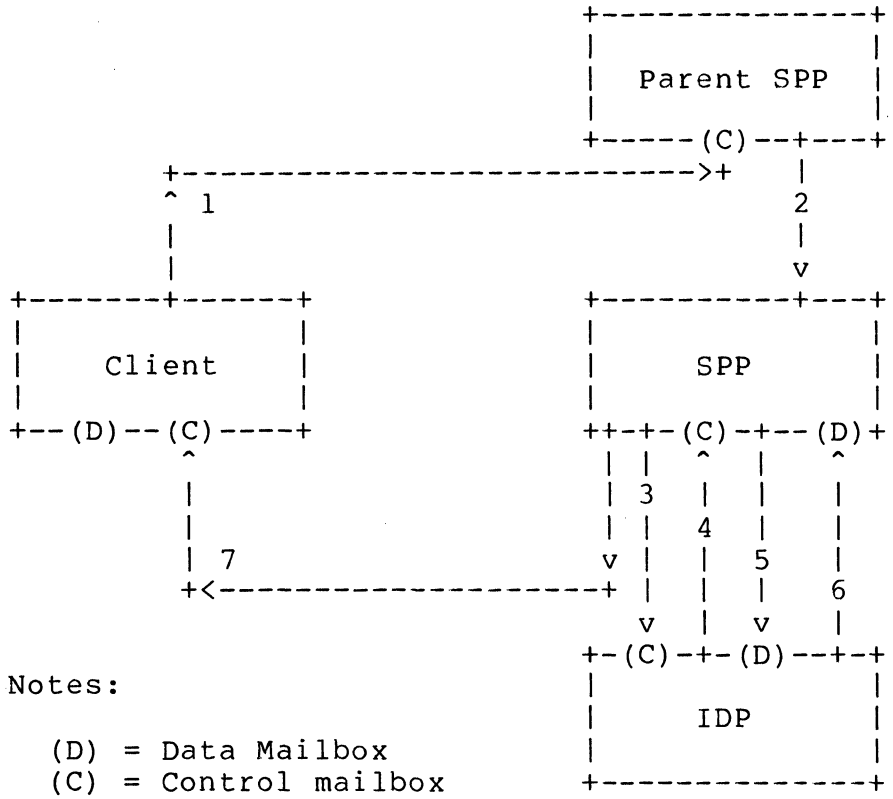
The mailboxes specified for control and data must be different. SPP uses the specified data mailbox to send data and attention messages to the client process. It should have a small maximum depth, so that SPP can react quickly whenever the client becomes congested. All other messages from SPP are sent to the specified control mailbox, which should not be flow-controlled.

The mode specifies whether to listen for incoming traffic (LISTEN), or initiate connection establishment (INITIATE).

The additional byte stream parameters (described fully in Section 4.1.7) are as follows:

Combine flag (for sending and receiving data)
Send maximum length
Receive maximum length

On startup, the newly created SPP sends a socket establishment message to IDP, waits for the socket open reply message, and then either listens for incoming packets from IDP or initiates connection establishment by sending the other end of the connection a System Packet with the Send Acknowledgement flag set, to force a reply. No data may be transmitted on the connection until SPP has received an initial packet from the other end of the connection. At that time, SPP sends a stream created message back to the client's control mailbox. On error, the client receives a stream not created message.



1. Create stream message
2. Process creation
3. Socket establishment request
4. Socket open message
5. Send data with initial packet
6. Receive data with initial packet
7. Stream created message

Figure 4-1 Connection Creation

4.1.3 Connection Termination

The Sequenced Packet Protocol specifies no means for sending termination packets to the other end of a connection. It is the responsibility of the client process to terminate reliably, such as by using the three-way "end/end-reply" exchange described in reference [5], Section 7.5.

Therefore, the client cannot request that SPP do a graceful close. When the client sends a delete stream request to the SPP control mailbox, the SPP process immediately exits. Any outgoing data packets queued for transmission, and any incoming packets not yet processed by SPP will be discarded. Figure 4-2 illustrates connection termination.

Similarly, when SPP decides to abort itself, for example because a packet cannot be transmitted to the other side, it does not attempt to inform its client of any packets that are in transit. It just sends a stream aborted message to the client's control mailbox.

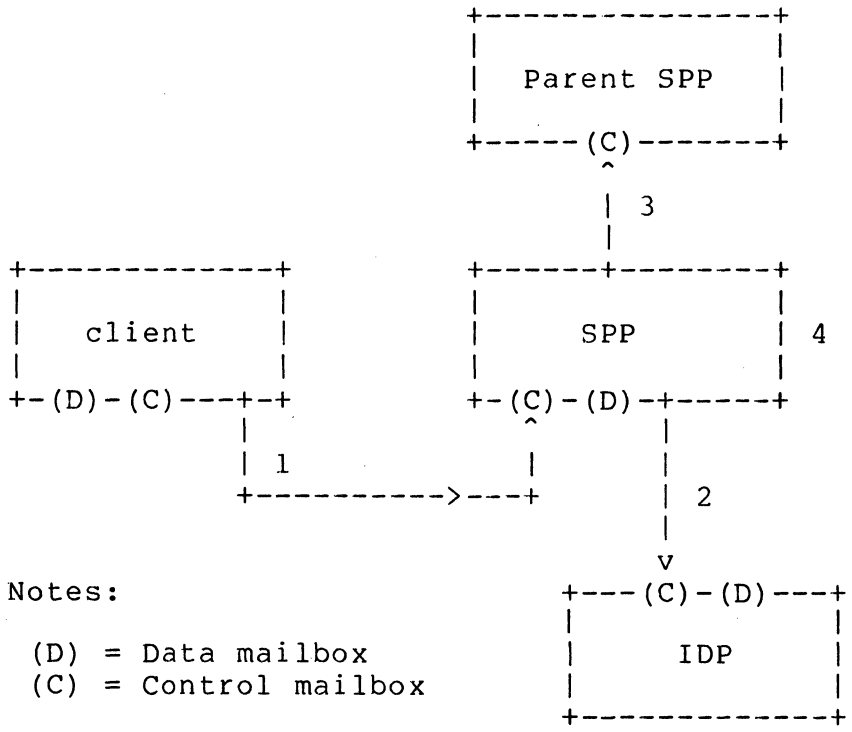


Figure 4-2 Connection Termination

Note that between the time the client sends a create stream request to the Parent process, and receives a stream created message from the new SPP process, it does not know its SPP's control mailbox address and therefore cannot send a delete stream request. To delete a connection in this state, the client must send a delete connection message to the Parent SPP process, which then forwards an abort message to the appropriate child process.

4.1.4 Data Transfer

The client passes outbound data to its SPP process in a data message. This message contains a buffer descriptor for the data, and transfers ownership of the buffer to SPP. There is no acknowledgement message from SPP informing the client that SPP is finished with the buffer. Therefore, to retain buffer ownership, the client must make a copy of the buffer by using the copybuf kernel primitive (which copies the buffer descriptor but not the actual buffer data) and pass the copy to SPP. Similarly, SPP sends outgoing data to IDP by transferring ownership of a buffer descriptor in a send data message. SPP retains a copy of all buffer descriptors except those for System Packets in order to do retransmissions.

IDP sends incoming data to SPP by passing a buffer descriptor in a receive data message. Similarly, SPP passes the data to its client in a data message. This message contains a buffer descriptor for the data, and transfers ownership of the buffer to the client. There are no explicit receive request messages between either the client and SPP, or SPP and IDP.

4.1.5 Attentions

Attentions between the client and its SPP process are out-of-band messages with mark. The client requests that an attention packet be sent by sending two attention messages to SPP, one to its data mailbox, to mark its place in the stream of data, and one to its control mailbox, to expedite its delivery. The attention message contains a data stream type, EOM flag, and one byte of data. SPP converts each pair of attention messages into a single data packet with the attention bit set. This data packet will be the next one to be sent to the other side, whether or not its sequence number is the next number to be sent (e.g., there may be queued data packets with lower sequence numbers than the attention packet which have not been sent because the send window is closed).

