Layered Communications

PSD 76.97.11

0 DOCUMENT CONTROL

0.1 Contents List

0.2 Changes since previous issues

Changes bringing document to 2/1

- section 0.2 (this section !) added
- section 0.3 added
- section 5 added
- auto re-dial interfaces removed
- numerous corrections and additions to section 4
- numerous corrections and amplifications throughout
- significant changes are highlighted with a single margin bar

Changes bringing document to issue 2/1

- various minor corrections
- changes are highlighted with double margin bars

0.3 Changes forecast

- the provision of the MNP file transfer layer into TLink
- provision of attention services within TLink
- provision of stream mode working
- provision of window size greater than 1
- provision of Bad Number List Processing within TLink to govern automatic redialling attempts
- provision of CALL PROGRAM interfaces rather than TRAP interfaces into TLink to support paged ROM

0.4 Documentation Cross-references

1 PSD 76.97.3.1 OPD Kernel Specification. Issue 5/2 R.R.Walton.

2 PSD 76.97.3.2 OPD Director Facilities for Application Writers. Issue 3/1 R.M.Mahon

3 Microcom Networking Protocol Specification Version 1/0. Microcom Inc., Norwood, MA, USA 15-Aug-83.

4 Reference deleted

5 PSD 76.97.3.3 OPD Telephone Handler Interfaces for Application Writers. Issue 2/1. G.F.Winter

6 TLink Physical Layer Component Product Design Document Issue 4. P. J. Vickers

7 TLink Link Layer Component Product Design Document Issue 6. P. J. Vickers

8 TLink Session Layer Component Product Design Document Issue 4. P. J. Vickers

9 TLink Manager Component Product Design Document Issue 2. R.McIntosh Shand

10 PSD 76.97.4 Telephony Module. Issue 2/3 A.Priestley

11 PSD 76.97.2.1 User Interface - Base Functional Software. Issue 2/3

12 Anatomy of a Microcomputer Protocol. Gregory Pearson Data Communications March 1983 pp 231-239.

13 PSD 76.97.41 Policing Auto Re-dial Attempts. Issue 2/1. R. M. Shand

1 GENERAL

1.1 Scope.

This document provides a description of the release 1 implementation of TLink - the Microcom Networking Protocol (MNP) - on the ICL One-Per-Desk (OPD) product, to a level of detail sufficient to allow software developers to use the facilities of TLink. It does not aim to provide implementation details, nor the method of implementation, nor the design of the product, nor does it describe the protocol itself, which is fully described in ref 3 and summarised in ref 12.

1.2 Introduction.

TLink has been adopted by British Telecom Enterprises (BTE) as a "de facto standard" for personal computer communications through the PSTN. It has therefore been chosen as the protocol to be used by ICL OPD for interpersonal messaging and other applications which require reliable communications between OPDs. This PSD describes the 'user interface' to TLink, by which is meant a description of the facilities of the protocol which are implemented on OPD and the method by which an application program gains access to them.

1.3 Terms and Abbreviations.

AGRA	Automatically Generated Redial Attempt. An attempt, initiated by software, to place a call to a number to which a previous call attempt had failed.
BNL	Bad Number List. A list of numbers to which attempts to establish data calls have failed.
BSC	Binary Synchronous Communications.
BTE	British Telecom Enterprises.
EPC	Evade Police Character. A character preceeding a dial string which causes circumvention of the checks for the number of redial attempts and the interval between them.
FTE	File Transfer Entity. An instantiation of a body of code which provides some or all of the services defined by the File Transfer Protocol of MNP.
ISO	International Standards Organisation
LE	Link Entity. An instantiation of a body of code which provides some or all of the services defined by the Reliable Link Protocol of MNP.

MNP	Microcom Networking Protocol. A layered protocol providing Link, Session and File Transfer services (ref 3).
OPD	ICL One-Per-Desk product.
OSI	Open Systems Interconnection.
PSTN	Public Switched Telephone Network
SE	Session Entity. An instantiation of a body of code which provides some or all of the services defined by the Session Protocol of MNP.
TH	Telephone Handler. The software subsystem concerned with all aspects of telephony (ref 5).
TLink	The marketing name chosen by British Telecom for MNP.
TLM	TLink Manager. A permanent OPD application which governs access to TLink for applications.
$	Values preceded by "$" are hexadecimal. For example, "$13" represents hexadecimal 13, decimal 19.

2 SUMMARY FOR MANAGEMENT

TLink is the name adopted by British Telecom Enterprises (BTE) for the Microcom Networking Protocol (MNP - ref 3), a protocol designed to provide reliable communications between personal computers across the Public Switched Telephone Network (PSTN). TLink has been adopted as the standard data communications protocol to be implemented on the ICL OPD. It provides a reliable link protocol at layer 2 of the ISO OSI model, a session protocol at layer 5 and a file transfer protocol, including the concept of a virtual filestore, at layer 7. An article describing the protocol is given as ref 12.

TLink has been widely accepted in the microcomputer industry by both hardware and software vendors. Among the licencees are Visicorp, Lotus Development Corporation, Apple, IBM and others. In addition, a number of network suppliers (such as GTE Telenet, Dow Jones News/Retrieval) are supporting TLink as an access protocol to packet-switched networks, financial information systems and electronic mail services.

This document describes the level of implementation of TLink on OPD at release 1, and the interface supplied to applications which require some or all of the services of TLink. It does not provide details of the implementation, nor a description of the protocol. The reader is assumed to be familiar with the MNP protocol.

3 TECHNICAL DESCRIPTION

3.1 Overall Description.

3.1.1 Features and characteristics.

TLink provides an efficient, reliable communications protocol suitable for use between personal computers across the PSTN. It provides a reliable link protocol, a session protocol and a file transfer protocol which includes the concept of a virtual directory.

At first release, TLink is utilised in OPD to provide communications for messaging applications. The physical layer described in this document is designed to provide services for applications other than TLink.

The release 1 implementation of TLink consists of Physical, Link, Session and TLink Manager tasks. These run independently of each other (asynchronously), but communicate through named pipes. TLink Manager exists to co-ordinate the functioning of the layer entities, provide a memory-management service for Applications, and control use of the communications protocol.

The layer entities communicate through pipes, and the messages they use have been designed to match, as far as possible, the layer primitives defined in ref 3. Additional parameters to the primitives have been defined to match local operating and system requirements. The approach taken has been to make the design extensible with the minimum alteration to existing interfaces and code.

3.1.2 Product Use.

In OPD release 1 TLink will be used to supply a reliable transfer service for messaging applications and certain file transfer functions. Other applications may use the physical layer, which is not specified in ref 3 but is regime-dependent.

3.1.3 Limitations

The release 1 implementation of TLink is a proper subset of the protocol specification (ref 3). Details of the limitations are given in the descriptive sections 3.2.2 to 3.2.4. In brief the limitations are:

Physical Layer	The limitations are those imposed by the modem hardware available: V21, V23, Bell 103 and Bell 202 working only.
Link Layer	Default link serial number used. No Link Attention services. Only asynchronous, half-duplex (logical) working with BSC framing and octet-stuffing

	for data transparency supported. Flow control window size is fixed at 1. Only block-mode working supported.
Session Layer	No Session Attention services.
File Transfer Layer	Not implemented at first release.

3.1.4 Foreign Language Requirements

There is no natural language text used or produced by TLink, and thus no requirements for foreign language variants. Should an application interface be provided at a later date, however, there may be a requirement for foreign language versions of the interface specification.

The housekeeping routine which allows users to remove numbers from the Bad Number List (BNL - s. 4.4) will have to be adapted for each foreign language version of OPD.

3.2 Detailed Descriptions.

The implementation of TLink on OPD has separate Physical, Link and Session activities, and a TLink Manager activity. Later releases may include Network and File Transfer activities. TLink Manager manages the initialisation of the other activities and provides the route by which applications connect with TLink activities. Each of these activities is described separately in the following sections 3.2.1 to 3.2.4 .

3.2.1 TLink Manager.

TLink Manager is the entity through which any of the services offered by TLink are made available to an Application. This section describes the interactions of TLink Manager (TLM) with both TLink and a requesting Application. An Application may interface to either the Physical Link or Session layers. These interactions are described in this section and in sections 3.2.2 to 3.2.4 following.

A further case, that of a data call being auto-answered, is described in section 3.2.5.

3.2.1.1 Establishing contact with TLink Manager (data call).

TLink Manager (TLM) will be a permanent activity, that is, it will be loaded at power-on. An application which requires TLink must:

1 Obtain TLM's activity number. This it can do with a LOAD PROGRAM (ref 2 s. 4.9.3) using the target program name " L ". The program name is <tilde><L>< >< >< >< >< >< >< >< >< > or as a hex string $7E4C20202020202020202020

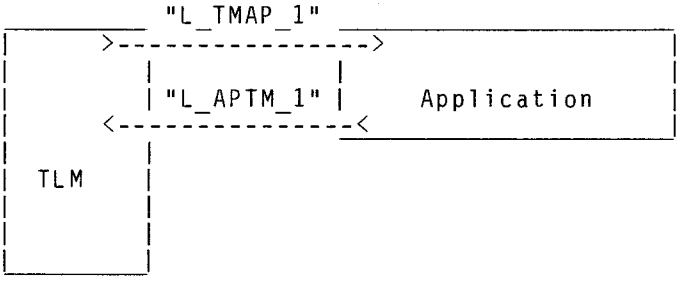
2 Open two pipes (ref 1 s. 17.5.1): one called "L_APTM_1", through which the the Application will send data to TLM; one called "L_TMAP_1", through which TLM will send data to the Application. The pipes must be opened in the order "L_TMAP_1", "L_APTM_1". If the response IU is obtained, then TLM is handling requests from another Application, so the requestor should wait 5s before retrying. The number of retries made, and the intervals between them, are at the Application writer's discretion.

Normally the response will be a new channel identifier, although the channel will not be complete. The Application sends a **TLM_SERVICE.request** (s. 4.2.2.1.1) through pipe "L_APTM_1", but it will not be received by TLM until the pipes are complete.

3 Cause local event 14 to TLM using the activity number obtained in step 1 (SIGNAL LOCAL EVENTS ref 1 s. 6.8.4).

TLM will detect local event 14 and will open the other ends of pipes "L_TMAP_1" and "L_APTM_1". It will wait for no more than 2s for a **TLM_SERVICE.request** to appear in pipe "L_APTM_1". The **TLM_SERVICE.request** will contain the Application's activity number and the type of service required. If no command has appeared within 2s TLM will close both pipes and wait to be signalled on local events.

The situation may be represented:



The request sent contains enough information for TLM to initialise the required TLink services. Once TLM has initialised the TLink entities it will send a **TLM_SERVICE.response** (s. 4.2.1.1.1) to the Application through pipe "L_TMAP_1". The information contained in the response will indicate to the application whether its request was successful or not.

If the request was successful the **TLM_SERVICE.response** will contain the names of pipes through which the Application will communicate with the requested service. For example, if the Application had requested Session services the pipe names might well be "L_SEFT_1" and "L_FTSE_1". The Application now opens the pipes named in the **TLM_SERVICE.response** and sends an **APP_READY.indication** (s. 4.2.2.1.2) to TLM. If it should fail it instead sends an **APP_CLOSE.indication** (s. 4.2.2.1.3) before closing down.

After successfully opening the pipes, the Applications closes pipes "L_APTM_1" and "L_TMAP_1" and commences communication with the layer entity. In the example given, the first action of the Application would be to send an **S_CONNECT.request** through pipe "L_FTSE_1".

3.2.1.2 Establishing contact with TLink Manager (data-in-voice call)

This is the same as for a data call (see previous subsection) excepting only that in step 3 local event 9 as well as local event 14 must be caused to TLM.

3.2.1.3 Control summary

The sequence of events described above may be summarised:

TLink Manager	Application
	Obtain TLM's activity No.
	Open pipes
	TLM_SERVICE.request <----------
	Signal local event 14 to TLM Signal local event 9 to TLM (if data-in-voice)
Open pipes	
Start requested TLink services	
TLM_SERVICE.response ---------->	
	Open pipes named in the **TLM_SERVICE.response.**
	APP_READY.indication <----------
	Close pipes "L_TMAP_1" and "L_APTM_1".
	Commence communication with the layer entity.

3.2.2 Physical Layer.

Note that this section includes a description of the physical layer services provided. It follows the format used in the MNP specification. However, it is not necessary (although helpful) for the user of physical services to be familiar with the Microcom specification.

3.2.2.1 Overview of Physical services.

The Microcom specification (ref 3) does not define a Physical Layer, but makes reference to the services required of it. On OPD the Physical Layer is designed to provides Physical services to the TLink Link entity and other activities as well.

The Physical Layer provides the following services:

Connection Establishment - dialling a specified number and configuring the modem as required.

Data Transfer - Transmission and reception of data in either full or half duplex on the line. The interaction with the user of the Physical Layer is full duplex.

Termination - Closing of the line and modem.

