The ATM Forum

Technical Committee ATM

Data Exchange Interface (DXI) Specification

af-dxi-0014.000

August, 1993

The information in this publication is believed to be accurate as of its publication date. Such information is subject to change without notice and the ATM Forum is not responsible for any errors. The ATM Forum does not assume any responsibility to update or correct any information in this publication.

Notwithstanding anything to the contrary, neither The ATM Forum nor the publisher make any representation or warranty, expressed or implied, concerning the completeness, accuracy, or applicability of any information contained in this publication. No liability of any kind shall be assumed by The ATM Forum or the publisher as a result of reliance upon any information contained in this publication.

The receipt or any use of this document or its contents does not in any way create by implication or otherwise:

- Any express or implied license or right to or under any ATM Forum member company's patent, copyright, trademark or trade secret rights which are or may be associated with the ideas, techniques, concepts or expressions contained herein; nor
- Any warranty or representation that any ATM Forum member companies will announce any product(s) and/or service(s) related thereto, or if such announcements are made, that such announced product(s) and/or service(s) embody any or all of the ideas, technologies, or concepts contained herein; nor
- Any form of relationship between any ATM Forum member companies and the recipient or user of this document.

Implementation or use of specific ATM standards or recommendations and ATM Forum specifications will be voluntary, and no company shall agree or be obliged to implement them by virtue of participation in the ATM Forum.

TABLE OF CONTENTS

TABLE OF FIGURES

1.0 INTRODUCTION

This document defines the Asynchronous Transfer Mode Data Exchange Interface (ATM DXI) which allows a DTE (such as a router) and a DCE (such as an ATM DSU) to co-operate to provide a UNI for ATM networks.

This ATM DXI document defines the data link protocol and physical layers which handle data transfer between the DTE and the DCE. This document also defines the Local Management Interface (LMI) and Management Information Base (MIB) for the ATM DXI.

All modes of operation of the DXI are capable of transparently supporting Service Specific Convergence Sublayers (SSCS) and other higher layers.

This document, however, addresses only the Common Part Convergence Sublayer (CPCS) per CCITT Recommendation I.363. The use and support of the SSCS is beyond the scope of this document. This interface does not allow the use of the "CPCS-user-to-CPCS-user" field of the AAL5 CPCS_PDU.

2.0 THE DATA LINK PROTOCOL

2.1 OVERVIEW

When an ATM DXI is used, as defined in this document, the implementation of the ATM UNI is split between the DTE and the DCE, as illustrated in Figure 2.1. Operating Modes 1a, 1b and 2 are defined below.