On receipt of a data packet with attention set, SPP sends an attention message to the client's control mailbox containing the fields in the attention packet. SPP also delivers the attention message in sequence to the client's data mailbox (i.e., attention messages are delivered twice, once in-band and once out-of-band).

4.1.6 Packet Stream Versus Byte Stream Interfaces

The messages sent between SPP and both packet-stream and byte-stream clients are the same, except for the initial create stream message from the client.

The packet stream client is aware of SPP packet boundaries. The data contained in each data message passed from the client to SPP is sent as a single packet to the other side of the connection, and the data in each data message sent from SPP to the client contains a single packet. The maximum amount of data in an SPP packet, and therefore in data messages, is 534 bytes.

The byte stream client does not see packet boundaries. The combine flag parameter of the create byte stream connection message tells the SPP process how to combine buffers from multiple data messages into single SPP packets, and how to combine multiple SPP packets into single data buffers.

There are two options on both outgoing and incoming data. For outgoing data from the client, passed to SPP in data messages, the choices are:

1. Do not combine multiple data buffers into single packets.

2. Combine data buffers until either a specified maximum number of bytes or an end-of-message signal has arrived, or a change in data stream type has occurred.

In all cases, SPP splits large amounts of data into 534-byte packets on behalf of the byte stream client. This is the only service that SPP performs if choice one is selected. If choice two is selected, SPP accumulates data until either an end-of-message signal or a new data stream type is received (indicated by a change in the send combine flag), or it has buffered a specified amount of data, whichever happens first. The specified maximum send length must be between 1 and 534 bytes.

For incoming packets from IDP, passed back to the client in data messages, the choices are:

1. Do not combine multiple packets into single data buffers.

2. Combine packets until either a specified maximum number of bytes or an end-of-message signal have arrived, or a change in data stream type has occurred.

If choice two is selected, SPP accumulates packets until either it sees an end-of-message signal or a new data stream type (indicated by a change in the receive combine flag), or it has buffered the specified amount of data, whichever happens first. The specified receive amount must be between 1 and 1024.

4.1.7 Service Listener Creation

The Parent SPP Process handles the level two protocol-specific details of the consumer/server hookup as described in reference [5], Section 7.4.2, for service listeners such as the Courier Server.

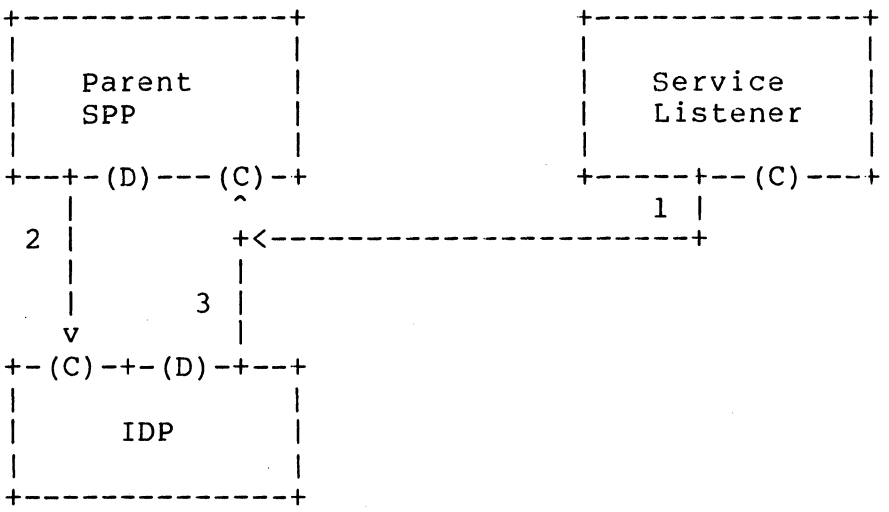
The service listener sends a start listening message to the Parent SPP process to request that it listen for initial packets on a well-known socket. On receipt of the start listening message, the Parent sends a socket establishment message to IDP for the specified socket, waits for a socket open message from IDP, and then starts listening for consumer packets.

On receipt of an initial SPP packet for a socket number on which it is listening, the Parent queues the packet internally and sends a connection request received message to the service listener. Note that the Parent SPP process must be able to recognize and reject duplicate initial packets.

The service listener typically creates a new service-supplying agent process to handle the connection; the new process sends a accept packet stream connection or accept byte stream connection message to the Parent containing the well-known mailbox address. The Parent SPP process handles the accept stream request by obtaining a unique socket number for the connection.

The service listener sends a reject connection request message to the Parent SPP process if it cannot accept the connection request.

Figure 4-3 and 4-4 illustrate the registration of a service listener and the creation of a service listener agent, respectively.

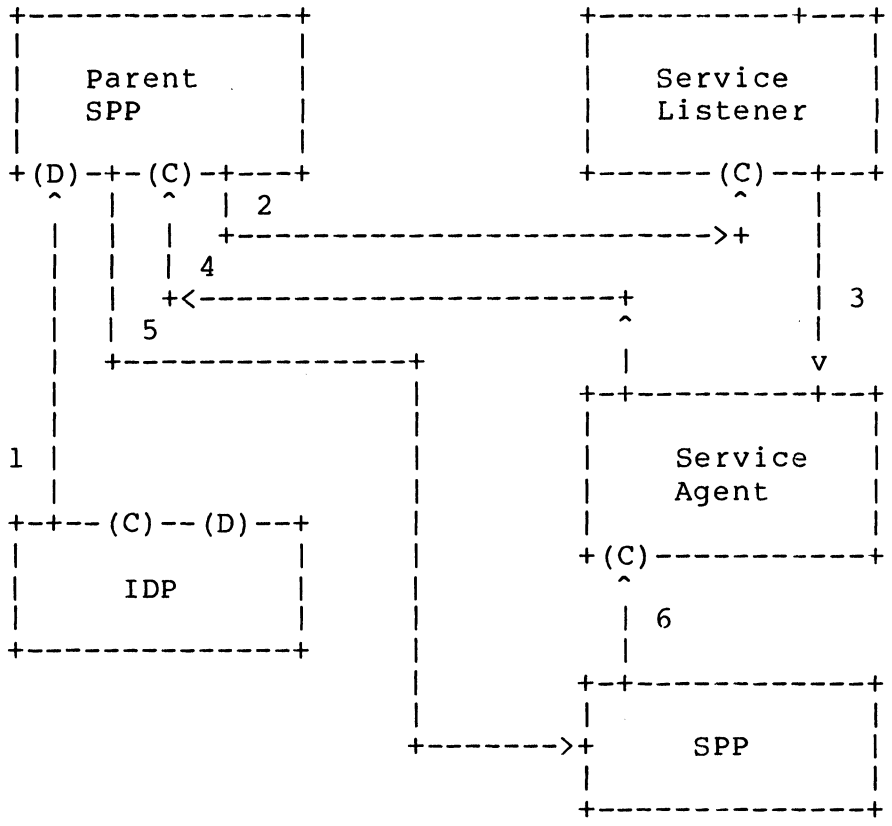


Notes:

(D) = Data mailbox
(C) = Control mailbox

1. Start listening message
2. Socket establishment message
3. Socket open message

Figure 4-3 Service Listener Registration



Notes:

(C) = Control mailbox
(D) = Data mailbox

1. Receive data with initial packet
2. Connection request received message
3. Agent process creation
4. Accept stream message
5. SPP process creation
6. Stream created message

Figure 4-4 Service Listener Agent Creation

4.1.8 The Receive Window

Each connection uses a different, fixed window size. There is a smaller round-trip delay between two SPP clients on CS/1s on the same Ethernet than between two clients on different networks connected by a GS/1 (i.e., an X.25 connection). Therefore, SPP must use a larger window size for a connection involving a GS/1 to achieve the same throughput as on a connection involving only CS/1s.