3.2.2.2 Specification of services.

3.2.2.2.1 Physical Connection Establishment service.

The P_CONNECT service is provided to establish a Physical connection. This provides a connection between two Physical entities which are capable of supplying a service to a link entity. The provision of the connection does not, however, require nor imply that an MNP link service will be supplied by the distant station.

There are two primitives associated with the P_CONNECT service: the **P_CONNECT.request** and **P_CONNECT.confirm**. The **P_CONNECT.request** indicates to the local Physical entity that it is to attempt a Physical connection. The **P_CONNECT**.confirm indicates that the Physical connection requested has been established.

```
P_CONNECT.request (
                    from U: Modem Details
                    from U: Data Receiver Identity
                    from U: Receive Event Number
                    from U: Line Number
                    from U: Timing Information
                    from U: Charge Band
                    from U: Physical Address.
                   )
```

P_CONNECT.confirm (
)

The implementation details of these and other primitives are given in section 4.

The Modem Details specify the configuration of the modem for connection establishment.

The Data Receiver Identity is the activity identifier of the Physical user.

The Receive Event Number is the event which will be signalled when data is sent by PE to the Physical user.

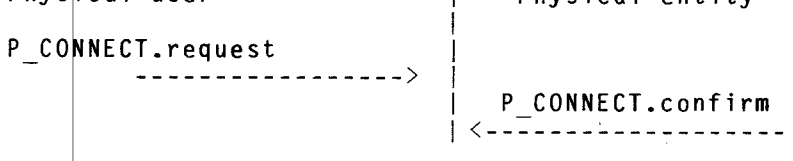
The Line Number is the line on which the connection is to be made.

The Timing Information specifies whether the call is to be timed or not.

The Charge Band specifies the costing to be applied to the call.

The Physical Address specifies the telephone number to be dialled.

The sequence of primitives for Physical connection establishment is shown below:



Note that there is no signalling shewn between the peer Physical entities. This is further discussed in section 3.2.5, but suffice it here to say that in both the call originator and call acceptor it is the Physical user that requests Physical connection; the Physical layer does not inform its user of a connection attempt by a distant Physical entity.

3.2.2.2.2 Physical Termination service.

The Physical termination service is provided to refuse Physical connection establishment or to terminate an already established Physical connection. The termination service is unilateral and abrupt, that is it may be requested by either Physical user at any time and cause the Physical connection to be terminated without regard for any data that may be pending transmission or in transit on the Physical connection.

There are two primitives associated with the P_DISCONNECT service:

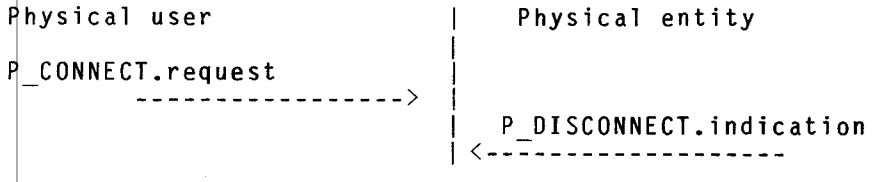
P_DISCONNECT.request and P_DISCONNECT.indication. The P_DISCONNECT.request primitive indicates to the PE that it should terminate the Physical connection. The P_DISCONNECT.indication primitive informs the Physical user that the Physical connection has been terminated.

```
P_DISCONNECT.request (
                        from U: Action
                      )

P_DISCONNECT.indication (
                         to U: Reason
                       )
```

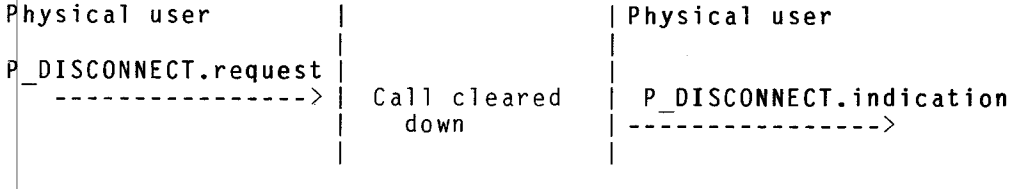
The Action parameter specifies whether the call is to be cleared down or not.

The Reason parameter indicates to the Physical user the reason for Physical connection termination.

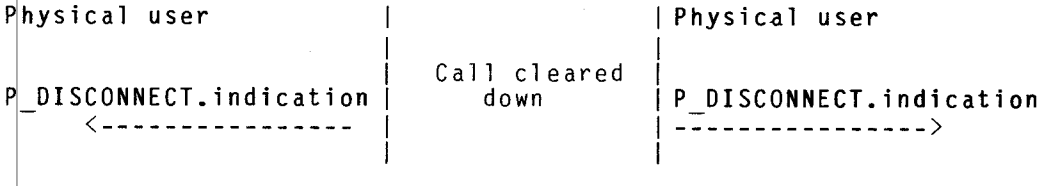
The sequence of primitives for Physical connection establishment refusal is as follows:



The sequence of primitives for user-initiated termination is as follows:



The sequence of primitives for layer initiated termination is as follows:



3.2.2.2.3 Physical Data Transfer Service.

The P_DATA service is provided to transfer data on an established Physical connection. There are two primitives associated with the P_DATA service: **P_DATA.request** and **P_DATA.indication**. The **P_DATA.request** conveys to the PE a pointer to the data to be transferred. The **P_DATA.indication** conveys to the Physical user a signal which indicates that some data has arrived.

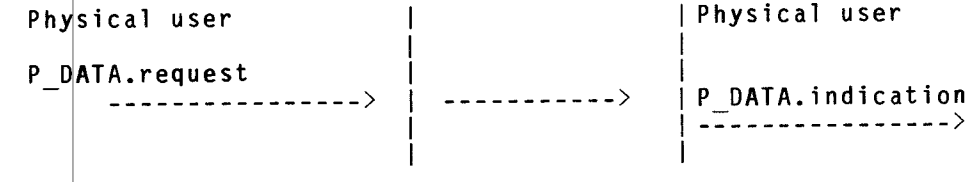
```
P_DATA.request (
                 from U: Pointer to Data for Transmission.
                 from U: Notification of Transmission.
                 from U: Amount of Data for Transmission.
               )

P_DATA.indication (
                  )
```

The Pointer to Data for Transmission is in the form of a Cell Tag (see s. 3.2.2.6.2).

The Notification of Transmission enables the Physical entity to notify an activity that the data has been transmitted.

The Amount of Data for Transmission refers to the data pointed to by the Pointer to Data for Transmission.

The sequence of primitives for the Physical data transfer service is as follows:



3.2.2.3 Sequence of Physical Services.

Physical services, described in the previous section, are logically ordered. Physical connection establishment precedes Physical data transfer and is followed by Physical termination.

3.2.2.4 Services required of the OPD hardware.

In order that the Physical layer may function it requires the interfaces and facilities defined in ref 5.

3.2.2.5 Physical Procedures.

3.2.2.5.1 Physical Connection Establishment.

The establishment of a Physical connection requires the exchange of **P_CONNECT.request** and **P_CONNECT.confirm** primitives between the PE and

the Physical user. There is no signalling between peer PEs except insofar as compliance with V25 auto-answering protocol is expected of the accepting station.

A Physical connection establishment attempt is initiated by a PE in response to the occurrence of a **P_CONNECT.request** at its local Physical user interface. The Physical entity actions the request by calls to the Telephone Handler (TH) (ref 5). If the response from TH the indicates that a data call has been successfully established, then PE will send a **P_CONNECT.confirm** to the Physical user. Otherwise a **P_DISCONNECT.indication** will be sent. Only one attempt will be made by PE to establish the call.

3.2.2.5.2 Physical Data Transfer

The transfer of data on an established Physical connection is initiated by the occurrence of a **P_DATA.request** at the local Physical user interface. PE will attempt to transmit the data to its peer entity by calls to TH. There is no provision for error checking, which is the proper concern of the Physical user. In the case of a half-duplex Physical connection, the Physical user need have no regard for the current line state (transmitting or receiving with respect to the local PE). PE will transmit the data as soon as it is able, which may be immediately (if no data is currently being received) or once data stops arriving and the distant PE turns the line round.

The arrival of data may be notified to the Physical user by the occurrence of a **P_DATA.indication** at the Physical user interface. The user may elect during Physical connection establishment to have the arrival of data notified or not.

3.2.2.5.3 Physical Connection Termination.

In order to terminate a Physical connection the PE instructs TH to end the call. The Physical user may initiate Physical connection termination by causing the occurrence of a **P_DISCONNECT.request** at its lower interface. There is no signalling between peer PEs: termination is immediate, and data pending transmission or delivery to the local Physical user will be lost.

A PE which is informed by TH that the call has been terminated causes the occurrence of a **P_DISCONNECT.indication** at its upper interface. Again, termination is immediate: data pending delivery to the local Physical user will be lost.

3.2.2.6 Using Physical Layer Services.

In order to use Physical layer services, an Application must make a request to TLink Manager (TLM) as described in section 3.2.1. The service type field of the **TLM_SERVICE.request** (see s. 4.2.2.1.1) is set either to 1 for unbuffered Physical service, or to 2 for buffered Physical service. In the case of buffered Physical service TLM will process requests for temporary data buffers as described in section 4.3. The buffers are not used by the PE.

TLM will attempt to start PE, and on doing so or failing to do so will send a **TLM_SERVICE.response** (see s. 4.2.1.1.1). If the status field of the message is non-zero TLM cannot provide the service requested and will close down. If the status field is zero, then the Physical service has been started. The Application must retain the Physical Buffer Id returned in the response for use during the data transfer phase of the Physical connection.

The Application must now open the pipes specified in the transmit and receive pipe name fields of the **TLM_SERVICE.response**, send to TLM an **APP_READY.indication** (see s. 4.2.2.1.2) and close pipes "L_TMAP_1" and "L_APTM_1". It may then commence use of the physical service as described in ss. 3.2.2.1 - 3.2.2.5. If the Application cannot open the pipes specified it should instead send an **APP_CLOSE.indication** (s. 4.2.2.1.3) before closing down.

3.2.2.6.1 Using the Physical connection establishment service.

The connection establishment service is described in ss. 3.2.2.2.1 and 3.2.2.5.1. The primitives involved are **P_CONNECT.request** (s. 4.2.4.3.1), **P_CONNECT.confirm** (s. 4.2.5.2.1) and **P_DISCONNECT.indication** (4.2.5.2.3).

The fields of the **P_CONNECT.request** are set as follows:

Length of dial string:

For the initiator side of a data call this is set to the number of characters to be dialled plus any pause characters. For example, the length of the dial string "01-980-4468" is $0B. The format of dial strings is defined in ref 5 s. 4.2.1.3.

On the acceptor side of a data call (one that has been data auto-answered) this should be set to any value other than 0. The message identifier differs for data call initiator and data call acceptor. A PE acting for a data call acceptor will not examine the dial string field.

On the initiator or acceptor side of a data-in-voice call this field should be set to 0.

Modem details:

These are set as defined in ref 10 s. 7.4.3.

Data receiver identity:

This is the activity identifier (ref 1 s. 5.3) of the Application. It is used by PE if the Application has requested that an event be caused on the arrival of data.

Receive event number:

This is set to the binary value (not the event mask) of the local event to be caused by PE on the arrival of data (thus local event 14 is encoded $0E). If it is set to 0 no event will be caused. It is used in conjunction with the data receiver identity.

Line Number:

This is the line number to be used when making a call or configuring a line to data. Its value may be 0 (don't care), 1 or 2. In the case of a data-in-voice call it must be the number of the line on which the call which is to carry data is already in progress. In the case of a data auto-answered call its value is unused.

Timing:

This is set to $0000 if the call is not to be timed, $0001 if timing is to be used, and $0002 if the timing default set for the line is to be used. It is not used if a call is data auto-answered.

Charge band:

This is a one or two character ASCII string defining the charge band to be used for call timing. It must be left justified, with binary zero fill, all zeroes if there is no charge band. For example, the charge band "UA" would be encoded $5541, the charge band "N" would be encoded $4E00.

Dial string:

This is a string of ASCII characters which will be passed to TH to be dialled. It may consist only of ASCII digits ($30 - $39), "*" ($2A), "#" ($CA) and the pause character "-" ($2D). It is not examined for data-in-voice calls or calls which have been data auto-answered.

Should the call fail, the fields of the **P_DISCONNECT.indication** will be set as described in the table in s. 4.2.5.2.3.

3.2.2.6.2 Using the Physical data transfer service.

The data transfer service is described in ss. 3.2.2.2.3 and 3.2.2.5.2. The primitives involved are **P_DATA.request** (s. 4.2.4.3.2) and **P_DATA.indication** (4.2.5.2.2).

The fields of the **P_DATA.request** are set as follows:

Cell Tag:

Data to be transmitted by PE must be held in a cell obtained from a cell allocator segment (ref 1 s. 8.4). The cell tag returned by the CREATE NEW CELL call (ref 1 s. 8.5.14) is passed to PE. PE will freeze the cell while transmitting data contained within it.