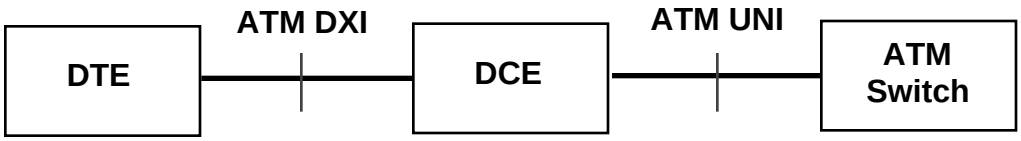


Figure 2.1 ATM DXI Illustration

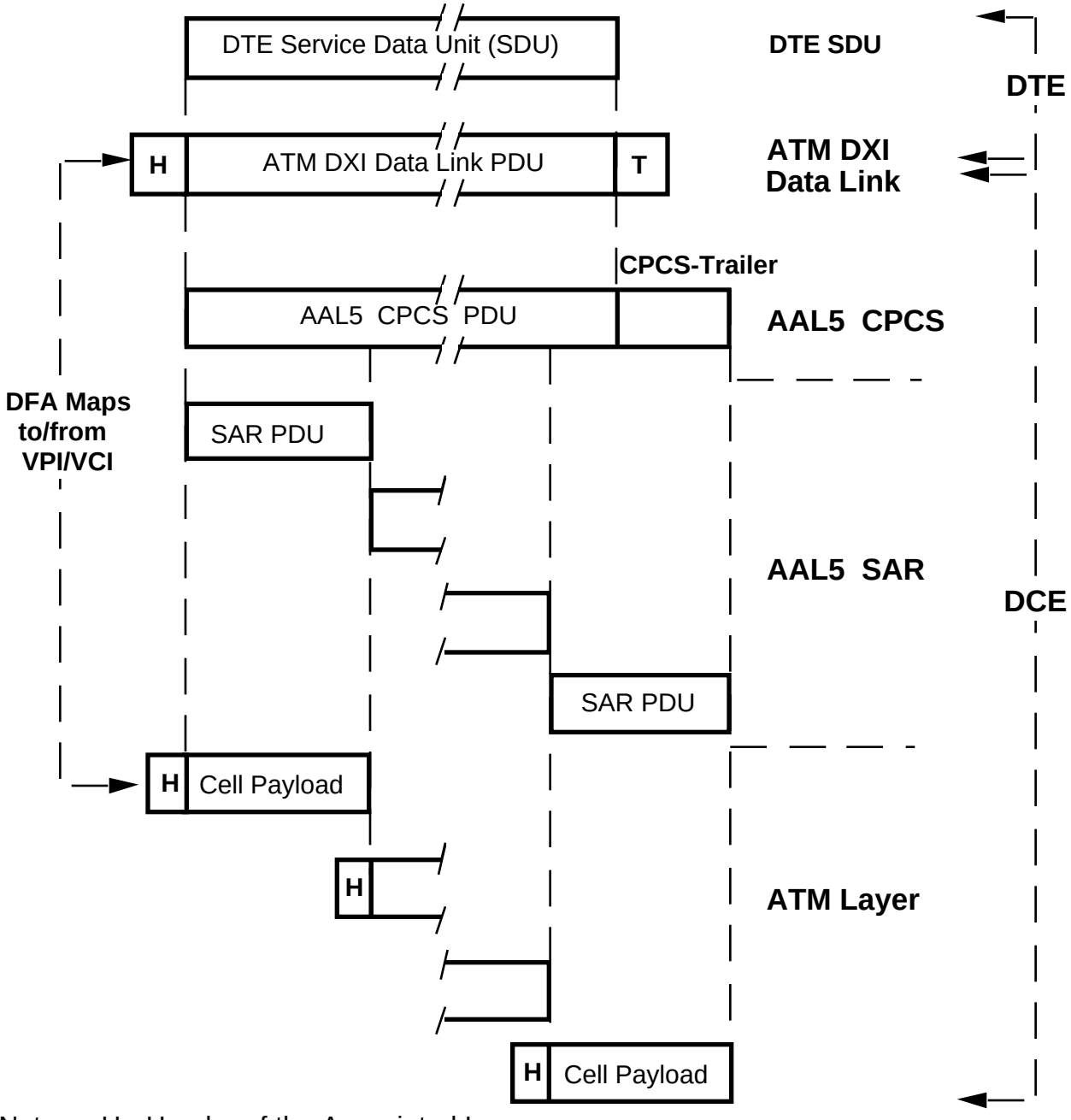
2.1.1 Mode 1a

Features

- Up to 1023 Virtual Connections
- AAL5 Only
- Up to 9232 Octets (DTE SDU)
- 16 Bit FCS (between the DTE and DCE)

The objective for the DTE is to transport a DTE Service Data Unit (SDU) via the ATM network, to an appropriate peer entity. The DTE SDU corresponds to the information field for the AAL protocol being used. With reference to Figure 2.2, the DTE will encapsulate its DTE SDU into a DXI Frame. The DCE performs the ATM Adaptation Layer 5 Common Part Convergence Sublayer (AAL5 CPCS), AAL5 Segmentation and

Reassembly (AAL5 SAR) functionality, and the ATM Layer functionality, as well as the ATM UNI functionality. The physical layer functions are not shown.