SPP creates its receive window using a calculated window size based on the round-trip delay experienced by the initial packet of a connection; the upper and lower limits of the window size range are determined at the time of system generation, and override the calculated allocation number. If the calculated number exceeds the range set at system generation, SPP adjusts the number upward or downward accordingly to fit within the range.

4.2 Client Interface to SPP

All interface between SPP and its communicants is accomplished by means of InterProcess Communication (IPC) messages. The following subsections describe these messages and the parameters passed with them. All messages to or from the Parent SPP have the following header structure:

```
typedef struct psmsg
{
        MSG             ps_m;           /* message header */
        SOCKADD         ps_ssocket;     /* source socket  */
} PSMSG;
```

All messages to or from individual SPP processes have the following common header structure:

```
typedef struct spmsg
{
        MSG             sp_m;           /* message header */
        SOCKADD         sp_ssocket;     /* source socket  */
} SPMSG;
```

4.2.1 Create Packet Stream Connection Message

The pscrtp message is sent from a client process to Parent SPP to request the creation of a packet stream connection.

"C" Declaration:

```
typedef struct pscrtp
{
     PSMSG           pscp_m;
     NETADD          pscp_snet;
     HOSTADD         pscp_shost;
     NETADD          pscp_dnet;
     HOSTADD         pscp_dhost;
     SOCKADD         pscp_dsocket;
     MBID            pscp_ctl;
     MBID            pscp_data;
     unsigned short  pscp_mode;
} PSCRTP;
```

Message Parameters:

pscp_m Header (contains source socket number and message type MPSCRTP).

pscp_snet Source network address.

pscp_shost
Source host address.

pscp_dnet Destination network address.

pscp_dhost
Destination host address.

pscp_dsocket
Destination socket.

pscp_ctl Client control mailbox.

pscp_data Client data mailbox.

pscp_mode Mode (LISTEN/INITIATE).

4.2.2 Create Byte Stream Connection Message

The pscrtb message is sent from a client process or service listener to Parent SPP to request the creation of a byte stream connection.

"C" Declaration:

```
typedef struct pscrtb
{
     PSCRTP          pscb_m;
     short           pscb_cflag;
     short           pscb_smax;
     short           pscb_rmax;
} PSCRTB;
```

Message Parameters:

pscb_m Header (contains source socket number, message type MPSCRTB, and all the Create Packet Stream parameters).

pscb_cflag
Combine flag.

pscb_smax Maximum send data length (maximum value 1024).

pscb_rmax Maximum receive data length (maximum value 534).

The combine flag values are defined as follows:

PS_SNDCMB Combine multiple user buffers until an EOM, data stream type change, or the number of bytes specified by pscb_smax are received from the user.

PS_RCVCMB Combine multiple SPP packets until EOM, data stream type change, or the number of bytes specified by pscb_rmax are received from IDP.

4.2.3 Accept Packet Stream Connection Message

The psaccp message is sent from a service listener to Parent SPP to request the creation of a new SPP process, in response to receipt of a "connection request received" message from the Parent SPP.

"C" Declaration:

```
typedef struct psaccp
{
     PSMSG          psap_m;
     MBID           psap_ctl;
     MBID           psap_data;
} PSACCP;
```

Message Parameters:

psap_m Header (containing source socket number and message type MPSACCP).

psap_ctl Client control mailbox.

psap_data Client data mailbox.

4.2.4 Accept Byte Stream Connection Message

The psaccb message is sent from a service listener to the Parent SPP. It is the equivalent of the create packet stream connection message, with the additional fields of information required for a byte stream connection.

"C" Declaration:

```
typedef struct psaccb
{
     PSACCP       psab_m;
     short        psab_cflag;
     short        psab_smax;
     short        psab_rmax;
} PSACCB;
```

Message Parameters:

psab_m Header (containing message type MPSACCB, plus all the Accept Packet Stream parameters).

psab_cflag
Combine flag (PS_SNDCMB or PS_RCVCMB).

pscb_smax Maximum send data length (maximum length 1Ø24).

pscb_rmax Maximum receive data length (maximum length 534).

The combine flag values are defined as follows:

PS_SNDCMB Combine multiple user buffers until an EOM, data stream type change, or the number of bytes specified by pscb_smax are received from the user.

PS_RCVCMB Combine multiple SPP packets until EOM, data stream type change, or the number of bytes specified by pscb_rmax are received from IDP.

4.2.5 Delete Connection Message

The delete connection message uses the PSMSG common header only (shown above in Section 4.2) with a message type of MPSDLT. This message is sent from a client process or service listener to the Parent SPP process to request the termination of the connection associated with the specified socket.

4.2.6 Start Listening Message

The psstart message is sent from a service listener to the Parent SPP process to request that it listen for initial packets on a well-known socket. On receipt of the message, the Parent SPP sends a "socket establishment" message to IDP for the socket, waits for a "socket open" message from IDP, then starts listening for packets.

"C" Declaration:

```
typedef struct psstart
{
     PSMSG          pss_m;
     NETADD         pss_snet;
     HOSTADD        pss_shost;
     MBID           pss_ctl;
} PSSTART;
```

Message Parameters:

pss_m — Header containing source socket number and message type MPSSTART.

pss_snet — Source network address.

pss_shost — Source host address.

pss_ctl — Control mailbox ID.

4.2.7 Stop Listening Message

The stop listening message uses the PSMSG common header only (shown above in Section 4.2) with a message type of MPSSTOP. This message is sent from the service listener to the Parent SPP to request that Parent SPP stop listening for packets on the specified socket.

4.2.8 Connection Request Rejected Message

The connection request rejected message uses the PSMSG common header only (shown above in Section 4.2) with a message type of MSPCREJ. This message is sent from the service listener to the Parent SPP if it cannot accept a connection request on the specified socket.

4.2.9 Parent SPP Reject Message

The psrej message is sent from Parent SPP to the client process if the client sends Parent SPP a message which it cannot understand.

"C" Declaration:

```
typedef struct psrej
{
     PSMSG          psr_m;
     short          psr_mtype;
     unsigned short psr_code;
} PSREJ;
```

Message Parameters:

psr_m Header containing source socket number and message type MPSREJ.

psr_mtype Rejected message type.

psr_code Reject code.

Reject Codes:

PSR_PARM Parameter error.

PSR_BADTYPE
Invalid message type.

4.2.10 Connection Request Received Message

The connection request received message uses the PSMSG common header only (shown above in Section 4.2), with a message type of MSPCONN. This message is sent from Parent SPP to the service listener when an initial packet is received for the specified socket number.

4.2.11 Listening Aborted Message

The pslabtd message is sent from Parent SPP to the service listener to advise the listener that Parent SPP has stopped listening on the specified socket for the specified reason.

"C" Declaration:

```
typedef struct pslabtd

     PSMSG          psa_m;
     unsigned short psa_code;
} PSLABTD;
```

Message Parameters:

psa_m — Header containing source socket number and message type MPSLABTD.

psa_code — Abort code.

Abort Codes:

PSA_RESOURCES
No resource for reqest (1).

PSA_KERROR
Error from kernel call (2).

PSA_NMERROR
Error from call to Network Management (3).

PSA_IDP — Received IDP error message (4).

4.2.12 Delete Stream Message

The delete stream message uses the SPMSG common header only (shown above in Section 4.2) with a message type of MSPDLT. This message is sent from a client process to an SPP processes' control mailbox to request that SPP terminate.