If the cell has been obtained from the TLM pool (see s. 4.3) PE will optionally return it to the pool on behalf of the Application. This it does if the event number field (below) is zero. If the event number is not zero PE will thaw the cell after transmission of data and cause the given event to the Application. It is assumed that the Application will then take charge of further use of the cell.

Event Number:

This is the number of the local event caused by PE when it has finished transmitting the data in the cell for the current **P_DATA.request**. If it is zero no event is caused.

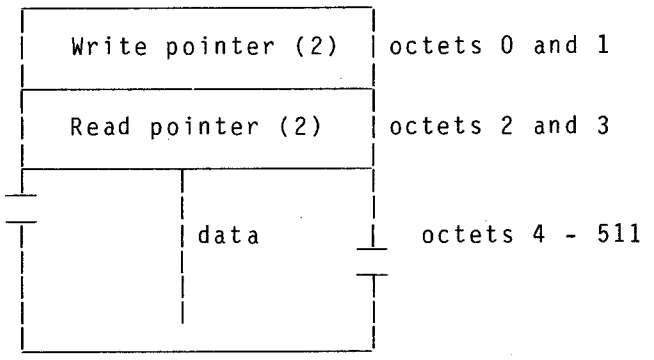
Activity Identity:

This is used in conjunction with the event number to signal a local event to the Application once the data for the current **P_DATA.request** has been transmitted. If the event number is 0 this field is ignored.

Length of Data:

This is the number of octets of data contained within the cell to be transmitted. PE makes no check on the size of the cell when transmitting, and commences from the first octet within the cell. The length is a word-long binary value, so a length of 255 octets would be encoded $00FF, a length of 65,535 would be encoded $FFFF.

The **P_DATA.indication** is not implemented as a pipe message, but as the changing of a pointer to a circular buffer and an optional local event. The sending of large amounts of data through a pipe is inefficient, so TLM allocates a frozen 512-octet buffer which can hold up to 508 octets of data in transit. The buffer is as follows:



The write pointer is maintained by PE, and points to the next octet into which received data will be placed. The read pointer is maintained by the Application and points to the next octet from which data will be extracted. Both the write pointer and the read pointer may have values 4..511 ($0004..$01FF). Their initial value is 4.

When the write pointer is equal to the read pointer, there is no new data for the Application to extract from the circular buffer, unless it has been 'lapped' by PE (PE has placed 498 or more octets of data in the buffer, none of which have been extracted by the Application). 'Lapping' is only avoided by regular checks on the read and write pointers, or by regularly processing the local events as described in the next paragraph. An Application should not alter the write pointer under any circumstances, and has sole control of the read pointer.

If a data receiver identity and local event were specified in the **P_CONNECT.request**, PE will cause the event to the Application on receipt of data. The frequency with which the event will be caused is undefined. Effectively, it is caused when data has been received and PE is in an idle loop. It will not be caused if no data has been received since last it was caused. However, an Application may already have extracted from the circular buffer the data whose arrival an event announces. It is advised, therefore, that the Application test the read and write pointer for equality on detecting a local event which heralds the arrival of new data.

3.2.2.6.3 Using the Physical termination service.

The Physical termination service is described in ss. 3.2.2.2.2 and 3.2.2.5.3. The primitives involved are **P_DISCONNECT.request** (s. 4.2.4.3.3) and **P_DISCONNECT.indication** (s. 4.2.5.2.3).

The Reason code field of the **P_DISCONNECT.indication** is set to a value as indicated in the table in s. 4.2.5.2.3.

3.2.3 Link Layer

The TLink Link Layer is fully described in chapter 5 of ref 3, and the reader is assumed to be familiar with that document. This section describes the OPD implementation of the TLink Link Layer by defining the subset implemented, the use of the Link Protocol Data Unit fields and the implementation of the Link primitives.

3.2.3.1 Implementation subset of TLink Link Layer.

The implementation of the Link Layer provides all the services and facilities specified in chapter 5 of ref 3 with the following restrictions:

The window size will be 1.

No Link Attention services are supplied.

Stream service type is not supported.

The default Link serial number is used.

Only Service Class 1 (asynchronous, half-duplex with BSC framing and octet-stuffing for data transparency) is supported.

3.2.3.2 Use of Link Protocol Data Unit Parameters.

This section describes the use made of the Link Protocol Data Unit (LPDU) parameters. The LPDUs are described in Appendix A of ref 3, and the parameters are only described in the following sections where further explanation is required.

3.2.3.2.1 Link Request LPDU.

Serial number:

This is always set to the default serial number as described in appendix B of ref 3. The serial number field of an incoming LR LPDU is checked for correct formation.

Secondary Address:

This is set to the value passed in the **L_CONNECT.request** dial string following the dial digits. It is optional, and if no secondary address is specified it will be omitted from the LR LPDU. It is ignored in an incoming LR LPDU.

3.2.3.2.3 Link Disconnect LPDU.

User reason:

This contains the octet in the data field of the **L_DISCONNECT.request**. It is extracted from an incoming LD LPDU and placed in the data field of the **L_DISCONNECT.indication**.

3.2.3.3 Using Link Layer Services.

In order to use link layer services, an Application must make a request to TLink Manager (TLM) as described in section 3.2.1. The service type field of the **TLM_SERVICE.request** (see s. 4.2.2.1.1) is set to 2 for Link services.

TLM will attempt to start PE and LE, and on doing so or failing to do so will send a **TLM_SERVICE.response** (see s. 4.2.1.1.1). If the status field of the message is non-zero TLM cannot provide the service requested and will close down. If the status field is zero, then the Link service has been started.

The Application must now open the pipes specified in the transmit and receive pipe name fields of the **TLM_SERVICE.response**, send to TLM an **APP_READY.indication** (see s. 4.2.2.1.2) and close pipes "L_TMAP_1" and "L_APTM_1". It may then commence use of the link service as described in chapter 5 of ref 3. If the Application cannot open the pipes specified it should instead send an **APP_CLOSE.indication** (s. 4.2.1.2.1.3) before closing down.

3.2.3.3.1 Using the Link connection establishment service.

The Link connection establishment service is described in sections 5.2.1 and 5.5.1 of ref 3. The primitives involved are **L_CONNECT.request** (s. 4.2.3.3.1), **L_CONNECT.confirm** (s. 4.2.4.2.2), **L_CONNECT.indication** (s. 4.2.4.2.1) and **L_CONNECT.response** (s.4.2.3.3.2).

The fields of the **L_CONNECT.request** are set as follows:

Inactivity indicator:

This is the maximum number of seconds for which LE will maintain the Link open without any activity taking place (no data transmitted or received). If specified to be less than 59 seconds ($3B) the default time of 59 seconds will be used.

Service level:

This is set to 1 for minimum link services. It is placed in the Protocol level indicator field of the LR LPDU.

Service type:

Set to 1 for block mode. It is used to specify a Maximum User Data Size of 260 in the LR LPDU.

Dial String:

May either be as described in section 3.2.2.6.1, or may have an additional "extension" field. The extension field is separated from the dialled digits by a "X" character ($58), and consists of a string of octets (normally ASCII digits) which will be placed in the Secondary Address parameter of the LR LPDU. Note that the dial string length must include the length of the extension field and separator if present.

Optionally, the first character of the dial string may be an Evade Police Character (EPC). The presence of this character in the dial string will prevent a check being made in the Bad Number List for the presence of the dial digits. The EPC may be used if a call is manually originated. Because of its use in circumventing checks for redial attempts, its value will not be published, but will be available to bona fide software developers on application to OPD Business Centre. Note that the dial string length must include the EPC if present.

Line number, modem details, dial string length, timing, and charge band are as described in section 3.2.2.6.1.

The fields of the **L_CONNECT.confirm** are set as follows:

Service level:

Only minimal link services are supported, so this will always be set to 1.

Service type:

Only block mode working is supported, so this value is always set to 1. This indicates a maximum user data size of 260 octets.

The Service level and Service type fields of the **L_CONNECT.indication** are set as described for those of the **L_CONNECT.confirm.**

The Service level and Service type fields of the **L_CONNECT.response** are set as described for those of the **L_CONNECT.confirm.** The inactivity indicator field is set as described for that of the **L_CONNECT.request.**

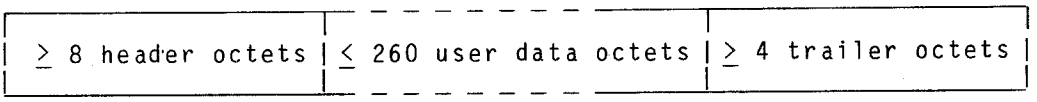
3.2.3.3.2 Using the Link data transfer service.

The Link data transfer service is described in sections 5.2.3 and 5.5.2 of ref 3. The primitives involved are **L_DATA.request** (4.2.3.3.4) and **L_DATA.indication** (s.4.2.4.2.4).

The Activity identity, cell tag, length of data and event number fields of the **L_DATA.request** are set as as in the **P_DATA.request** (see s. 3.2.2.6.2). In addition:

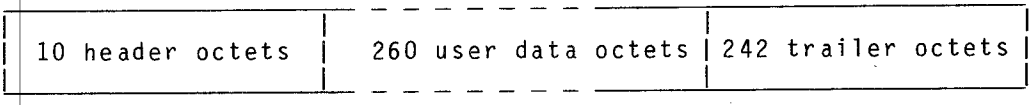
Offset:

This is the offset from the start of the cell at which the data to be transmitted begins. It must be at least eight. In addition there must be at least four unused octets between the end of the user data and the end of the cell. LE uses those octets to insert protocol-specific data before transmission. The structure of the cell is thus:



The data length is the number of octets of user data, and does not include either the header or trailer octets.

It is important that the offset and length of data are correctly understood. As an example, if the cell identified by the cell tag were constructed as follows:



the offset would be set to $0A and the length of data to $0104.

The fields of the **L_DATA.indication** are set as follows:

Cell tag

Data received from line is placed in a cell obtained from the TLM pool. Once the Application has extracted or finished with the data in the cell, it should return it to the pool as described in section 4.3. Failure to do so will result in the pool becoming exhausted and this will result in abrupt Link termination.

Length of data.

This is the number of octets of data within the cell which are being delivered to the Application. The cell will contain other, random, data which the Application should ignore.

Offset

This is the offset from the start of the cell at which the data

being delivered to the Application begins. It may have any value greater than or equal to 0. A value of 0 indicates that the Application should take its data from the first octet of the cell.

3.2.3.3.3 Using the Link termination service.

The Link termination service is described in sections 5.2.2 and 5.5.4 of ref 3. The primitives involved are **L_DISCONNECT.request** (s. 4.2.3.3.5) and **L_DISCONNECT.indication** (s. 4.2.4.2.3).

The fields of the **L_DISCONNECT.request** are set as follows:

Data:

This is passed transparently to the peer Link user entity.

The fields of the **L_DISCONNECT.indication** are set as follows:

Reason:

This field indicates whether the disconnection was initiated by the peer Link-user or in the local Link or Physical Entities. The values used are listed in section 4.2.4.2.3

Data:

This field contains the data passed from the peer Link user in the case of a user-initiated disconnection.

3.2.4 Session Layer.

The TLink Session layer is fully described in chapter 6 of ref 3 and the reader is assumed to be familiar with that document. This section describes the OPD implementation of the TLink Session layer by defining the subset implemented, the use of the Session Protocol Data Unit fields and the implementation of Session primitives.

3.2.4.1 Implementation subset of TLink Session Layer.

The implementation of the Session Layer provides all the services and facilities specified in chapter 6 of ref 3 with the following restrictions:

No Session Attention Services are supplied.

3.2.4.2 Use of Session Protocol Data Unit Parameters.

This section describes the use made of the Session Protocol Data Unit (SPDU) parameters. The SPDUs are as described in Appendix A of ref 3, and the parameters are only described in the following sections where further explanation is required.

3.2.4.2.1 Session Request SPDU.

System Type:

Will always be set to 7 (assigned by Microcom Inc for OPD) in outgoing SR SPDUs.

File System Type:

The value passed to the Session Entity in the **S_CONNECT.request** is used in outgoing SR SPDUs. The value will normally be 8 (assigned by Microcom for OPD Kernel), but Applications which emulate other file system types may use other values as defined in Appendix A of ref 3.

This is extracted from incoming SR SPDUs and passed to the Session user in **S_CONNECT.indication** and **S_CONNECT.confirm** primitives.

Source Id:

The value passed in the **S_CONNECT.request** primitive is used in outgoing SR SPDUs. This is extracted from incoming SR SPDUs and passed to the Session user in the **S_CONNECT.indication** and **S_CONNECT.confirm** primitives.

Source Address:

The value passed in the S_CONNECT.request primitive is used in outgoing SR SPDUs. This is extracted from incoming SR SPDUs and passed to the Session user in the S_CONNECT.indication and S_CONNECT.confirm primitives.

Destination Id:

The value passed in the S_CONNECT.request is used. This is the name of the Application in the distant OPD to which communications are to be established. Its value is extracted from an incoming SR SPDU and used to load the Application (see s. 3.2.5).