Note: H - Header of the Associated Layer
T - Trailer of the Associated Layer

Figure 2.2 Modes 1a and 1b Processing for AAL5

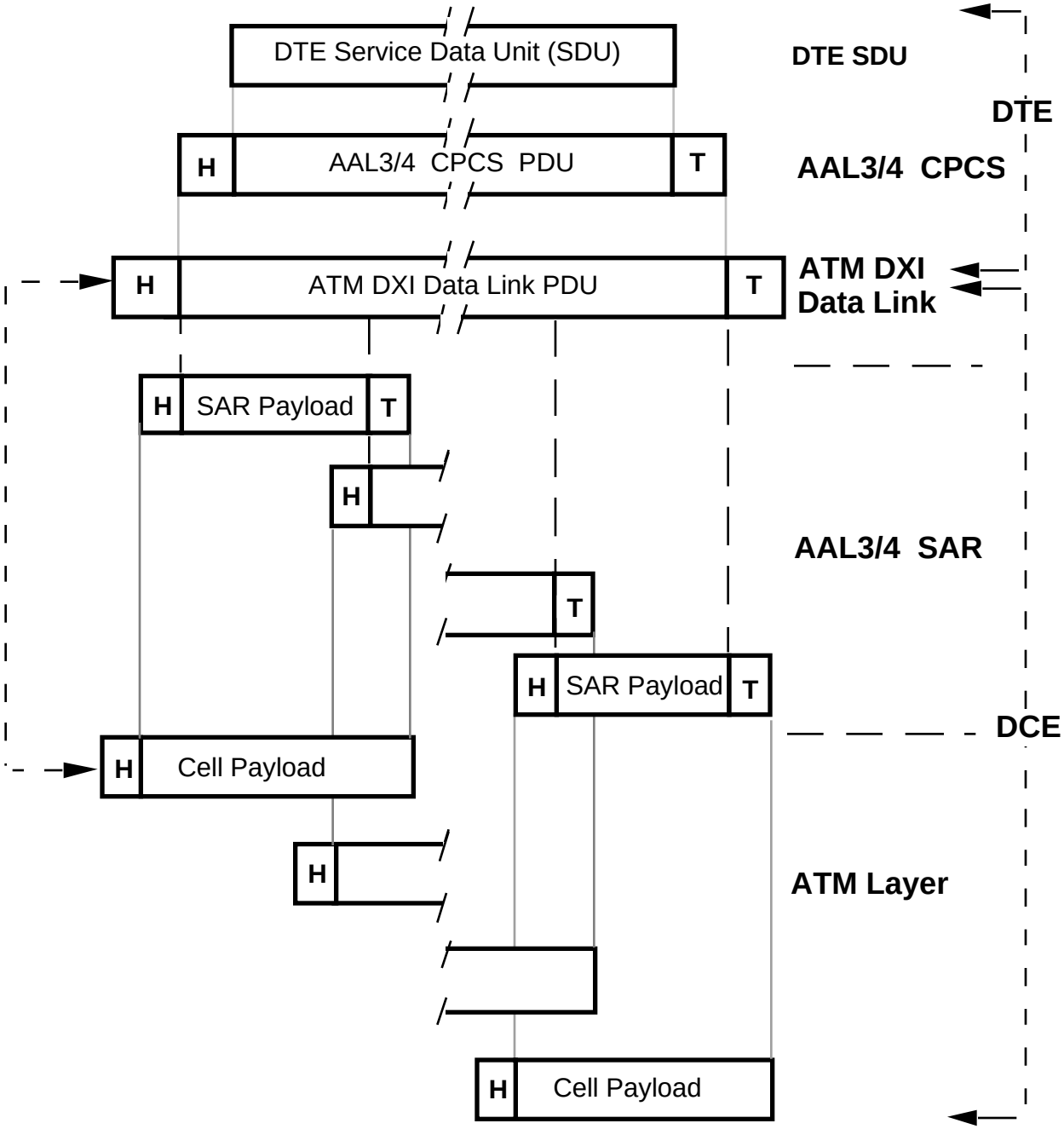
2.1.2 MODE 1b.