4.2.13 Attention Message

The attention message is sent from a client process to SPP's data mailbox (to mark its place in the data stream) and also simultaneously to SPP's control mailbox (to expedite delivery of the attention byte). The SPP process will convert the pair of attention messages into a single data packet, which will be the next packet sent to the other end of the connection regardless of its sequence number.

The same message is also sent by SPP to the client's data and control mailboxes on receipt of an attention packet. When SPP receives an attention packet from IDP, it immediately sends the packet to the client's control mailbox. However, SPP does not send another attention packet to the client's data mailbox until the packet is in sequence; this allows any missing data packets to be forwarded first.

"C" Declaration:

```
typedef struct spattn
{
     SPMSG     spa_m;
     char      spa_attn;
     char      spa_dst;
     char      spa_eom;
} SPATTN;
```

Message Parameters:

spa_m	Header containing source socket number and message type MSPATTN.
spa_attn	Attention byte.
spa_dst	Data stream type.
spa_eom	End-of-message flag

4.2.14 Data Message

The data message is sent from a client's data mailbox to SPP's data mailbox, or from SPP's data mailbox to the client's data mailbox.

"C" Declaration:

```
typedef struct spdata
{
     SPMSG     spd_m;
     char      spd_dst;
     char      spd_eom;
} SPDATA;
```

Message Parameters:

spd_m	Header containing source socket number, message type MSPDATA, and BD for data.
spd_dst	Data stream type.
spd_eom	End-of-message flag.

4.2.15 Stream Created Message

The spcrtd message is sent from SPP to the client's control mailbox on receipt of the initial packet from the other end of the connection.

"C" Declaration:

```
typedef struct spcrtd
{
     SPMSG          spc_m;
     MBID           spc_ctl;
     MBID           spc_data;
     NETADD         spc_dnet;
     HOSTADD        spc_dhost;
     SOCKADD        spc_dsocket;
} SPCRTD;
```

Message Parameters:

spc_m — Header containing source socket number and message type MSPCRTD.

spc_ctl — SPP control mailbox ID.

spc_data — SPP data mailbox ID.

spc_dnet — Destination network address.

spc_dhost — Destination host address.

spc_dsocket — Destination socket.

4.2.16 Stream Not Created Message

The spabtd message is sent from SPP to the client's control mailbox if an error is detected on receipt of an initial packet.

"C" Declaration:

```
typedef struct spabtd
{
     SPMSG          spb_m;
     unsigned short spb_code;
} SPABTD;
```

Message Parameters:

spb_m — Header containing source socket number and message type MSPNCRTD.

spb_code — Error code.

Error Codes:

SPA_ERROR
: Error protocol packet received (1).

SPA_ALLOC Allocate error (2).

SPA_SNDMSG
: Sendmsg error (3).

SPA_IDERR Error message from IDP (4).

SPA_NOTIFY
: Notifynfull error (5).

SPA_BADMSG
: Invalid message for state (6).

SPA_ALARM Setalarm error (7).

SPA_NORESP
: Other side not responding (8).

SPA_KERROR
: Kernel call error (9).

4.2.17 Stream Aborted

The spabtd message is sent from an SPP process to the client's control mailbox to inform the client that SPP has aborted. No attempt is made to inform the client of any packets in transit.

"C" Declaration:

```
typedef struct spabtd
{
     SPMSG          spb_m;
     unsigned short spb_code;
} SPABTD;
```

Message Parameters:

spb_m — Header containing source socket number and message type MSPABTD.

spb_code — Error code.

Error Codes: Same as stream not created message (see Section 4.2.16).

4.2.18 SPP Reject Message

The sprej message is sent from SPP to the client's control mailbox if the client sends SPP a message which it cannot understand.

"C" Declaration:

```
typedef struct sprej
{
     SPMSG          spr_m;
     short          spr_mtype;
     unsigned short spr_code;
} SPREJ;
```

Message Parameters:

spr_m — Header containing source socket number, message type MSPREJ and BD (if any).

spr_mtype — Rejected message type.

spr_code — Reject code.

Reject Codes:

SPR_PARM — Parameter error (1).

SPR_STATE — Invalid message for state (2).

SPR_NOTYOURS — Not your SPP stream (3).

SPR_BADTYPE — Invalid message type (4).

4.3 Peer Protocol

This version of SPP supports all the required features described in reference [5], except as indicated in Section 4.1 of this manual.

4.4 SPP Sysgen Parameters

This section describes the system generation parameters which apply to the Packet Stream and Byte Stream Services.

4.4.1 Minimum Receive Window Size

This parameter specifies the minimum size of the receive window. The window determines the number of unacknowledged packets which the other end of a connection may send.

4.4.2 Maximum Receive Window Size

This parameter specifies the maximum size of the receive window.

4.4.3 Maximum Number of Retransmissions

This parameter specifies the maximum number of times SPP will retransmit a packet to the destination socket before assuming the other socket is no longer operational and aborting the session.

4.4.4 Maximum Number of Probes

This parameter specifies the maximum number of times SPP will send a probe packet to the destination socket without receiving an acknowledgement before assuming the socket is no longer operational and aborting the session.

4.4.5 Minimum Retransmission Timeout

This parameter specifies the minimum interval (in milliseconds) between attempts to retransmit the oldest unacknowledged packet in the retransmit queue.

4.4.6 Maximum Retransmission Timeout

This parameter specifies the maximum interval (in milliseconds) between attempts to retransmit the oldest unacknowledged packet in the retransmit queue.

4.4.7 Probe Timeout

This parameter specifies (in milliseconds) the interval between probes (system packets with the Send Acknowledgement bit set).

4.4.8 Connection Initialization Timeout

This parameter specifies the interval (in milliseconds) between probes sent to establish a connection.

4.4.9 Maximum Number of SPP Processes

This parameter specifies the maximum number of SPP process which Parent SPP may spawn. The value is governed by the number of ports present in the system, the amount of memory present, and the number of sessions permitted per port.

4.4.10 Initial Packet Mailbox Limit

This parameter specifies the depth of the data mailbox used for receipt of initial packets.

4.4.11 Acknowledgement Timeout

This parameter specifies the interval (in milliseconds) SPP will wait before sending an acknowledgement in response to a Send Acknowledgement request. Rather than immediately creating a special packet for the acknowledgement, SPP waits for the specified interval in case the acknowledgement can be piggybacked on another packet destined to the requesting socket.

5.0 VIRTUAL TERMINAL CONNECTION SERVICE

This section describes the implementation of the Virtual Terminal Connection Service in the ESPL environment.

5.1 Overview

The initial implementation of the Virtual Terminal Service supports TTY-like terminals and TTY hosts. The implementation makes provisions for future enhancements to support intelligent terminals and other protocols.