Destination Address:

Not used.

Session Optimisation:

If the User type in the S_CONNECT.request is 1 (MNP user), this value is set to 1 (data SPDU not used). Otherwise this will be set to 0 (no optimisation).

3.2.4.2.2 Session Close SPDU

Reason:

This is taken from the data field of a S_CLOSE.request message. It is extracted from an incoming SC SPDU and placed in data field of the S_CLOSE.indication.

3.2.4.3 Using Session Layer Services.

In order to use Session layer services, an Application must make a request to TLink Manager (TLM) as described in section 3.2.1. The service type field of the TLM_SERVICE.request (see s. 4.2.2.1.1) is set to 4 for Session services.

TLM will attempt to start PE LE and SE, and on doing so or failing to do so will send a TLM_SERVICE.response (see s. 4.2.1.1.1). If the status field of the message is non-zero TLM cannot provide the service requested and will close down. If the status field is zero, then the Session service has been started.

The Application must now open the pipes specified in the transmit and receive pipe name fields of the TLM_SERVICE.response, send to TLM an APP_READY.indication (see s. 4.2.2.1.2) and close pipes "L_TMAP_1" and "L_APTM_1". It may then commence use of the Session service as described in chapter 6 of ref 3. If the Application cannot open the pipes specified it should instead send an APP_CLOSE.indication (s. 4.2.2.1.3).

3.2.4.3.1 Using the Session Connection Establishment Service.

The Session connection establishment service is described in sections 6.3.1 and 6.5.1 of ref 3. The primitives involved are **S_CONNECT.request** (s. 4.2.2.2.1), **S_CONNECT.confirm** (s. 4.2.3.2.2), **S_CONNECT.indication** (s. 4.2.3.2.1) and **S_CONNECT.response** (s. 4.2.2.2.2).

The fields of the **S_CONNECT.request** are set as follows:

Line number, modem details, dial string length, timing, charge band and dial string are as described in s. 3.2.2.6.1.

Inactivity indicator is as described in s. 3.2.3.3.1.

Attention:

Set to 0 to enable Session attention services (which are not supported), non-zero to disable them. Must be set non-zero.

File System Type:

This will normally have the value $08 (assigned by Microcom to OPD Kernel). However, an Application which emulates another file system may use any of the permitted values. These are listed in Appendix A of ref 3.

Source Identity:

This is an octet string, preceded by its length encoded as a binary value. The length does not include the length octet. For example:

$06 53 74 72 69 6E 67

This is passed transparently to the peer Session user.

Source Address:

This is an octet string, preceded by its length encoded as a binary value. The length does not include the length octet. For example:

$06 53 74 72 69 6E 67

This is passed transparently to the peer Session user.

Destination Identity:

This is an octet string, preceded by its length encoded as a binary value. The length does not include the length octet.

For example:

$0C 7E 20 54 52 41 4E 53 46 45 52 20 20

represents the string " TRANSFER ".

User Type:

If the Session User is using TLink File Transfer PDUs only, this is set to $01. Otherwise it is set to $02.

The fields of the **S_CONNECT.confirm** are set as follows:

Destination System identity:

Describes the type of computer on which the peer Session entity is running. The value $07 indicates ICL OPD. Other values are as described in Appendix A of ref 3.

Line number:

The number of the line (1 or 2) on which the call has been made.

Destination File System Type:

The file system used on the distant machine. The OPD Kernel file system has the value $08. Other values are given in Appendix A of ref 3.

The fields of the **S_CONNECT.indication** and **S_CONNECT.response** are as described for **S_CONNECT.request** and **S_CONNECT.confirm**.

3.2.4.3.2 Using the Session Data Transfer Service.

The Session Data Transfer service is described in sections 6.3.4 and 6.5.2 or ref 3. The primitives involved are **S_DATA.request** (s. 4.2.2.2.5) and **S_DATA.indication** (s. 4.2.3.2.5).

The fields of the **S_DATA.request** are as described for the **L_DATA.request** (s. 3.2.3.3.2), except that the minimum value for the offset is 10, not 8.

The fields of the **S_DATA.indication** are as described for the **L_DATA.indication** (s. 3.2.3.3.2).

3.2.4.3.3 Using the Session Graceful Termination service.

The Session Graceful Termination service is described in sections 6.3.2 and 6.5.4 of ref 3. The primitives involved are **S_CLOSE.request** (s. 4.2.2.2.4) and **S_CLOSE.indication** (s. 4.2.3.2.4).

The fields of the **S_CLOSE.request** are set as follows:

Data:

This is passed transparently to the peer Session user.

The fields of the **S_CLOSE.indication** are set as follows:

Data:

This is passed transparently from the peer Session user.

3.2.4.3.4 Using the Session Abrupt Termination Service.

The Session abrupt termination service is described in sections 6.3.3 and 6.5.5 of ref 3. The primitives involved are **S_DISCONNECT.request** (s. 4.2.2.2.3) and **S_DISCONNECT.indication** (s. 4.2.3.2.3).

The fields of the **S_DISCONNECT.request** are set as follows:

Data:

This is passed transparently to the peer Session user.

The fields of the **S_DISCONNECT.indication** are set as follows:

Reason:

Indicates whether the cause of the disconnection was detected locally or remotely.

Data:

If the disconnection was initiated by the distant Session user, this contains the data placed in the Data field of the **S_DISCONNECT.request**. Otherwise it is undefined.

3.2.5 Answering a data call.

The OPD user may specify that incoming data calls are to be auto-answered by TLink. This is achieved by setting the auto-answer protocol to "TLINK" (in any combination of upper- and lower-case letters) (ref 11 s. 13.1.6). Alternatively an application may wish to cause TLink to answer a call when a voice call is switched to data.

If the call is to be auto-answered :-

the interface between TLM and the Telephone Handler for auto-answering is described in ref 9. Suffice it here to say that the modem configuration will be taken from CMOS record 2. TLink Physical, Link and Session entities will be started. Link connection will complete and the Link entity will enter the Link data transfer phase, the Session entity will commence Session connection establishment and wait for the arrival of an SR SPDU.

If the call is to be data-in-voice :-

the application causes events 9 and 13 to TLM. The modem is configured by looking at CMOS. A TLink SESSION level service is started. The reliable link will establish and the session entity will await the arrival of an SR SPDU.

When an SR SPDU arrives, either for a data or data-in-voice call, the contents of the Destination ID field will be extracted and used to start an Application using LOAD PROGRAM (ref 2 s. 4.9.3). If the Destination ID field is empty or omitted then TLM will use the default as set up in CMOS. If this default is absent then the call is failed.

TLM will obtain the Application's Activity Identity from the LOAD PROGRAM call. Not sooner than 1s and not later than 2s after detecting that the Application has been loaded successfully, TLM will cause local event 14 to the Application and open pipes "L_TMAP_1" and "L_APTM_1" and send an **APP_SERVICE.indication** (s. 4.2.1.1.2) to the Application through pipe "L_TMAP_1". This message contains the names of the pipes the Application should open to communicate with the Session entity. It is important that the Application requests notification of local event 14 immediately after being loaded. If TLM causes the local event to the Application before the Application has requested notification of it, the event will be lost.

The Application now opens the pipes named in the **APP_SERVICE.indication** and sends an **APP_READY.indication** (s. 4.2.2.1.2) to TLM indicating that it will accept the call. It then closes pipes "L_TMAP_1" and "L_APTM_1" and commences its interaction with the Session entity. The first event to occur across the Session/Application boundary will be the occurrence of a **S_CONNECT.indication** sent by the Session entity informing the Application of the connection attempt.

If the Application fails to open the pipes to the Session entity, or cannot proceed for any other reason, it should send an **APP_CLOSE.indication** to TLM through pipe L_APTM_1.

The sequence of events described above may be summarised:

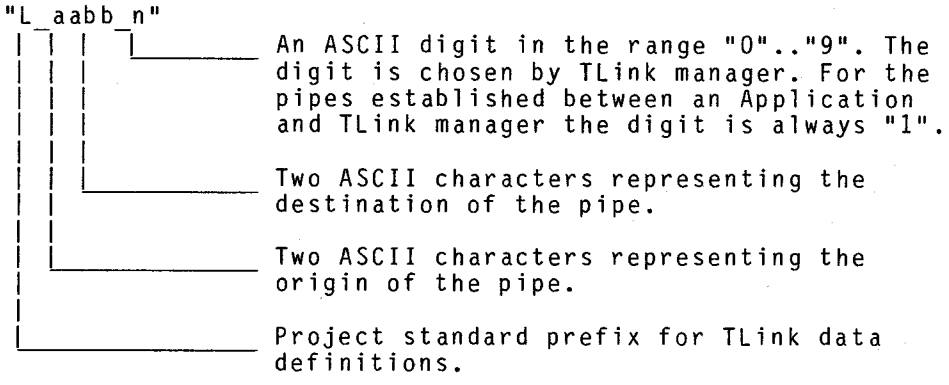
TLink Manager	Application
To answer data-in-voice, cause local events 9 and 13 to TLM.	
To auto-answer the data call starts Physical, Link and Session entities. Obtains the name of the requested Application from Session.	
LOAD PROGRAM	
Open "L_TMAP_1" and "L_APTM_1"	
APP_SERVICE.indication ---------->	
Signal local event 20 to Application.	
	Open "L_APTM_1" and "L_TMAP_1"
	Open pipes to Session.
	APP_READY.indication <----------
	Close "L_APTM_1" and "L_TMAP_1"
	Await **S_CONNECT.indication**

4 INTERFACES

The interfaces are described in this section in three categories. First the names of the pipes established between layer entities, TLink manager and Applications are listed, followed by definitions of the primitives which are passed as messages through the pipes. There follows a section on memory management of the temporary store used to hold data received or awaiting transmission.

4.1 Naming of pipes.

The pipes which are established between layer entities, TLink Manager and Applications have a naming convention consistent with both OPD project naming standards and ease of implementation. They are constructed as follows:



4.1.1 Summary of Pipe names.

In the summary below the abbreviations used are as follows:

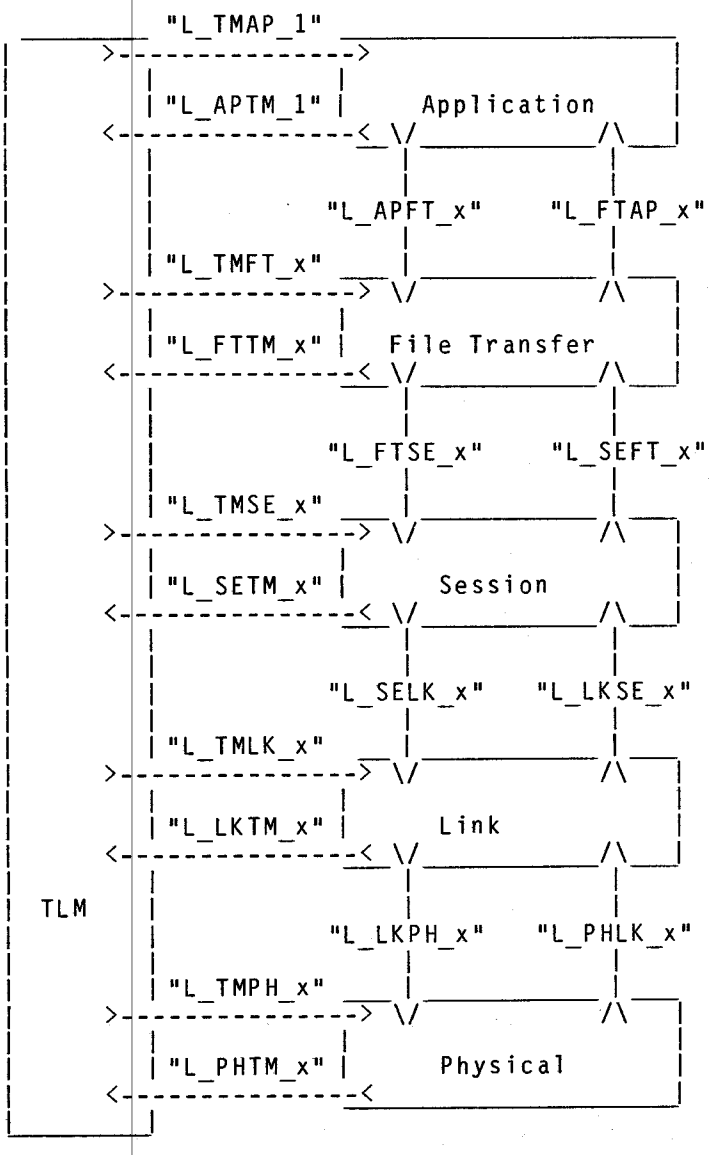
LE Link Entity
PE Physical Entity
SE Session Entity
FTE File Transfer Entity
TLM TLink Manager

The character "x" represents an ASCII digit in the range "0".."9".