Features

- Up to 1023 Virtual Connections
- AAL3/4 for at least one Virtual Connection
- AAL5 for other Virtual Connections
- Up to 9232 octets (DTE SDU) when using AAL5 and
- Up to 9224 octets (DTE SDU) when using AAL3/4.
- 16 Bit FCS (between the DTE and DCE)

This mode includes the features of Mode 1a, plus the addition of support for AAL3/4 on individually configurable virtual connections. Figure 2.3 illustrates processing of AAL3/4. The DTE first performs the AAL3/4 CPCS encapsulation, then it further encapsulates that AAL3/4 CPCS_PDU into an ATM DXI Frame. The DCE performs the AAL3/4 Segmentation and Reassembly (AAL3/4 SAR), the ATM Layer functionality, and the ATM UNI functionality. Processing of AAL5 for Mode 1b is the same as described for mode 1a, as shown in Fig 2.2. The physical layer functions are not shown.

(Note: The DCE/DTE need only support AAL3/4 for a single VPI/VCI to be compliant. For example, it is acceptable for the DTE/DCE to only support AAL3/4 in support of a particular service across a single well-known VPI/VCI. Thus, following Bellcore specifications [5], a DTE/DCE could preassign the use of AAL3/4 in support of SMDS across VPI=0, VCI=15 for use with a public UNI following those specifications.)



Note: H - Header of the Associated Layer
T - Trailer of the Associated Layer

Figure 2.3 Mode 1b Processing for AAL3/4

2.1.3 Mode 2.

Features

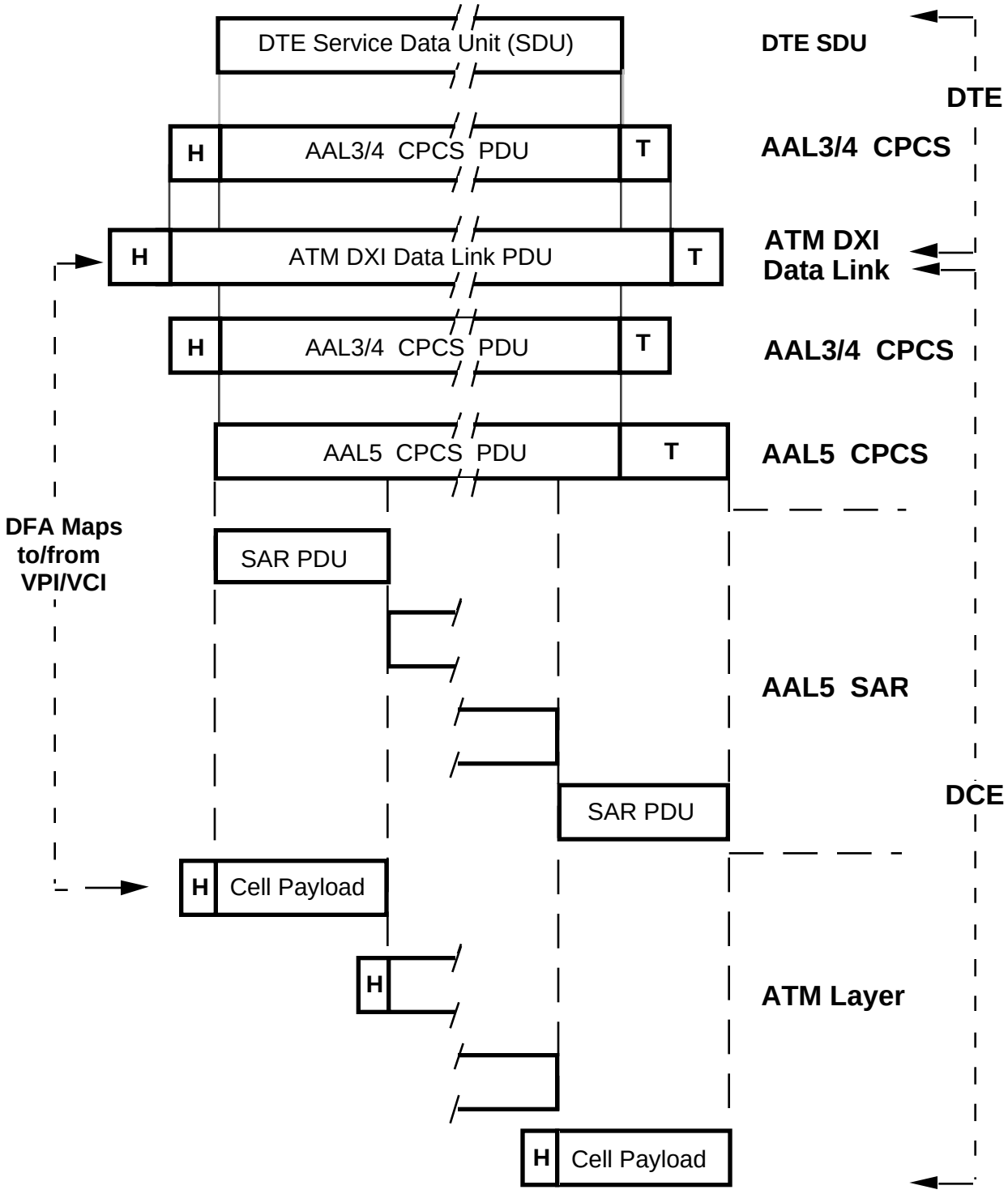
- Up to 16,777,215 (2^24-1) Virtual Connections
- AAL5 and AAL3/4, one per Virtual Connection
- Up to 65,535 (2^16-1) Octets (DTE SDU)
- 32 Bit FCS (between the DTE and DCE)