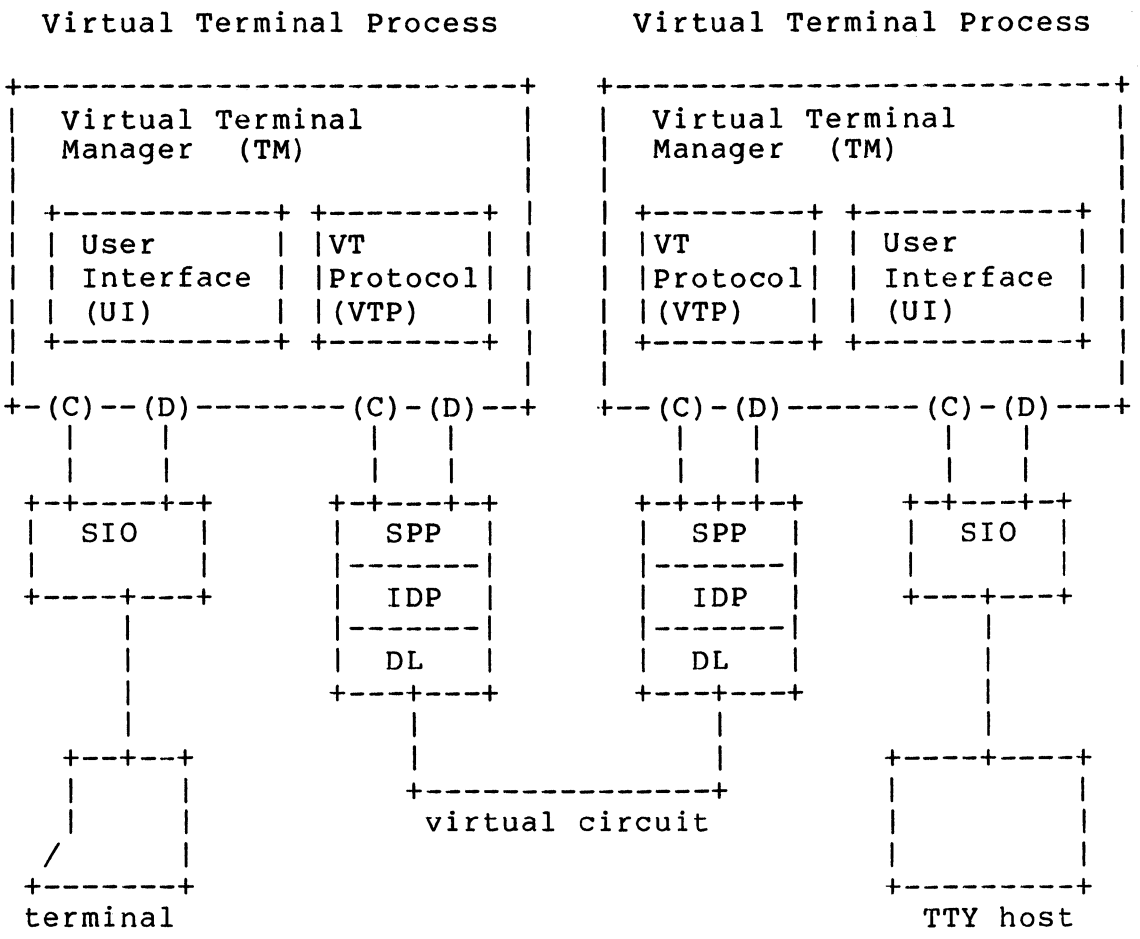
5.1.1 Process Structure

A virtual terminal process is typically composed of three modules: one virtual terminal manager and two message translators. These modules could be implemented as separate processes, but this approach would consume more system resources and CPU time than necessary and is not justified since the three modules are functionally very tightly coupled.

In the ESPL, the Virtual Terminal Connection Service is implemented as a single module containing several subprocesses. The structure of the Virtual Terminal Connection Service module is illustrated in Figure 5-1, which presents an example of Connection Service usage. Following the illustration are definitions of each entity represented.

The ESPL implementation uses a procedure call interface between subprocesses rather than a message interface. The advantage of this scheme is performance; there is no intermediate queuing structure between subprocesses, and there is no case statement to test on the message type. The disadvantage of not using a queuing structure is that the flow control mechanism has to be handled in a special way.

The customer adding software that utilizes the Virtual Terminal Connection Service can use either the same interactive interface to the service as is used by the SIO Agent in the ESPL (a combination message and procedure call interface, described in Section 5.2) or a message-based program interface to the Virtual Terminal Manager subprocess and bypass the User Interface subprocess. Refer to Section 5.3 for details on the program interface.



Notes:

(C) = Control Mailbox
(D) = Data Mailbox

Figure 5-1 Virtual Terminal Connection Service

The entities represented in Figure 5-1 that have not been defined earlier in this manual are as follows:

1. Virtual Terminal Manager (TM). This subprocess manages all the virtual terminal-related data structures. TM performs all necessary error checking, so both UI and VTP can be relatively uncomplicated, and simply perform their message conversion jobs.

2. User Interface (UI). This subprocess defines a set of interactive commands to provide users a easy method of network access. A subset of the UI function provides a host interface, which communicates with a host or a terminal-emulator process rather than with a human being.

3. Virtual Terminal Protocol (VTP). This subprocess translates a network virtual terminal message into an internal event and reports it to TM, and sends network virtual terminal messages on behalf of TM.

   The ESPL VTP utilizes the XNS Courier protocol as defined in reference [6]. Each VTP consists of client and server portions, and the communication between a VTP client and a remote VTP server is via Courier remote procedure calls.

   As with SPP, the ESPL implementation consists of a Parent VT process plus an individual VT process for each active port. The individual VT processes share the same code and utilize the kernel's stack sharing mechanism.

4. Serial I/O (SIO) Module. The SIO module consists of firmware residing on the SIO board and a software agent residing on the MCPU board. The primary function of the SIO Agent is to provide an interface to the SIO firmware. The Agent and the firmware together comprise an interrupt-driven mechanism for transferring data, attention and flow-control signals to and from serial devices attached to the SIO board. The SIO Interface is described in detail in Volume Three, Section 3.0.

5.1.2 Initialization

At the time of system initialization, the "init" process creates a Parent VT process, creates a default mailbox for the process, and makes the Parent VT process runnable. The Parent VT's default mailbox name is "PV". When the Parent VT process initializes, it learns the Parent SPP's default mailbox identifier and the SIO port configuration. All the SIO ports are initialized by the Parent VT process. The Parent VT process accesses all the sysgenable parameters loaded by the bootstrap loader, and transfers SIO-related parameters to the SIO firmware through a procedure call provided by the SIO Agent (sav2ssparm, described in Volume Three, Section 3.4).

After initialization, the primary task of the Parent VT is to spawn a new child VT process whenever a connection request is made by the SIO Agent or the Parent SPP.

The Parent VT sends a message to the Parent SPP requesting that the Parent SPP start listening on socket number 5.

As defined in reference [6], socket number 5 is the well-known Courier socket. Any Courier remote procedure call from a remote entity is addressed to this socket number. If VT receives an unimplemented Courier procedure call, it rejects the call with the error code defined in reference [6].

5.1.3 Connection Establishment

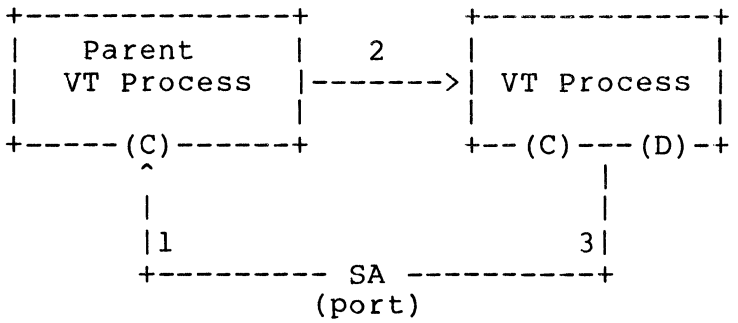
The process of connection establishment consists of three main steps. The first step encompasses the actions taken when a connection is initiated by the SIO agent and accepted by a VT process. The second step encompasses the actions required to set up a connection between the local VT process and the remote VT process. The third step includes the actions taken when the connection is completed from the remote VT process to the remote SA.

Step One (SA to VT)

The first major step of the connection establishment procedure occurs when a connection is initiated from SA and accepted by a virtual terminal process.

Whenever the SIO firmware detects the presence of a terminal, and the state of the detected port is DISCONNECTED, the firmware asks SA to issue a connection request message to the Parent VT process, specifying the port ID. The Parent VT then spawns a child VT process and passes on the connection request to it. The child VT process issues a procedure call provided by SA to acknowledge this connection, including as parameters the appropriate control and data mailbox IDs. The SIO port is then put into the CONNECTED state. Note that this CONNECTED state refers to the port being "connected" to a VT process; it does not imply a connection to another port.

Figure 5-2 illustrates this step.