Name	From	To
"L_TMPH_x"	TLM	PE
"L_PHTM_x"	PE	TLM
"L_TMLK_x"	TLM	LE
"L_LKTM_x"	LE	TLM

"L_TMSE_x"	TLM	SE
"L_SETM_x"	SE	TLM
"L_TMFT_x"	TLM	FTE
"L_FTTM_x"	FTE	TLM
"L_TMAP_1"	TLM	Application
"L_APTM_1"	Application	TLM
"L_PHLK_x"	PE	PE user
"L_LKPH_x"	PE user	PE
"L_LKSE_x"	LE	LE user
"L_SELK_x"	LE user	LE
"L_SEFT_x"	SE	SE user
"L_FTSE_x"	SE user	SE
"L_FTAP_x"	FTE	FTE user
"L_APFT_x"	FTE user	FTE

The following diagram illustrates the pipe names used. The character "x" represents some ASCII character. Although a File Transfer Entity is shewn in the diagram it is not provided at release 1.



4.2 Definition of Layer primitives.

The primitives defined in this section are the implementations of the primitives defined in the Microcom specification (ref 3) and in this document. Each primitive begins with a length octet, which is the length of the primitive, not including the length octet. All primitives so far defined must have a length of 255 octets or less.

Implementors are advised to extract the first octet of a primitive from the pipe with a call to GET BYTE IMMEDIATE (ref 1 s. 9.7.3). This will provide the length parameter to a subsequent call of QUEUED GET STRING (ref 1 s. 9.7.5) to obtain the rest of the primitive.

The message identifiers are summarised below:

Identifier	Primitive	From	To
$01	S_START.request	TLM	SE
	L_START.request	TLM	LE
	P_START.request	TLM	PE
	S_CONNECT.request	FT	SE
	L_CONNECT.request	SE	LE
	P_CONNECT.confirm	PE	LE
$02	TLM_SWITCH.request	TLM	SE
	"	TLM	LE
	"	TLM	PE
	S_CONNECT.indication	SE	FT
	L_CONNECT.indication	LE	SE
	P_DISCONNECT.indication	PE	LE
$03	S_CONNECT.response	SE	FT
	L_CONNECT.response	LE	SE
	APP_SERVICE.indication	TLM	APP
$04	S_CONNECT.confirm	SE	FT
	L_CONNECT.confirm	LE	SE
	TLM_SERVICE.response	TLM	APP
$05	S_DISCONNECT.request	FT	SE
	L_DISCONNECT.request	SE	LE
$06	S_DISCONNECT.indication	SE	FT
	L_DISCONNECT.indication	LE	SE
$07	S_CLOSE.request	FT	SE
$08	S_CLOSE.indication	SE	FT
$09	S_DATA.request	FT	SE
	L_DATA.request	SE	LE
$0A	S_DATA.indication	SE	FT

	L_DATA.indication	LE	SE
$21	TLM_CLOSE.indication	PE	TLM
$23	APP_START.response	TLM	SE
	APP_START.request	SE	TLM
$31	P_CONNECT.request (initiator)	LE	PE
$32	P_DATA.request	LE	PE
$33	TLM_CLOSE.indication	LE	TLM
$41	TLM_CLOSE.indication	SE	TLM
$61	APP_CLOSE.indication		
$62	TLM_SERVICE.request	APP	TLM
$72	S_START.confirm	SE	TLM
	L_START.confirm	LE	TLM
	P_START.confirm	PE	TLM
$73	P_DISCONNECT.request	LE	PE
$B1	P_CONNECT.request (acceptor)	LE	PE

In the following sections the messages themselves are listed by origin, that is, all the messages originating from the TLink Manager are listed together, all those originating from the Application are listed together and so on. Within each of these categories, all messages destined for TLink manager are listed together, and so on.

Note that the naming of the message categories below assumes that an Application uses Session services, so that the Session, Link and Physical entities are all active. An Application using Link services would have to use the messages defined between the Session and the Link entities; an Application using Physical services would have to use the messages defined between Link and Physical Entities.

Fields marked with an asterisk are checked for correctness. If the values are incorrect the layer entity detecting the error will close down.

All fields are one octet long, excepting those which are marked with a bracketed number. The number indicates the length, in octets, of the field. Variable length fields are described in the accompanying text.

4.2.1 Messages originating from the TLink Manager.

4.2.1.1 Messages destined for the Application

4.2.1.1.1 **TLM_SERVICE.response**

message length	
$04	
Physical buffer id (4)	See note (2)
status	See note (1)
Transmit pipe name (8)	See note (3)
Receive pipe name (8)	See note (4)

Notes:

(1) The status values are:

0 Service started OK.
Any other value indicates failure.

(2) This is the address of the circular receive buffer used by the Physical Entity. It is of interest only to an Application that has requested physical services.

(3) This is the name of the pipe through which the Application sends messages to the requested service

(4) This is the name of the pipe through which the Application receives messages from the requested service.

4.2.1.1.2 APP_CLOSE.indication

message length
$61

This message is used if CMOS RAM is corrupt rather than a TLM_SERVICE.response. It should be catered for and treated as a negative TLM_SERVICE.response meaning "CMOS RAM corrupt".

4.2.1.1.3 APP_SERVICE.indication

message length
$03
Physical buffer id (4)
Not used
Transmit pipe name (8)
Receive pipe name (8)

See the explanatory notes for **TLM_SERVICE.response** (s. 4.2.1.1.1)

4.2.1.2 Messages destined for the Session Layer.

4.2.1.2.1 S_START.request

message length
$01
command subtype

- 1 = initiator 2 = acceptor

4.2.1.2.2 TLM_SWITCH.request

message length
$02
buffer address (4)

Address of buffer used to store data recieved from modem.

4.2.1.2.3 APP_START.response

length of message
$23
response code

- 0 = OK Anything else = fail

4.2.1.3 Messages destined for the Link Layer.

4.2.1.3.1 **L_START.request**

message length	
$01	
command subtype	- see note 1
modem details (2)	- see note (2)

Notes :

(1) Await L_CONNECT.request (initiator). Value = 1
Await L_CONNECT.request (D-I-V initiator) 2
Await LR_LPDU (acceptor). 3
Await LR_LPDU (D-I-V acceptor) 4

(2) As required. A default setting is contained in CMOS.

4.2.1.3.2 **TLM_SWITCH.request**

message length	
$02	
buffer address (4)	- address of buffer used to store data received from the modem.

4.2.1.4 Messages destined for the Physical Layer.

4.2.1.4.1 P_START.request

Length of message	- not including length-of-message octet
$01	- message identifier
Command Subtype	- await P_CONNECT.request from user. current value = 1

4.2.1.4.2 TLM_SWITCH.request

Length of message	- not including length-of-message octet
$02	- message identifier
buffer address(4)	- address of buffer in which to place incoming data. See note 1

Notes :

(1) Buffer assumed permanently frozen

4.2.2 Messages originating from the Application.

4.2.2.1 Messages destined for the TLink Manager

4.2.2.1.1 TLM_SERVICE.request

message length	
$62	
Activity identifier (4)	See note (2)
Service required	See note (1)

Notes:

(1) Service codes are as follows:

1 Physical - no buffering
2 Physical - buffered
3 Link
4 Session

(2) This is used by TLM to signal events to the Application.

4.2.2.1.2 APP_READY.indication

message length	this interface responds positively to a TLM_SERVICE.response
$63	

4.2.2.1.3 APP_CLOSE.indication

message length	this interface responds negatively to a TLM_SERVICE.response
$61	

4.2.2.2 Messages destined for the Session Layer.

4.2.2.2.1 S_CONNECT.request

message length	
$01	
line number	- As required. 0,1 or 2. Mandatory for data-in-voice calls
modem details (2)	- see note (1)
dial string length (2)	- if zero, a data-in-voice call
timing (2)	- As required. Ignored for data-in-voice calls
charge band (2)	- As required. Ignored for data-in-voice calls
inactivity indicator	- See note (3)
attention *	- enable/disable 0/non zero Must be "disable"
file system type	- As required. See note (4)
source identity	- See note (7)
source address	- See note (7)
destination identity	- destination application name and enhancement level (note (7))
destination session address	- See note (7)
user type *	- MNP/other 1/2 See note (2)
dial string	- variable length. Undefined if data-in-voice call

Notes :

(1) As required. Reference 10 section 7.4.3 describes modem control bytes. A default is set in CMOS.

(2) MNP enables session data phase optimisation. See MNP spec (ref 3) for further details. This may be used if the Session user is passing only MNP File Transfer PDUs in **S_DATA.requests.**

(3) Minimum value of 59 seconds used. Any value less than 59 will be coerced to 59s. any See ref 3 s.5.5.5.6 for explanation of inactivity timeouts.

(4) Values assigned to file system types are listed in Appendix A of ref 3. The value for OPD Kernel is 8.

(5) The line number, timing and charge band parameters are described in ref 5 section 4.

(6) If this call is used for data-in-voice conversion then the line number must obviously not be zero.

(7) Each of these parameters consists of

- an octet containing the length, in octets, of the rest of the parameter.
- any number of octets making up the parameter value.

Zero length parameters are permissible, ie parameter length octet = zero and parameter value = null.

Thus each of these fields is at least one octet long.

(8) The maximum length of this message is 254 octets, not including the message-length octet.

4.2.2.2.2 S_CONNECT.response

message length	
$03	
attention *	- enable/disable 0/non-zero Must be set to disable
inactivity indicator	- Null. Value undefined. See note (1)
file system type	- See s. 4.2.2.2.1 note (4)

Notes

(1) The reason for this parameter is unknown as the acceptors´inactivity timer is specified during the reliable link establishment.

4.2.2.2.3 S_DISCONNECT.request

message length	
$05	
data	- this is passed on to the peer session-user

4.2.2.2.4 S_CLOSE.request

message length	
$07	
data	- passed transparently to the peer session user
data	- ditto

4.2.2.2.5 S_DATA.request

message length	
$09	
cell tag (2)	- where data to be transmitted is. See note (1)
activity identity (4)	- identity of session user. See note (2)
length of data (2)	- total length of data to be transmitted, in octets
event no	- optional event to be caused See note (2)
offset	- offset, within cell, of start of data (in octets)

Notes

(1) The cell indicated must have least ten octets reserved at the start of the cell and four octets reserved at the end of the data. These octets are written by TLink.

The length-of-data parameter does not include these reserved octets.

(2) If a the event number field is zero then TLink will free the cell (s. 4.3.1), the activity identity is not

required in this case. If the event number is not zero then the event is caused to the activity specified in the activity identity field.

(3) Before transmitting the data TLink freezes the associated segment. After transmitting the data the segment is thawed before any associated event is caused.

4.2.3 Messages originating from the Session Layer.

4.2.3.1 Messages destined for the TLink Manager.

4.2.3.1.1 S_START.confirm

message length	
$72	
response code	- see note 1

Notes :

(1) Value zero = OK
Any other value is a fail indication and reason code.
The reason codes are as defined in s. 4.2.3.2.3.

4.2.3.1.2 TLM_CLOSE.indication

message length	
$41	
reason code	- reason for close. See note 1

Notes :

(1) This parameter is not currently used but is reserved for future expansions.

4.2.3.1.3 APP_START.request

length
$23
application name (12) - see note (1)

(1) If the destination identity field in the SR SPDU is used then this is the contents of that field. It is truncated or padded with trailing spaces as necessary to bring it to exactly 12 characters. If the destination identity field is missing then this is the default application name held in CMOS.

4.2.3.2 Messages destined for the Application

4.2.3.2.1 S_CONNECT.indication

message length	
$02	
source system type	- See note (1)
source file system type	- See section 4.2.2.2.1 note (4)
line number	- 1 or 2 . Telephone line in use
source identity	- See note (2)
source address	- See note (2)
attention	- enable/disable 0/non-zero Will be set to disable.

Notes :

(1) Values assigned to system type are listed in Appendix A of ref 3. The value assigned to OPD is 7.

(2) These are the strings passed to TLink in a CONNECT.request primitive by the call originator. See 4.2.2.2.1 for example.

4.2.3.2.2 S_CONNECT.confirm

message length	
$04	
attention	- enable/disable 0/non-zero. Will be set to disable
destination system identity	- See section 4.2.3.2.1 note (1)
line number	- telephone line in use 1/2
Dest. file system type	- See section 4.2.2.2.1 note (4)
Source identity ()	- See section 4.2.2.2.1 note (7)
Source address ()	- See section 4.2.2.2.1 note (7)

4.2.3.2.3 S_DISCONNECT.indication

message length	
$06	
reason	- reason for disconnection. See note 1
data	- data passed from peer session user. See note 2

Notes

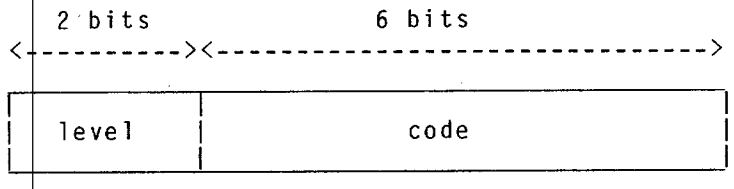
(1) This indicates whether the disconnection was induced by the peer session user (value $FF) or by the co-located session layer. In the latter case the value of this field is the reason for

disconnection and these values are given in the attached table.