The DTE performs the AAL3/4 CPCS encapsulation and further encapsulates that AAL3/4 CPCS_PDU into an ATM DXI Frame.

The DCE performs one of the following:

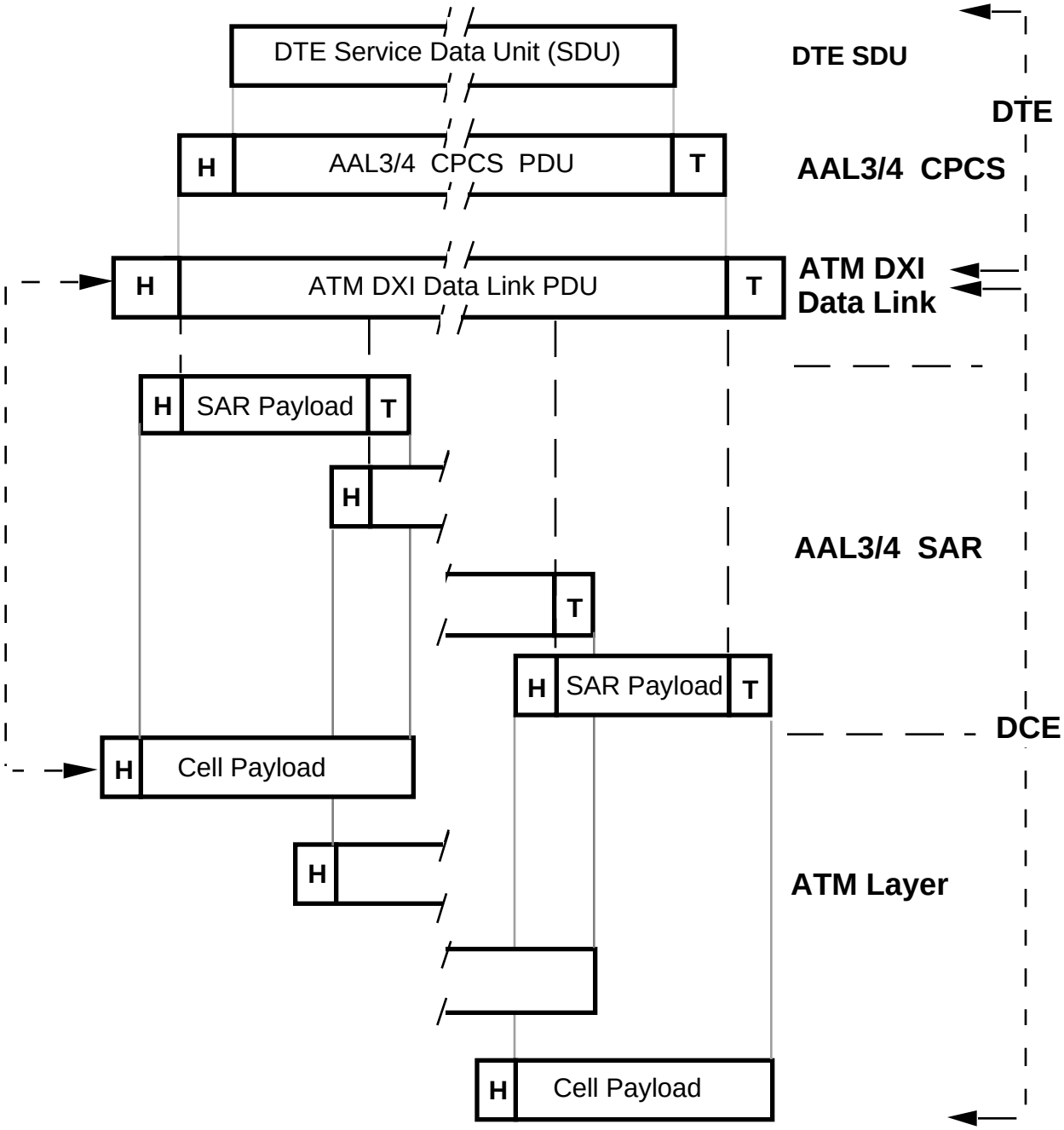
For VPI/VCI values indicating an AAL5 Virtual Connection, the DCE strips off the AAL3/4 CPCS Header/Trailer and encapsulates the remainder of the PDU into an AAL5 CPCS_PDU per Fig 2.4. The DCE then performs the AAL5 SAR, the ATM Layer functionality, and the ATM UNI Physical Layer functionality. The physical layer functions are not shown in figure 2.4.