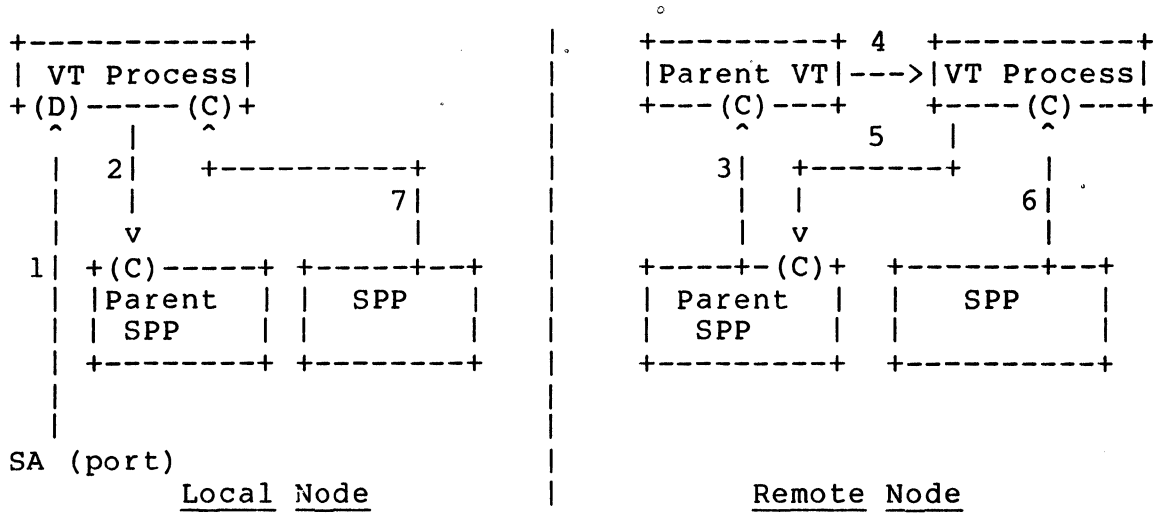
Notes:

1. Connection request with portid specified.
2. VT process spawned, portid forwarded.
3. VT replies to the port with mailbox IDs.

Figure 5-2 Connection Establishment, Step One

Step Two (VT to VT)

This step occurs when the device wishes to establish a connection with another port. During this step, a connection is established between the local VT process and the remote VT process through the SPP Packet Stream Service. The detailed connection mechanism is described in Section 4.0. Figure 5-3 provides a summary of this mechanism. It shows the message flow between the VT process and the related system elements. However, it does not show the message flow between processes lower than SPP and Parent SPP.



Notes:

1. "Connect host" data string is sent from the local terminal to VT, and translated by the UI subprocess.
2. VT sends create stream message to Parent SPP, who responds by spawning a child SPP.
3. The remote Parent SPP sends a connection request received message to the remote Parent VT.
4. The remote Parent VT spawns a child VT process and forwards the connection request.
5. The new VT sends Parent SPP a create stream message, who responds by spawning a child SPP.
6. The new SPP sends VT a stream created message.
7. The local SPP sends the local VT a stream-created message when an initial packet is received from the remote station.

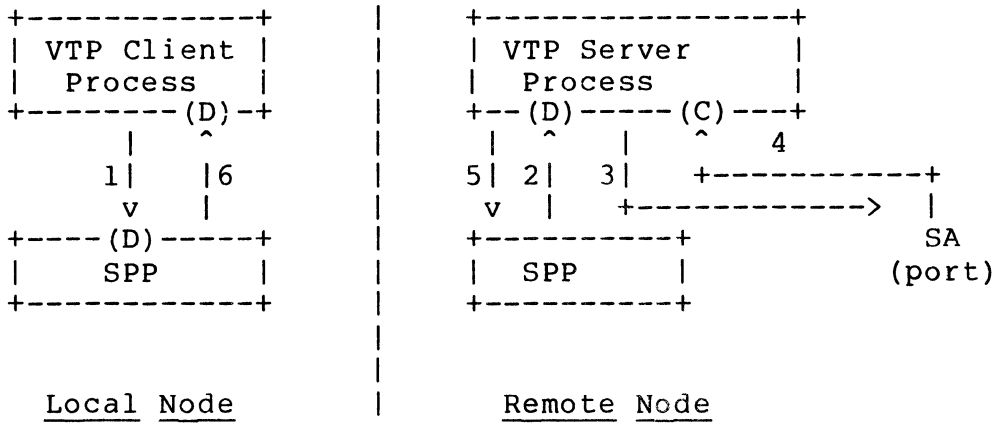
Figure 5-3 Connection Establishment, Step Two

Step Three (Remote VT to SA)

During this step, the connection is initiated by the remote VT process to the remote SIO agent.

The SIO agent may receive a phone number from the virtual terminal process if it is a switched line; this request will cause the modem signals to activate or dial the phone number through an autodialer. On receiving the answering signal from the remote host, the SIO agent will issue a connected message back to the VT process and put itself into the CONNECTED state. If the host device is directly connected, the SIO agent will immediately return a connected message and put itself into the CONNECTED state.

Figure 5-4 illustrates this step.



Notes:

1. VTP client issues CREATE call via a Courier message, specifying phone number, port ID, device type, transport information.
2. VTP server receives CREATE call from SPP.
3. VT process initiates dialing request (if necessary).
4. Remote modem answers (if necessary).
5. VTP server builds Courier RETURN message and ships it.
6. VTP client completes CREATE call.

Figure 5-4 Connection Establishment, Step Three

5.1.4 Multiple Session Support

Each virtual terminal process is capable of creating multiple sessions by opening multiple SPP streams; each SPP stream is mapped into a separate mailbox. However, the VT process has a perception of "current session", and all non-current sessions are suspended and flow-controlled. Users are able to change the current session by entering command mode and typing a SWitch command.

5.2 Communication Between VT and Client/Service Supplier

This section describes the portion of the interactive interface which is defined by the UI subprocesses. This interface might be used when replacing the SIO agent. Section 5.3 defines the program interface to the VT process.

The portion of the interface between VT and SPP which is defined by SPP is described in Section 4.0. The portion of the interface between VT and the SIO agent which is defined by the agent is described in Volume Three, Section 3.0.

5.2.1 SA Data String Procedure Call

This procedure call is used to process data strings received from the SIO Agent. It causes UI to begin parsing the data string if UI is in Command Mode, and to pass the data string directly to TM if UI is in Data Transfer Mode.

"C" Declaration:

```
uiS2Udata ( data, eom )
BD *data;
ushort eom;
```

Input Parameters:

data Pointer to data buffer descriptor.

eom Either TMEOM or !TMEOM.

Output Parameters: None

5.2.2 SA Attention Signal Procedure Call

This procedure call is used to process an attention signal received from SA.

"C" Declaration:

```
uiS2Uattn ( signal )
ushort   signal;
```

Input Parameters:

signal The attention signal.

Output Parameters: None

5.3 Program Interface to VT

The descriptions of the VT Connection Service module and subprocesses provided in Section 5.1 assume the standard interactive access to VT via the User Interface. An interface to VT is also available to a program (typically to an application program agent that requires access to the Connection Service).

For the program interface to VT, the command passing and interpretation functions provided by UI are not needed, so the UI subprocess is replaced by a Program Interface Handler (PIH). The same facilities that were available to UI are available to the VT client, via messages to VT's data and control mailboxes. The messages are received by PIH and directly mapped onto the procedure calls that UI would have issued to TM.

This message-based program interface permits a clean decoupling between Bridge Communications' VT code and the customer's client code. No limitation is imposed on the VT client as far as kernel calls and stack sharing rules are concerned.

The Program Interface VT (PI VT) process treats the VT client as a virtual port, unlike the normal VT which deals with a physical port represented by the SIO agent. Figures 5-5 and 5-6 illustrate the difference in communication between the two types of VT process and their clients.