(2) If the disconnection was induced by the co-located session layer this field is undefined. Unless the remote application is known, be cautious about assigning meanings to this parameter if the disconnection was induced remotely.

(3) When this message is received then the session has been destroyed ie no further communication with the remote end is possible. TLink has suicided.

The disconnection reason code consists of two fields as shown :-



The level indicates whereabouts in TLink the fault occurred.

0 at the physical layer
1 at the link layer
2 at the session layer
3 reserved and undefined

So, for example :

If SESSION failed to negotiate parameters the disconnection reason code would be hexadecimal 83 :-

1 0 0 0 0 0 1 1

code = 3

level = 2

A table of error codes as generated by the SESSION layer follows :-

Value	Meaning
1	protocol violation - unrecognisable message or illegal parameter
2	default application not found
3	parameter negotiation failed
4	software fault - internal inconsistencies.
5	hardware fault - unrecoverable.
6	modem not connected to line
7	SR SPDU transmitted twice without reply SR SPDU
8	unable to start the application identified in the SR SPDU
9	can't get in contact with user
10	illegal cell tag in L DATA.ind
11	unable to get TLM buffer for SR
12	illegal cell tag in S DATA.request
13	error when thawing TLM buffers
14	unable to start SR retransmission timer
15	unable to open SR timer channel

16	unable to close SR timer channel
17	error upon sending message to LINK layer
18	error upon receiving message from LINK layer
19	destination application timed out

4.2.3.2.4 S_CLOSE.indication

message length	
$08	
data	- as passed from peer session user
data	- ditto

4.2.3.2.5 S_DATA.indication

message length	
$0A	
cell tag (2)	- location of the data. See note (1)
length of data (2)	- total length of data, in octets
offset	- offset, within the cell, of where the data starts

Note

(1) The segment containing the indicated cell is not frozen. It is the responsibility of the user to free the buffer after use (see section 4.3.2)

4.2.3.3 Messages destined for the Link Layer.

4.2.3.3.1 **L_CONNECT.request**

message length	
$01	
inactivity indicator	- see note (1)
service level *	- minimum/standard 1/2 Must be set to "minimum"
service type *	- block/stream mode 1/2
line number	as required. 0,1 or 2. See - note 3
modem details (2)	as required. See ref 10 s. 7.4.3 for details
dial string length (2)	- if zero, a data-in-voice call
timing (2)	- as required. Ignored if a data-in-voice call
charge band (2)	- as required. Ignored if a data-in-voice call undefined and unexamined for
dial string	- data-in-voice call. See note 4

Notes :

(1) Zero to 58 means that the minimum value of the inactivity timer is to be used (59 seconds). Any other value is the timeout, in seconds, to be set.

NB. The inactivity timer, as the name suggests, protects against the link staying open indefinitely. If it matures the link is closed.

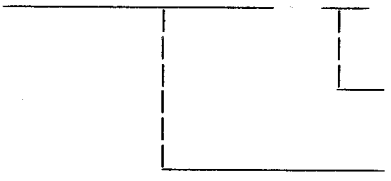
(2) The timing and charge-band parameters are defined in the ref 5.

(3) If this call is used to switch the line from voice to data, ie a data-in-voice call, then the line number must be explicitly set up. See ref 10.

(4) An "extension" number may be specified in the dial string. This "extension" will not be used for auto-dialling but will be used as the secondary address parameter in the LR LPDU. The "extension" is separated from the rest of the dial string by an "X".

eg.

0 1 2 3 4 5 6 7 8 X 9 9



this goes into the LR secondary address parameter

this is used to auto-dial

4.2.3.3.2 **L_CONNECT.response**

message length	
$03	
service level *	- minimum/standard 1/2. Must be set to "minimum"
service type *	- block/stream mode 1/2
inactivity indicator	- see note (1)

Notes :

(1) A value of zero to 58 means that the minimum value of the inactivity timeout will be set (59 seconds). Any other value is the value of the timeout, in seconds.

4.2.3.3.3 L_DATA.request

message length	
$09	
cell tag (2)	- location of the data to be transmitted. See note 1
activity identity (4)	- identity of owner of data buffer. See note 1
length of data (2)	- total length,in octets, of data to be sent. See note 2
event number	- optional event to be caused after output. See note 1
offset	- offset, within cell, of start of data. See note 1

Notes :

(1) If no event is associated with the request (ie event number = 0), then TLink will assume the cell is a TLM buffer and release it (see section 4.3.1). In this case the activity indicator field is undefined. If an event is associated it will be caused, after transmission complete, to the specified activity.

(2) At least eight octets must be reserved before the start of the data and four octets after the end of the data. These octets are filled in by TLink. The length-of-data parameter does not include these reserved octets

(3) Prior to transmission TLink will freeze the segment holding the indicated cell. After transmission the segment is thawed before causing any associated event.

4.2.3.3.4 L_DISCONNECT.request

message length	
$05	
data	- passed unchanged to peer link user

4.2.4 Messages originating from the Link Layer.

4.2.4.1 Messages destined for the TLink Manager.

4.2.4.1.1 **L_START.confirm**

message length	
$72	
response code	- see note 1

Notes :

(1) Value 0 = OK
Any other value is a fail indication and reason code.
The reason codes are as defined in section 4.2.4.2.3 .

4.2.4.1.2 **TLM_CLOSE.indication**

message length	
$33	
reason code	- reason for close down. See note 1.

Notes :

(1) Zero = "normal" close down ie L_DISCONNECT.request or LR LPDU received. Any other value is an error code as defined in section 4.2.4.2.3.

4.2.4.2 Messages destined for the Session Layer.

4.2.4.2.1 L_CONNECT.indication

message length	
$02	
service level	- minimum/standard 1/2 will be set to "minimum"
service type	- block/stream mode 1/2

4.2.4.2.2 L_CONNECT.confirm

message length	
$04	
service level	- minimum/standard 1/2 will be set to "minimum"
service type	- block/stream mode 1/2

4.2.4.2.3 L_DISCONNECT.indication

message length	
$06	
reason	- disconnection reason. See note 1
data	- data passed from peer link-user. See note 2

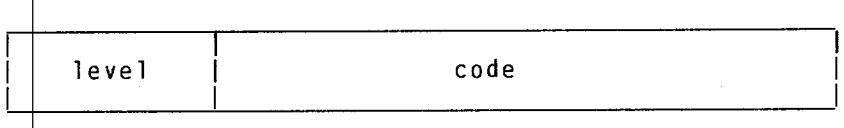
Notes :

(1) This parameter indicates whether the disconnection was induced by ther peer link user or within the co-located TLink. In the first case the parameter value is $FF In the second case the value is the reason why TLink disconnected. These values are shown in the attached table.

(2) If the disconnection was TLink induced, then this parameter has no meaning and its value is undefined. If the disconnection was not TLink induced, it is passed transparently by TLink. If the corresponding TLink is not in an OPD, then no meaning can be assigned to this parameter.

(3) This disconnection message means that the reliable link has been ended and TLink has suicided.

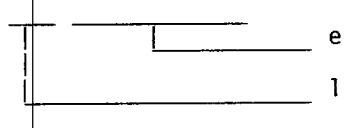
The disconnection reason octet consists of two fields :-



The level indicates whereabouts in TLink the error causing the disconnect happened. Values are :-

0 within the physical layer
1 within the link layer
2 within the session layer
3 reserved and undefined

Thus if the LT retransmission count were exceeded the disconnection reason code would be hexadecimal 4C :-



A table of error codes as generated by the LINK layer follows :-

Code	Reason
01	Illegal message received
02	LINK busy/overloaded eg no space on the output queue
03	Serial/series number checksum fault
04	Software fault/illegal response to P_CONNECT.request
05	Hardware fault eg device error on modem
06	timeout matured
07	receiver did not get PDU within 10 seconds (ie little old lady timer matured)
*08	attempt to redial failed number too soon after the last
*09	attempt to dial proscribed number
*10	bad number list full and unextendable
11	LR retransmission count exceeded
12	LT retransmission count exceeded
13	error receiving message from user
14	unable to allocate buffer for users' data
15	user inspired error
16	error on delivering data to user

17	error delivering L_CONNECT.confirm to user
18	unable to initialise LINK layer
19	illegal cell tag received from user
20	not enough reserved space in users data
21	can't signal user that his buffer is free
22	unable to send P_CONNECT.request
23	unable to get buffer for overflow user data
24	unable to open interval timer channel
25	error receiving message from PHYSICAL
26	bad cell tag in output queue
27	error sending message to PHYSICAL
28	too many successive CRC failures
29	not minimal service/block mode
30	can't get buffer for LR LPDU
31	users' data exceeds protocol maximum
65	disconnection induced by peer LINK

* indicate error codes allocated to bad number list processing which is not implemented in the current version of TLink

4.2.4.2.4 L_DATA.indication

message length	
$0A	
cell tag (2)	- location of data. See note 1
length of data (2)	- total length, in octets, of the data
offset	- offset within cell, in octets, of data start

Notes :

(1) The data is passed in a TLM buffer (see section 4.3.2) which must be passed back to the free pool after use.

(2) The segment holding the indicated cell is not frozen.

4.2.4.3 Messages destined for the Physical Layer

4.2.4.3.1 **P_CONNECT.request**

Length of message	Not including length of message octet
Message identifier	See note (1)
Data receiver identity (4)	Activity Ident. of who receives input data
Receive event no.	Event to be signalled when data is received. See note (3)
Line no	Line no to use in call. See note (5)
Modem details (2)	As required
Length of dial string (2)	A zero value indicates a data-in-voice call. See note (2)
Timing (2)	See note (6)
Charge Band (2)	See note (6)
Dial String	See note (6)

Values in brackets indicate the actual number of octets used by the field. For instance the modem details occupy two octets of information.

Notes:

(1) On the initiator side of a connection value = $31. On the acceptor side value = $B1.

(2) On the initiator side of the link this will hold the actual length of the dial-string. On the acceptor side any non-zero value can be used to distinguish between a data/data-in voice call.

(3) A value of zero means that no local event will be caused.

(5) Must be used for data-in-voice calls. The rules governing setting of this number for line selection

are as per OPD rules.

(6) These fields have no meaning on the acceptor side of the link.

(7) The meaning of the fields containing the modem details line number, timing and charge band are explained in ref 5.

4.2.4.3.2 **P_DATA.request**

Length	Not including length octet
$32	Message Identifier
Cell tag (2)	Where the data to be sent is
Activity id. (4)	Activity ident. that event is to be caused to
Data length (2)	Total length of data to be transmitted
Event No	Event to be caused when transmission complete

If the event no. is zero no event will be signalled.

4.2.4.3.3 **P DISCONNECT.request**

Length of message	Not including length-of-message octet
$73	Message identifier
Action	$00 = Don't disconnect the call $01..$FF = disconnect the call

4.2.5 Messages originating from the Physical Layer.

4.2.5.1 Messages destined for the TLink Manager.

4.2.5.1.1 P_START.confirm

Length of message	- not including length-of-message octet
$72	- message identifier
Response code	- successful/failure indication. see note (1)

Notes: (1) Response code 0 = OK. Non-zero is fail and fail reason. Values as defined in section 4.2.5.2.3.

4.2.5.1.2 TLM_CLOSE.indication

Length of message	- not including length-of message octet
$21	- message identifier
Reason Code	- not currently used

4.2.5.2 Messages destined for the Link Layer.

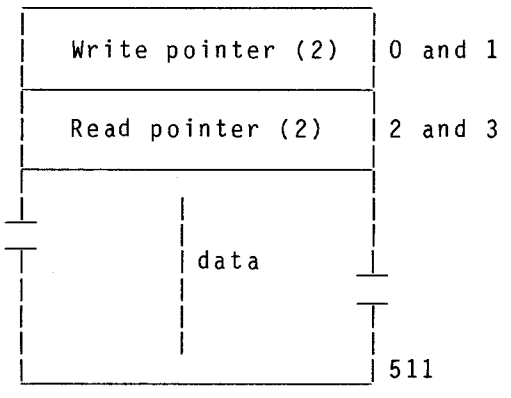
4.2.5.2.1 **P_CONNECT.confirm**

Length of message	Not including length-of-message octet
$01	Message identifier

This is an implicit success response. Any failures following a P_CONNECT.request are notified via a P DISCONNECT.indication (see 3.2.2.2.2)

4.2.5.2.2 **P_DATA.indication**

If requested in the P_CONNECT.request (see s. 3.2.2.5.1) an event will be signalled when one or more octets of data have been received. The received-data buffer looks like this:-



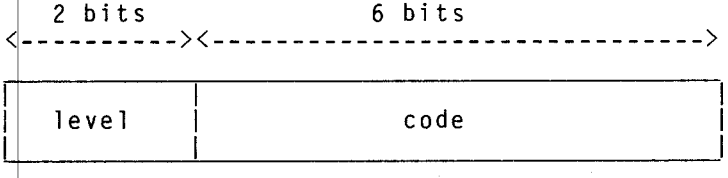
The physical layer updates the write pointer when data is put into the buffer. The read pointer is never changed by the physical layer. The physical layer writes data cyclically and makes no attempt to check wrap-around. It is recommended that changes to the write pointer, other than by the physical layer, are done with caution.