For VPI/VCI values indicating an AAL3/4 Virtual Connection, the DCE performs AAL3/4 SAR and ATM layer functionality per Figure 2.5, and the ATM UNI Physical Layer functionality, which is not shown in Figure 2.5.



Note: H - Header of the Associated Layer
T - Trailer of the Associated Layer

Figure 2.4 Mode 2 Processing for AAL5



Note: H - Header of the Associated Layer
T - Trailer of the Associated Layer

Figure 2.5 Mode 2 Processing for AAL3/4

2.1.4 Addressing

The DXI Frame Address (DFA) is contained in the DXI Frame Header. The DFA is used to pass the ATM Virtual Path Identifier (VPI) and Virtual Channel Identifier (VCI) information between the DTE and DCE. The DFA is ten (10) bits in Modes 1a and 1b, and twenty-four (24) bits in Mode 2. Details are provided in the sections below.

A predetermined DFA is reserved as a management channel for communications between the DTE and DCE. The predetermined DFA is set at all 0s.

2.2 THE PHYSICAL LAYER

The DTE/DCE will use the V.35, EIA/TIA 449/530 and/or EIA/TIA 612/613 (HSSI) physical interfaces.

Clock rates are left as an implementation option.

The HSSI specification allows for clock "gapping". This feature can be used to control the flow of data to the DCE.

2.3 THE DATA LINK LAYER

2.3.1 Overview

The Data Link Layer defines the method by which the DXI Frames and their associated addressing (DFA) are formatted for transport over the physical layer between the DTE and DCE. The protocol is dependent on the mode selected, as described below.

2.3.2 Mode 1a

For data transmission to the ATM network, the DTE generates the DXI header which contains the DFA, encapsulates the DTE SDU into a DXI (Data Link) Frame, and transports the resulting frame to the DCE. The DCE strips off the DXI Frame encapsulation to gain access to the DTE SDU and associated DFA. The DCE, then, encloses the DTE's SDU in an AAL5 CPCS_PDU and segments the result into AAL5 SAR_PDUs. The DCE maps (as noted in Fig. 2.8) the DFA to the appropriate VPI/VCI thus forming the ATM cells. The DCE, then, transmits the resulting ATM cells to the ATM network.