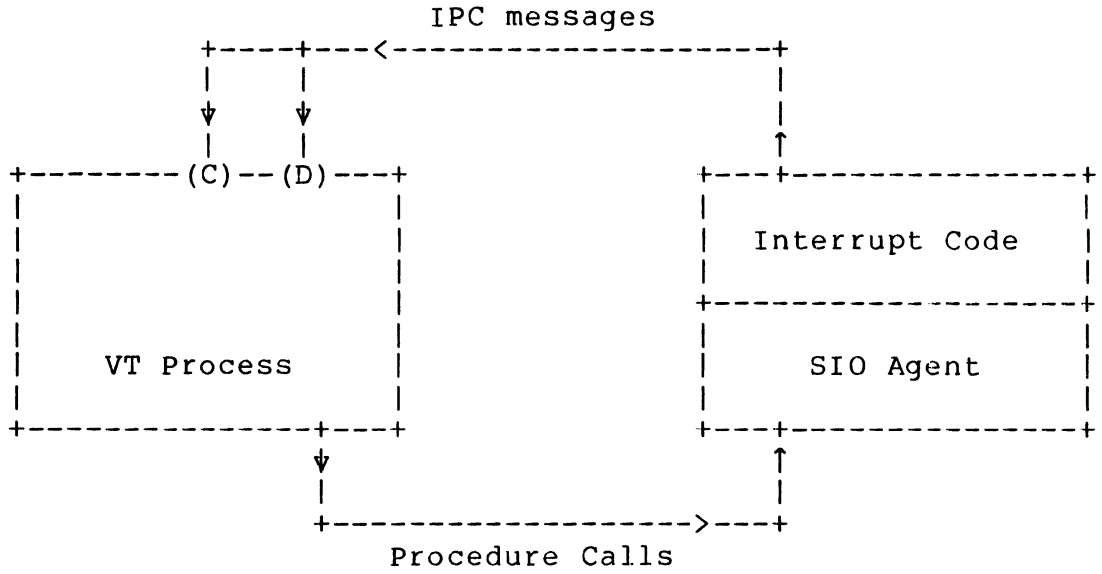


Figure 5-5 Interactive VT to Physical Port Communications
(Used in Standard ESPL Connection Service)

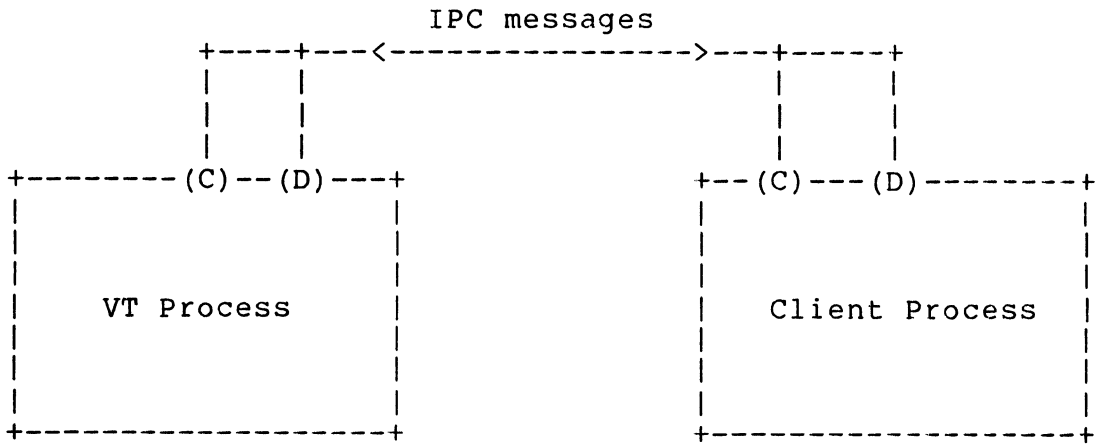


Figure 5-6 PI VT to Virtual Port Communications
(Used in Program Interface to VT)

The following subsections provide descriptions of the additional IPC messages used to implement a program interface between VT and a client program.

The program interface is accessible through a set of system elements called virtual ports. The VT Connection Service can support up to 48 virtual ports. Each virtual port exists as a structured array containing state information and mailbox identifiers used for communication between a client process and a PI VT process.

5.3.1 Open Port Message

This message is sent by the client to Parent VT's control mailbox to request that a free virtual port be opened. By specifying a destination address, the client may optionally request that a connection be established when the virtual port is opened. If the destination address is not specified at the time the port is opened, the client may use the Connect message to establish a connection at a later time.

"C" Declaration:

```
VT_OPEN {
     MSG   m;
     short portid;
     MBID  cmbox;
     MBID  dmbox;
     LADDR dest_addr;
     }
```

Message Parameters:

m	Kernel message header (message type MPIOPEN).
portid	Virtual port identifier (a portid of -1 is used to request any available unused virtual port).
cmbox	Client control mailbox ID.
dmbox	Client data mailbox ID.
dest_addr	Destination address of the connection.

5.3.2 Port Opened Message

This message is sent by the newly created VT to the client's control mailbox to indicate that a virtual port has been opened. The client process should record VT's control and data mailbox IDs for subsequent access to the virtual port.

"C" Declaration:

```
VT_OPENED {
     MSG   m;
     short portid;
     MBID  cmbox;
     MBID  dmbox;
     }
```

Message Parameters:

m	Kernel message header (message type MPIOPENED.
portid	The virtual port identifier to which the VT process is attached.
cmbox	The VT process's control mailbox ID.
dmbox	The VT process's data mailbox ID.

5.3.3 Close Port Message

This message is sent by the client to VT's control mailbox to request the closing of a virtual port.

"C" Declaration:

```
VT_CLOSE {
     MSG   m;
     short portid;
     }
```

Message Parameters:

m	Kernel message header (message type MPICLOSE).
portid	The virtual port identifier.

5.3.4 Port Closed Message

This message is sent by VT to the client's control mailbox to acknowledge that the specified virtual port has been completely closed.

"C" Declaration:

```
VT_CLOSED {
     MSG   m;
     short portid;
     }
```

Message Parameters:

m	Kernel message header (message type MPICLOSED).
portid	The virtual port identifier.

5.3.5 Connection Request Message

This message is sent by the client to VT to request the establishment of a connection, or is sent by VT to the listening client's control mailbox to indicate that an incoming connection request has been received.

"C" Declaration:

```
VT_CONNECT {
     MSG     m;
     short   portid;
     LADDR   dest_addr;
     }
```

Message Parameters:

m	Kernel message header (message type MPICONNECT).
portid	Virtual port number.
dest_addr	Destination address of the connection.

5.3.6 Connected Message

This message is sent by VT to the client process's control mailbox to inform the client of the establishment of a connection, or is sent by the client process to VT's control mailbox to acknowledge the incoming connect request.

"C" Declaration:

```
VT_CONNECTED {
     MSG   m;
     short portid;
     }
```

Message Parameters:

m	Kernel message header (message type MPICONNECTED).
portid	Virtual port identifier.

5.3.7 Data Message

This message is sent by the client to VT's data mailbox to request transmission of data for a connection, or is sent by VT to the client's data mailbox to indicate that data has been received.

"C" Declaration:

```
VT_DATA {
     MSG  m;
     short portid;
     short eom;
     }
```

Message Parameters:

m	Kernel message header, containing a pointer to the buffer descriptor for the data and a message type of MPIDATA.
portid	Virtual port identifier.
eom	End of message flag.

5.3.8 Disconnection Request Message

This message is sent by the client process to VT's control mailbox to disconnect a connection, or is sent by VT to the client's control mailbox to indicate that a disconnection request has been received.

"C" Declaration:

```
VT_DISCONNECT {
     MSG  m;
     short  portid;
     }
```

Message Parameters:

m	Kernel message header (message type MPIDISCONNECT).
portid	Virtual port identifier.