Note that there is no message sent through L_PHLK_x associated with a P_DATA.indication.

4.2.5.2.3 P_DISCONNECT.indication

Length of message	Not including length-of-message octet
$02	Message identifier
Reason code	Reason for failure

The disconnection reason code consists of two fields as shown :-

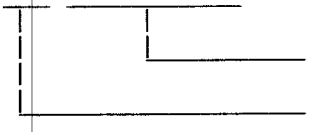


The level indicates whereabouts in TLink the fault occurred.

0 at the physical layer
1 at the link layer
2 at the session layer
3 reserved and undefined

Thus if PHYSICAL fails to open the modem channel the disconnection reason code octet would be hexadecimal 13 :-

0 0 0 1 0 0 1 1



The possible reason code values, as generated by the PHYSICAL layer, and their meanings are shown in the attached table.

Code	Reason
00	normal disconnection - P_DISCONNECT.req received
01	illegal parameter in message
02	telephone engaged / modem busy / out of memory
03	telephone line not open
04	software fault
05	hardware fault
08	Telephone Handler timed out
09	modem channel status fail after call connection
10	error closing modem output channel
11	error while sucking the modem
12	error receiving message from user
13	unable to start SENDER
14	error from TH when data auto-answering
16	error sending message to SENDER
17	unable to deliver P_CONNECT.confirm
18	failed to initialise

19	failed to open modem channel
20	error from TH when configuring to data
21	can't close pipe (the diorrhea syndrome)
22	can't open pipe (the constipation syndrome)
23	message from user exceeds maximum permissible size
24	error from TH on auto-dialling data
25	unable to signal user layer

4.3 Memory Management Interfaces.

The management of memory which holds data for transmission and data received is considerably simplified for Application writers by the provision of a centralised pool of data buffers which are handled by TLink Manager. Applications need not use the TLM cell pool but under most circumstances it would be simpler for them so to do.

Once an Application requests either Session, Link or buffered Physical service, or once a data call is auto-answered by TLink, TLM sets up a pool of cells which are available to both Applications and layer entities for temporary storage.

4.3.1 Use of Pool Cells in Transmission.

An Application which has data to transmit requests a cell using a REQUEST TLM CELL call (s. 4.3.3.1). It places the data for transmission in the cell provided (with due regard for offsets required from the start of the cell), and quotes the cell identifier in the **x_DATA.request** to the layer entity it is using. If it provides an event number in the request, that event will be caused once the data in the cell has been transmitted and acknowledged. The Application can then either re-use the cell or release it back to the pool using a FREE TLM CELL call.

If no event number has been provided, the Physical entity assumes responsibility for returning the cell to the pool once the data has been transmitted. Thus the Application need not concern itself further with the cell.

Applications which undertake themselves to return cells to the pool must be well-behaved. If cells are not promptly returned to the pool it will become exhausted and abrupt Link termination will result.

Applications which do not use pool cells must associate an event number with them in the **x_DATA.request**. If they do not, the Physical Entity will attempt, and fail, to release the cell after transmission of the data it contains.

4.3.2 Use of Pool Cells in Reception

Applications which use Physical services obtain data from a circular buffer as described in s. 3.2.2.6.2. Applications using Link or Session services, however, receive a cell identifier in an **x_DATA.indication**. The cell contains received data. Once the Application has extracted or finished with the data in the cell it must return the cell to the pool with a FREE TLM CELL call as described below (s. 4.3.3.2). Failure to return the cells to the pool promptly will cause abrupt Link termination when the pool becomes exhausted.

4.3.3 Cell Allocation and Release calls

Two methods of access are provided, this is to enable backwards compatibility for early OPD software/firmware using TLink services. An application may either :-

(1) use DIRECTOR traps

(2) use the CALL PROGRAM interface

The second method is only possible on OPDs that support paged ROM. The first method may be used on any OPD, however it is provided for backwards compatibility only. It is most strongly recommended that the CALL PROGRAM interface is used wherever possible as this is the preferred method for future enhancement capability. To this end applications using memory management should do a GIVE VERSION NUMBER call (reference 2 on the base functional software. If it transpires they are running on a machine supporting paged ROM then the CALL PROGRAM interface should be used. If the host machine does not support paged ROM then the TRAP interface must be used.

4.3.3.1 Memory management TRAP interfaces

To use the following calls, an Application should include the file SYS:99.TLINK.L9VALUES.DG which contains the required action values. The calls themselves are made by obeying the instruction:

TRAP #T.TLINK

with a particular Action Value in D0.B.

Processing of the call may consume up to 128 octets of the Application's stack, and care should be taken to ensure that sufficient stack is available when the call is made.

4.3.3.1.1 REQUEST TLM CELL

Trap Name: T.TLINK
Action Value (D0.B): L.GETCELL

Return values:

D0.B Number of free buffers remaining.
ERR.OM if no buffer could be allocated.
ERR.NF if there is no TLink activity running

D1.W Cell tag of buffer allocated.

The cell returned from a successful call is 274 octets long.
The associated memory is not frozen.

4.3.3.1.2 FREE TLM CELL

Trap Name: T.TLINK
Action Value (D0.B): L.FREECELL

Additional Call parameters:

D1.W Cell tag of buffer to be freed.

Return values:

D0.B ERR.BP if the cell specified could not be freed.

Note that if a valid TLM buffer is specified by this call, the cell will be freed whether it has been assigned to the freeing application or not.

4.3.3.2 Memory management CALL PROGRAM interfaces

This section will be updated as the OS firmware becomes available. (reference 2 applies)

4.3.3.2.1 REQUEST TLM CELL

TBD.

4.3.3.2.2 FREE TLM CELL

TBD.

5 PROGRAMMERS GUIDE - HOW TO USE TLINK

This section is essentially a recap of what has gone before. It is arranged in a more digestable form for programmers. Hints and troubleshooting are added. This section describes how to :-

- make calls via TLink
- receive calls via TLink
- interpret error messages from TLink
- obey automatic redialling rules

For simplicity the usual case of making/receiving a data call via the SESSION service of TLink is described. Those using services of other layers or data-in-voice calls may use this section as a guide but must refer to earlier sections for detail.

Any references to earlier sections in this document are presented thus {4.2.2.1}

Whether making or receiving calls the same general process needs to be followed. This is

i) set up TLink to make/receive call.
ii) send or receive data via TLink.
iii) tell TLink to close down.

In the event of unrecoverable errors TLink must be aborted. Similarly TLink can abort if it detects unrecoverable errors.

5.1 Making calls via TLink

As stated the process falls into three parts :-

i) Get the TLink Manager (TLM) to establish a TLink SESSION service
ii) Use TLink to make the phone call and then transfer data
iii) Close down

We will discuss each in turn.

5.1.1 Getting TLink Manager to set up TLink service

Execute the following steps.

1. use DIRECTOR to find the TLM activity identity {3.2.1.1 part 1}
2. open pipes L_TMAP_1 and L_APTM_1 to and from TLM
3. signal local event 14 to TLM

4. send TLM_SERVICE.request to TLM {4.2.2.1.1}

5. wait for TLM_SERVICE.response or APP_CLOSE.indication {4.2.1.1.1 and 4.2.1.1.2}

6. send APP_READY.indication or APP_CLOSE.indication to TLM {4.2.2.1.2 and 4.2.2.1.3}

This concludes the dialogue with TLM and applications need have nothing more to do with TLM. All being well the TLink activities are all running and all further communication is with SESSION.

Troubleshooting

This subsection answers some of the more frequently asked questions and suggests reasons for various outcomes. Specific solutions are not usually offered as these must vary with an application's needs.

Perceived problem	possible explanation/recommendation
Can't send TLM_SERVICE.request down the pipe	1) Not enough time allowed for TLM to react to event 14 and open its end of the pipes. Recommendation : Wait a short while for TLM to get going 2) TLM is busy with another request and will not see event 14 for several seconds. Recommendation : Back off and try again.
No reply to TLM_SERVICE.request	1) TLM busy with another TLM_SERVICE request and will not respond for several seconds. Recommendation : Back off and retry later 2) Too slow in sending TLM_SERVICE.req. TLM will wait only 1-2 seconds after seeing the event 14. Recommendation : Start from scratch.
Negative TLM_SERVICE.response	1) out of memory 2) TLink in use 3) internal TLink fault In all cases it is recommended that applications s end an APP_CLOSE.ind back to TLM and ignore any faults from doing this.

Response to TLM_SERVICE.req is APP_CLOSE.ind	Recommendation : treat as negative TLM_SERVICE.response meaning CMOS found to be corrupt
Can't get TLM activity identity	1) TLink deleted via store report 2) Another application specified as the data auto-answer protocol so TLink not started at power-up time. In either of the above two cases manual intervention is required.
Positive TLM_SERVICE.response but I don't want TLink any more	Recommendation : send APP_CLOSE.indication to TLM rather than APP_READY.ind.

5.1.2 Using TLink to make the telephone call

Assuming that the steps in 5.1.1 have completed OK we now need to get TLink to make the phone call and establish a link with the remote end.

To do this :_

1. open the pipes specified in the TLM_SERVICE.response
2. send a S_CONNECT.request {4.2.2.2.1} See also hints below
3. wait for S_CONNECT.confirm {4.2.3.2.2}

Some hints for the S_CONNECT.request

- Line number zero tells the OPD to use the default data line or if in use the other line or if single line OPD to use the single line. In other words line number zero is always acceptable and will produce a good result.
- A default set of modem details is contained in CMOS record 2. This record is 4 bytes long. The first two bytes contain the default modem details for receiving calls and the last two bytes the details for making calls. Thus you may wish to read this record and use the last two bytes of it in the connect request as there is a good chance of the modem at the far end having a match.
- the timing and charge band parameters are as described in reference 5. However values of binary zero mean no timing and no chargeband.
- the source identity and address fields are transferred unchanged to the application at the far end. So they can be used to transfer information. For instance the source address could contain your

telephone number.

- the destination identity field is very important and readers are advised to peruse the following carefully. The destination identity is used by TLink to determine what application program to start up the far end of the link. So, if you want your application to be started up, so you can talk to it you must fill in its name here. If this parameter is set to "FRED" ,for example, the remote TLink will look for application FRED at the remote end. The application name used for the search is exactly 12 characters long and the remote TLink will truncate or add trailing spaces if necessary. If the destination identity is null then the remote TLink will start the application specified in CMOS as the "data auto-answer application name". This record can be altered with the (co-located) CONFIGURATOR tool. In all cases the remote TLink will search memory and both microdrives for the specified application.

Hints for the S CONNECT.confirm

- If the destination system type is not "OPD" then you have contacted an alien machine operating the TLink protocol. If the destination file type is "KERNEL" then the called machine is emulating an OPD/KERNEL.

- If the destination system type is "OPD" but the file system type is not "KERNEL" then you have contacted an OPD emulating an alternative operating system.

Applications that can run on alien machines must make their own arrangements to ensure that they have contacted their own application on that machine. This is because alien TLink implementations may not use the destination identity field in the same way that OPD does. An OPD emulating a non-KERNEL operating system will presumably offer interface compatibility to TLink so that the application specified in the destination identity will be activated.

Troubleshooting

Perceived problem	possible explanation/recommendation
Can't send S_CONNECT.request	TLink is started after the APP_READY.ind. Possibly the connect request was made before TLink was ready. Recommendation : Try again
No response to S_CONNECT.req	Haven't waited long enough. MNP protocol allows 2 minutes for SESSION set up. Recommendation : Wait at least two minutes, if time expires give up and start from scratch

If all has gone well the receipt of the S_CONNECT.confirm means that TLink has established the call and started up the specified application at the far end. We now enter the data phase.

5.1.3 Sending and receiving data

Once in the data phase data can be sent or received. To send data make a S_DATA.request {4.2.2.2.5} To receive data, monitor the pipe from SESSION for a S_DATA.indication {4.2.3.2.5}.

Sending data

The data to be sent must be placed in a cell and the cell tag passed to TLink. There are two ways to get data into a cell. The first is a do-it-yourself approach and the second is to use the TLink shared buffer pool. We will discuss each in turn.

a) The D-I-Y approach is to accquire memory, make a cell segment, fill a cell with data and use its cell tag in the S_DATA.req. If you do this you must associate an event number with the S_DATA.request. The reasons for this will become apparent later. The event will be signalled back to you when the data has been successfully received at the far end. Naturally the data in the cell must not be changed before this event is signalled. It is possible to send more than one S_DATA.req at a time but be aware of two problems :-

   - if you overload TLink it will disconnect
   - different events must be associated with each request or you cannot determine when it is safe to re-use a cell.