For data transmission from the ATM network to the DTE the reverse process is followed.

Figure 2.6 shows the protocol stacks for the above described process. Figure 2.2 shows the process.

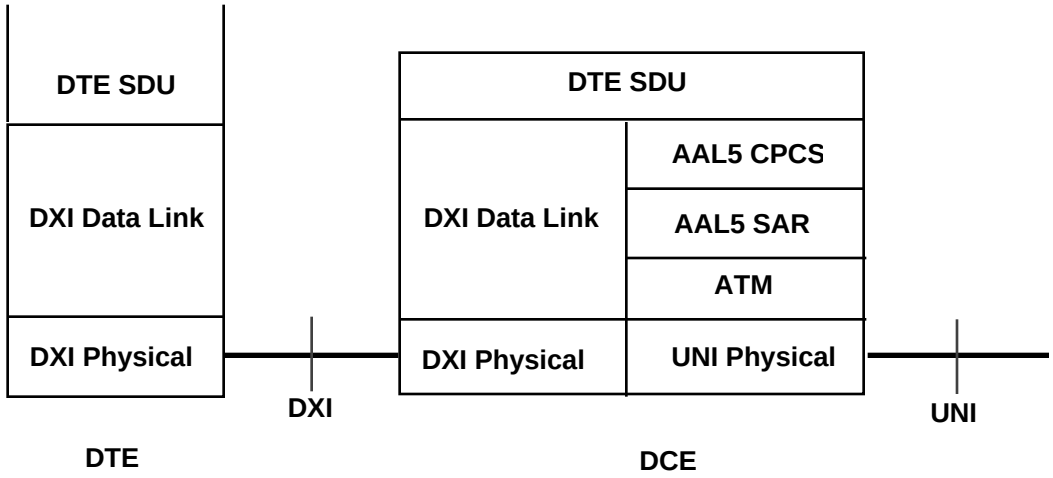


Figure 2.6 Modes 1a and 1b Protocol Architecture for AAL5

2.3.2.1 DXI Frame Encapsulation

In Modes 1a and 1b, when using AAL5, the DTE SDU is encapsulated with the DXI frame header and trailer as shown in Figure 2.7.

In Mode 1b, when using AAL3/4, the AAL3/4 CPCS_PDU is encapsulated with the DXI frame header and trailer as shown in Figure 2.7a.

(The intent of this encapsulation is to emulate standard Frame Relay encapsulation. Using it as the basis advances the primary goal of the ATM DXI effort: to provide an expedient ATM Access method, while requiring minimal changes to the installed base of equipment).