5.3.9 Disconnected Message

This message is sent by VT to the client's control mailbox to complete the disconnection sequence, or is sent by the client to confirm VT's incoming disconnect request.

"C" Declaration:

```
VT_DISCONNECTED {
     MSG       m;
     short     portid;
     ushort    ret_code;
     }
```

Message Parameters:

m	Kernel message header (message type MPIDISCONNECTED).
portid	Virtual port identifier.
ret_code	Reason for the disconnection.

Error Codes:

NOERROR	No error detected (Ø).
ILLREQ	Illegal request or request in wrong state (-1).
NOMEM	No memory resource available (-2).

INCOMP	Remote software incompatible (-3).
RMTHWERR	Remote hardware error (-4).
RMTBUSY	Remote is busy (-5).
BADADDR	Remote received invalid address (-6).
UNKNOWN	Remote detected unknown error (-7).
BADPARM	Remote received bad parameter (-8).
NOSESSION	No more sessions for this port (-9).
TPFAIL	Transport layer failure; no response (-10).
TMREXPIRED	Timeout failure (-11).

5.3.10 Attention Message

This message is sent from the client to VT's control mailbox when the client wishes to send an attention on a connection, or is sent by VT to the client's control mailbox when an attention signal is received.

"C" Declaration:

```
VT_ATTN {
     MSG    m;
     short  portid;
     ushort ioband;
     }
```

Message Parameters:

m	Kernel message header (message type MPIATTN).
portid	Virtual port identifier.
ioband	Indicates in-band or out-of-band attention signal.

5.3.11 Start Listening Message

This message is sent by the client to the Parent VT's control mailbox to authorize acceptance of incoming connection requests to a virtual port. The Parent VT will continue listening on the specified port until the client sends the Parent VT a Stop Listening message for the port.

"C" Declaration:

```
VT_LISTEN {
     MSG     m;
     ushort  portid;
     MBID    cmbox;
     MBID    dmbox;
     }
```

Message Parameters:

m	Kernel message header (message type MPILISTEN).
portid	Virtual port number (a portid of -1 indicates that the client process wishes to listen to incoming connect requests on any virtual ports).
cmbox	Client control mailbox ID.
dmbox	Client data mailbox ID.

5.3.12 Stop Listening Message

This message is sent to PV's control mailbox to cancel a previous Start Listening message.

"C" Declaration:

```
VT_STOP {
     MSG     m;
     ushort  portid;
     }
```

Message Parameters:

m	Kernel message header (message type MPISTOP).
portid	Virtual port number.

5.4 Virtual Terminal Connection Service Sysgen Parameters

This section describes the Sysgen parameters which apply to the VT Connection Service.

5.4.1 Maximum Number of Sessions

This parameter specifies the maximum number of sessions that Parent VT will permit.

5.4.2 Mailbox Depth to SPP

This parameter specifies the depth of the mailbox used for messages between TM and each SPP process.

5.4.3 Mailbox Depth to SIO Agent

This parameter specifies the depth of the mailbox used for messages between TM and the SIO Agent (SA).

5.4.4 User Interface Buffer Size

This parameter specifies the maximum size of buffers allocated to UI, and also controls the maximum number which may be specified by the configuration parameter "BufferSize". If a user sets "BufferSize" to a number larger than the number specified by the Sysgen buffer size parameter, the user's port hangs waiting for the buffers to be allocated. Since allocation is only done at initialization or Sysgen time, the buffers never become available. The only way to unhang the port is to use the SET command remotely to respecify the "BufferSize" configuration parameter.

5.4.5 Port Number Prompt

When set to "1", this parameter changes the CS/1 prompt from the default prompt (typically "CS/1>") to a prompt that incorporates the SIO port number. When set to "Ø", the default CS/1 prompt will appear.

5.4.6 Clearinghouse Domain

This parameter specifies the Clearinghouse domain name.

5.4.7 Clearinghouse Organization

This parameter specifies the Clearinghouse organization name.

5.4.8 User Interface Privilege Levels

This section describes the default privilege levels associated with UI commands, keywords and configuration parameters which may be changed at Sysgen time.

The following privilege levels (listed from lowest to highest priority) may be specified:

User = User Privilege Level
LNM = Local Network Manager Privilege Level
GNM = Global Network Manager Privilege Level

Table 5-1 lists all the User Interface Commands and indicates the default privilege level required for each command.

Table 5-2 lists all the keywords that may be specified with the SHow command and indicates the default privilege level required for each keyword. Note that the SHow command only displays those keywords which are valid for the privilege level currently assigned to the session.

Table 5-3 lists the configuration parameters and indicates the privilege level required to set each parameter via the SET and SETDefault commands. The SET command is used to specify configuration parameters for the current session only, while the SET-Default command is used to specify default configuration parameters for all new sessions.

Table 5-1 UI Command Privilege Levels

Command Name	Default Required Privilege
Broadcast	LNM
Connect	User
DEfine	LNM
DisAble	LNM
DisConnect	User
DO	User
Enable	LNM
Listen	User
Name	LNM
ReaD	LNM
REName	LNM
RESume	User
RmtSet	LNM
ROtary	LNM
SAve	LNM
SET *	User
SETDefault *	LNM
SHow *	User
SWitch	User
Transmit	User
Unname	LNM

* Note that the command itself has a required privilege level; the keywords or parameters applicable to the command may have varying privilege requirements. Refer to Tables 5-2 and 5-3 for information on keywords and parameters, respectively.

Table 5-2 SHow Keyword Privilege Levels

Keyword Name	Default Required Privilege
AllSessions	LNM
CHNames	User
CSlStrings	LNM
Date	User
DefaultParameters	User
NetMAP	User
Macros	User
PARAmeters	User
Rotaries	User
SEssions	User
STATistics	LNM
VERSion	User

Table 5-3 Configuration Parameter Privilege Levels

Parameter	Privilege	Parameter	Privilege
AutoDisconnect	User	GlobalPassWord	GNM
BAud	User	IdleTimer	User
BReakAction	User	InterAction	User
BReakString	User	LFDelay	User
BSDelay	User	LFPad	User
BSPad	User	LineERase	User
BUffersize	User	LocalEDiting	User
CRDelay	User	LocalPassWord	LNM
CRPad	User	MaxSessions	LNM
DataBits	LNM	MOde	User
DataForward	User	NMPrompt	LNM
DAte	LNM	PARIty	LNM
DeVice	User	PRIvilege	User
DisableECHO	User	PROMPt	LNM
DUplex	User	ReprintLine	User
ECHOData	User	StopBits	LNM
ECHOMask	User	TabDelay	User
EnterCmdmodeStr	User	TabPad	User
EOMChar	User	UseDCDout	LNM
EOMMapping	User	UseDTRin	LNM
ERase	User	VERBatim	User
FFDelay	User	WelcomeString	LNM
FFPad	User	WordERase	User
FlowControlFrom	User	XOFF/XON	User
FlowControlTo	User		

5.5 Changing Table-Driven User Interface Strings

At the time of system generation, the only changes to the User Interface which may be made are related to default privilege levels applicable to the UI commands and parameters. In addition to changing privilege levels, the user may wish to modify the strings used to execute UI commands or to specify UI parameters, or the error messages issued by the UI. Table 5-4 indicates the UI source files which must be edited (and recompiled) in order to make such modifications.

Table 5-4 Table-Driven User Interface Strings

String Type	Source File
Command name	cslrlse/ui/src/uiram.c
Error message	cslrlse/ui/src/uidisplay.c
Parameter name or parameter value	cslrlse/ui/src/pnpv_tbls.c

Bridge
Communications
Inc.

Bridge Communications, Inc.
10440 Bubb Road
Cupertino, California 95014
408/446-2981