   It is difficult to quantify "overload" as the demands on TLink vary in real-time. However as a rough guide no more than three data requests should be outstanding at any one time.

b) The shared buffer approach is recommended. Using such, it is not necessary to associate events and you can send as many S_DATA.requests as you can with no fear of overloading TLink. The shared buffer pool is always created whether or not you use it as TLink uses the pool to pass data to an application.

   To get use of a TLink buffer use the interfaces described in section 4.3 of this document, specifically the "request TLM cell" interface. Freeze the buffer, fill it with data, thaw it and send to TLink. If you do not associate an event (ie event field is zero) then forget about it and TLink will automatically release the cell back to the pool when ready. If you do associate an event then, when it is signalled, you can either re-use the buffer or release it back to the free pool via the "free TLM cell" interface.

The reason that D-I-Yers must use an event is now clear. If you don't TLink will assume it's a shared buffer cell and attempt to release it back to the free pool. The resulting catastrophe will cause TLink to disconnect.

TLink can handle the maximum number of S_DATA.requests possible when using the shared buffer pool.

To get a shared buffer an algorithm along the lines of the following is recommended :-

```
FOR a fixed number of times

    get TLM cell

    IF got then continue processing with the supplied cell

    IF response to "Get TLM cell" was "none available"
    THEN

        wait x seconds

    ELSE (must be an error of some sort)

        do error processing

    ENDIF

ENDFOR

IF not got
THEN  (we have made our maximum number of attempts to get a buffer)

    error processing

ENDIF
```

The "fixed number of times" and "x" will vary with the needs of the application. As a guide, existing applications try 10 - 20 times at intervals of 5 - 10 seconds.

Receiving data

TLink will pass data to an application in a S_DATA.indication. {4.2.3.2.5} This is quite unsolicited so applications must continuously monitor the pipe from TLink. The data is passed in a TLink shared buffer cell. The associated segment is not frozen on receipt of the S_DATA.indication. The buffer must be passed back to the free pool when you have finished with the data. To free the buffer use the "free TLM cell" interface in section 4.3 of this document. If cells are not released back to the pool and it becomes exhausted TLink will simply stop. However, it should be noted that if the inactivity timer expires ie TLink is stopped for a period of 59 seconds, then TLink will disconnect. If the other end is trying to send data it will eventually get tired of doing retransmissions and disconnect. This time is between 90 and 120 seconds.

5.1.4 Closing down

After sending and receiving all the data it only remains to close down TLink. This can most simply be done by the application at either end sending a S_DISCONNECT.request. However it is recommended that a DISCONNECT.request be used only in emergencies and that normal closedown use the graceful disconnection mechanism that will now be described.

The graceful closedown is a two way handshake with one end telling the other "I'm finished now" and waiting for the other end to say "I'm finished too". On completion of the handshake TLink disconnects without any further prompting. To start graceful closedown one application sends a S_CLOSE.request {4.2.2.2.4} to TLink and waits for a S_CLOSE.indication {4.2.3.2.4}. When an application receives a S_CLOSE.indication it completes any outstanding work and then sends a S_CLOSE.request. When the sender of the first S_CLOSE.request receives a S_CLOSE.indication it considers closedown complete, TLink will closedown after sending the S_CLOSE.indication. When the receiver of a S_CLOSE.indication sends a S_CLOSE.request it may consider closedown complete, TLink will disconnect once the CLOSE has been delivered to the far end.

It is irrelevant to TLink which end initiates a graceful closedown so applications can either operate the mechanism asynchronously (ie the first end to finish starts the closedown sequence) or in an agreed manner (eg the call originator always initiates closedown).

5.1.5 Emergency closedown

An application may abort proceedings and force TLink to abandon by sending a **S_DISCONNECT.request** {4.2.2.2.3}. This request may be made to TLink at any time once an APP_READY.indication has been sent. TLink will tell the other end of the link to abort and will endeavour

to pass any disconnection reason supplied to the distant application. It is impossible to have further conversation with TLink once this request has been issued.

Conversely TLink can abort proceedings by sending a S_DISCONNECT.indication {4.2.3.2.5}. This can be sent at any time once a TLM_SERVICE.response has been issued and applications must always be prepared to receive it and react. It is impossible to converse with TLink once a DISCONNECT.indication has been received. TLink will disconnect both ends of the link and endeavour to pass the same disconnection reason to the connected applications.

5.2 Receiving calls via TLink

As with making calls we go through three phases :-

i) using TLink Manager to set up TLink
ii) using TLink to send or receive data
iii) closing down

The data and closing down phases are identical to those for a call originator.

5.2.1 Getting in touch with TLink

If you have read section 5.1 you will know that TLink starts the application specified in the destination identity field of the S_CONNECT.request. Generally, this is the first an application knows anything about an incoming call. However we will briefly describe how an incoming call is answered.

When the phone rings and the line is set for auto-answer-data TLink Manager is informed. TLM starts up a TLink SESSION level service. TLink then waits for the caller to establish contact. Eventually an SR SPDU should arrive at the SESSION layer which contains the destination identity of the application to be started. As described in section 5.1.2 a default application will be started if this field is null.

TLink starts the specified application in such a way that if it is already running the application will not be affected. Once TLink has started the application or found it is already running it will cause event 20 to it. It is clear that incoming calls are unsolicited from the point of view of an application using TLink and therefore you must always be prepared to receive and act upon event 20. (It is not possible to receive this event while an application is using TLink so if you are making a call this event cannot be signalled until the TLink service is closed down)

Once the event is seen the following steps need to be taken :-

1. Open pipes L_TMAP_1 and L_APTM_1 from and to the TLink Manager and wait for the arrival of an APP_SERVICE.indication. {4.2.1.1.2}

2. Open the pipes specified in the APP_SERVICE.indication to establish contact with TLink.

3. Send an APP_READY.indication {4.2.2.1.2} to TLM.

4. Close the pipes specified in 1. above

5. Wait for a S_CONNECT.indication {4.2.3.2.1} from TLink

6. Send a S_CONNECT.response {4.2.2.2.2} to TLink

If all has gone smoothly the link is now established and you can send or receive data

Hints for the S_CONNECT.indication

- if the source system type is not "OPD" then you are in contact with an alien machine which uses the TLink protocol and has somehow managed to start your application. If the source file type is "KERNEL" then you are in contact with an alien machine emulating an OPD.
- if the source system type is "OPD" but the source file type is not "KERNEL" then you are in contact with an OPD emulating a different operating system.
- the source identity is the string passed to TLink in the S_CONNECT.request by the application at the remote end of the link.
- ditto for the source address

If the source system type/source filestore type are incompatible then an application must decide on whether to proceed. The notes in section 5.1.2 on connecting with alien machines may be of interest.

Troubleshooting

Perceived problem	Possible explanation/recommendation
No contact with TLink from incoming calls	TLink waits for 1-2 seconds after starting an application before sending event 20. Therefore you must be prepared to accept events within this time frame otherwise the event 20 will be lost.

5.2.2 Sending and receiving data

Once the connection to TLink is established it does not matter whether an application originated or received the call from the point of view of sending and receiving data. Hence section 5.1.3 provides all the information required.

5.2.3 Closing down

It is irrelevant to TLink whether the call originator or call acceptor initiates closedown. Refer to section 5.1.4.

5.2.4 Emergency closedown

Either the call acceptor or call originator can abort at any time. Section 5.1.5 refers. Programmers are reminded that TLink can abort at any time too. Again, section 5.1.5 refers.

5.3 Use of error codes

If TLink disconnects it will pass a S_DISCONNECT.indication to the application. There are three possible reasons for TLink disconnecting.

1) The co-located TLink has detected an unrecoverable fault
2) The remote TLink has detected an unrecoverable fault
3) The remote application has decided to terminate and has sent a S_DISCONNECT.request to TLink

A S_DISCONNECT.indication contains two information fields {4.2.3.2.3}. These are the "reason" and "data" fields. These are interpreted as follows :-

- if the "reason" field is -1 ($FF) then the disconnection was induced by the remote application.
- if the "reason" field is -2 ($FE) then the disconnection was induced by the remote TLink.
- if the "reason" field is -1 then the "data" field contains whatever the remote application put into the "data" field of the S_DISCONNECT.request {4.2.2.2.3}.
- if the "reason" field is not -1 and is not -2 then the "data" field has no meaning and the "reason" field contains the TLink error code. TLink error codes are discussed below.
- if the "reason" code is -2 then the "data" field contains the TLink error code for the remote TLink disconnection. TLink error codes are discussed below.

Examples

1) An application wishes to abrubtly disconnect and sends TLink a S_DISCONNECT.req with the "data" field set to $AB. The application at the other end of the link will receive a S_DISCONNECT.ind with the "reason" field set to -1 ($FF) and the "data" field set to $AB.

2) TLink discovers internal error number $81. The co-located application receives a S_DIS.ind with the "reason" field set to $81. No meaning can be ascribed to the "data" field. The remote application receives a S_DIS.ind with the reason field set to -2 ($FE) and the "data" field set to $81.

5.3.1 TLink error codes

If the "reason" field in a S_DISCONNECT.indication is not -1 ($FF) then the "reason" field contains the reason why TLink disconnected. The "reason" field contains two sub-fields as follows :-

bits 6 and 7 - level at which the fault was detected
0 = at the PHYSICAL layer
1 = at the LINK layer
2 = at the SESSION layer
3 = not used but reserved for possible future expansion

bits 0 - 5 - an error code

Thus in example 2 above; error reason $81 is error code number 1 at the SESSION level.

The error code numbers for each level may repeat. For example :-

error reason $03 is error code number 3 at the PHYSICAL level
error reason $83 is error code number 3 at the SESSION level
similarly error reason $53 is error code number 19 at the LINK level

The error code numbers for each level are tabulated in the following sections :-

PHYSICAL	section 4.2.5.2.3
LINK	section 4.2.4.2.3
SESSION	section 4.2.3.2.3

5.4 Automatic redialling rules

An application may detect that a call has failed, and may wish to automatically re-dial the number for another attempt. Such an action is termed an Automatically Generated Re-dial Attempt (AGRA). Telephone network authorities in most countries have specified rules governing the making of AGRAs. Typically, the rules require that a minimum time must elapse between each attempt to a given number and specify a maximum number of attempts that can be made without human intervention. Failure to comply with the rules may mean that an application will not be given permission to connect to the PTSN.

A component termed the Bad Number List Processor (BNLP) is provided to support compliance with the relevant rules. All applications which make AGRAs **must** make use of the BNLP. Reference 13 covers its use.

6 STANDARDS

The implementation of TLink is fully in accordance with the Microcom specification (ref 3).

7 PRODUCT PERFORMANCE

Full-duplex communications at 300 bps, assymetric full duplex at 1200 and 75 bps and half-duplex at 1200 bps will be supported dependant on the processor loading contributed by other programs active during a communications session. The effects of other programs running and their competition for resources is for further study.

8 HARDWARE CONFIGURATION AND SOFTWARE ENVIRONMENT.

Tlink uses the software interfaces provided by Kernel (ref 1), Director (ref 2) and Telephone Handler (ref 5). In addition, the hardware support described in ref 10 is required.

9 COMPATIBILITY WITH OTHER ICL PRODUCTS.

There are currently no committed implementations of TLink on ICL products. It is, however, intended that compatibility will be achieved by adherence to the Microcom specification (ref 3).

10 RELIABILITY, RESILIENCE, MAINTENANCE AND USABILITY.

The implementation will behave in accordance with the Microcom specification and the interfaces will be as described in this document. It will be as reliable and resilient as the underlying OPD hardware and software. There are currently no metrics available which can be used to measure its reliability or resilience.

TLink will reside in ROM with base software. It is therefore not field-maintainable.

There are no usability issues associated with TLink, other than the interfaces described in section 4 of this document. These have been designed for both efficient implementation of TLink and Applications using it, and are known to be usable by competent software implementors.

11 VALIDATION AND PERFORMANCE PROVING TESTS.

These will be fully described in a test specification which will be based on test schedules to be supplied by BTE.

12 TEST EQUIPMENT and TEST SOFTWARE.

In order to test against Microcom's own implementations of their protocol, some or all of the following may be used:

- IBM Personal Computer with autodialling modem running IBM Personal Communications Manager package.
- IBM Personal Computer with Microcom ERA 2 modem running ERA 2 package.
- Microcom PCS 2000 modems.
- Apple McIntosh running MacModem
- Reference implementation to be supplied by BTE.

To capture and observe the flow of data between an OPD and another machine, a telephony module be modified to provide a V24 interface to which a data communications tester may be connected. Currently a Digitech Encore 200 is used to capture data, and programs have been written to analyse the captured data and identify the Link, Session and File Transfer protocol data units being sent and received.

13 ENHANCEMENT CAPABILITY.

The design of the current implementation is modular and modelled on a finite-state machine. Every effort has been made to ensure a simple enhancement route, consistent with the exigencies of achieving acceptable performance and minimal code size.

14 DOCUMENTATION.

Project design (refs 6, 7, 8 and 9) and test (tbs) documentation is, or shortly will be available. This document is to be used as the basis of a chapter of a reference guide for applications writers.