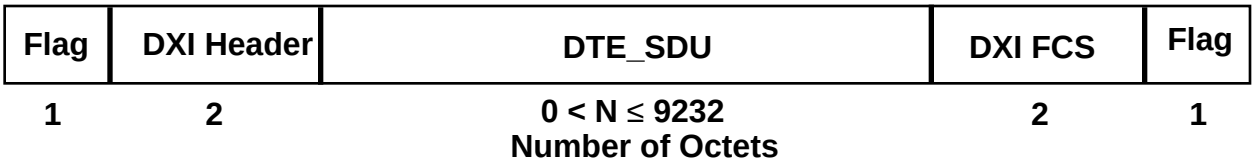


Figure 2.7 Modes 1a and 1b Data Link Frame for AAL5

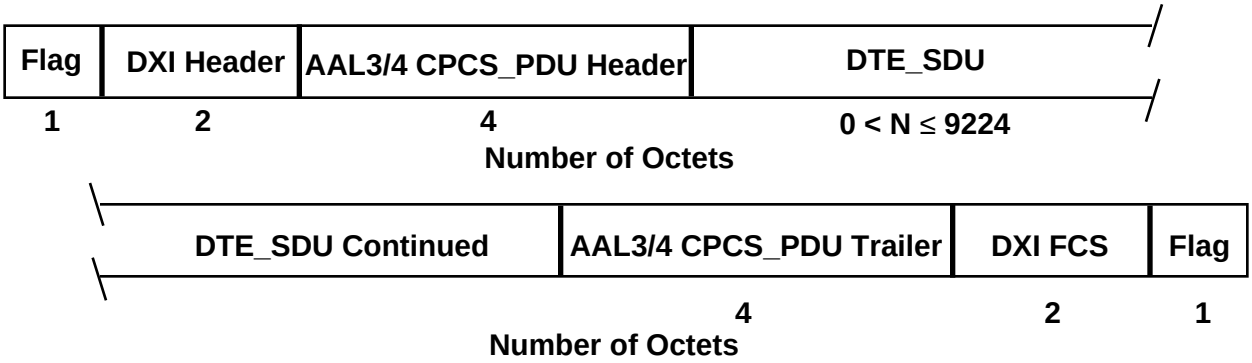


Figure 2.7a Mode 1b Data Link Frame for AAL3/4

The flag, DXI header and DXI trailer fields are shown in more detail in Figure 2.8.

Bit	8	7	6	5	4	3	2	1	Octet
Flag	0	1	1	1	1	1	1	0	0
DXI Header	DFA						RSVD	0	1
	DFA				CN	RSVD	CLP	1	2
DXI Trailer (FCS)	2^8	2^9	2^10	2^11	2^12	2^13	2^14	2^15	n-1
	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	n
Flag	0	1	1	1	1	1	1	0	n+1

RSVD Reserved - These bits are set to 0.

DFA Octet 1, bit 6 through bit 3 represent the 4 LSBs of the VPI. The four MSBs of the VPI are set to zero by the DCE on send and ignored on receive. They are not coded in the DFA field. Octet 1, bit 8 and bit 7, and Octet 2, bit 8 through bit 5, represent the six LSBs of the VCI. The ten MSBs of the VCI are set to zero on by the DCE send and ignored on receive, and are not coded into the DFA field. The all 0s DFA is not translated to the corresponding VPI/VCI and is not transmitted over the UNI. See Annex A for the bit-to-bit mapping between DFA and VPI/VCI.

FCS 16 bit FCS (Frame Check Sequence) is shown, CCITT Q.921 CRC16 is supported. (Note: Reasonable undetected error rates for CRC16 occur at data sizes under 4 kilobytes. For larger sizes, higher layer protocols can be used).

CN If PTI = 01x in the last ATM cell composing the DXI frame, then DCE sets CN (Congestion Notification) equal to one for that DXI frame, otherwise the DCE sets CN equal to zero. The DTE always sets the CN to zero.

CLP The DCE copies the CLP Bit sent from the DTE into the ATM cell header CLP bit. The CLP bit from the DCE to the DTE is always set to zero.

NOTE: The left most bit of the octet (i.e., bit 8) is the Most Significant Bit (MSB). In diagrams with multiple octets the left most bit in the top most octet is the MSB. These bits are transmitted right to left, top to bottom.

Figure 2.8 Modes 1a and 1b Data Link Frame Details

2.3.2.2 Mode 1b.

This mode consists of Mode 1a plus the addition of supporting AAL3/4 on, at least one, individually configurable virtual connections. When processing AAL3/4 in this mode (refer to Figure 2.3), the DTE forms the AAL3/4 CPCS Protocol Data Units, generates the associated address, encapsulates these in the DXI Frame and transports the resulting frame to the DCE. The DCE strips off the DXI Frame encapsulation to gain access to the AAL3/4 CPCS_PDU and associated DFA. The DCE, then, segments the AAL3/4 CPCS_PDU into AAL3/4 SAR_PDUs and translates the DFA to the appropriate VPI/VCI thus forming the ATM cells and performs the appropriate ATM UNI processing then transmits the resulting ATM cells to the ATM network.

For data transmission from the ATM network to the DTE, the reverse process is followed.

Figure 2.9 shows the protocol stacks for the AAL3/4 process described above. Figure 2.3 shows the process. For AAL5 processing, the protocol stack is shown in Fig 2.6 and the process is shown in Figure 2.2.

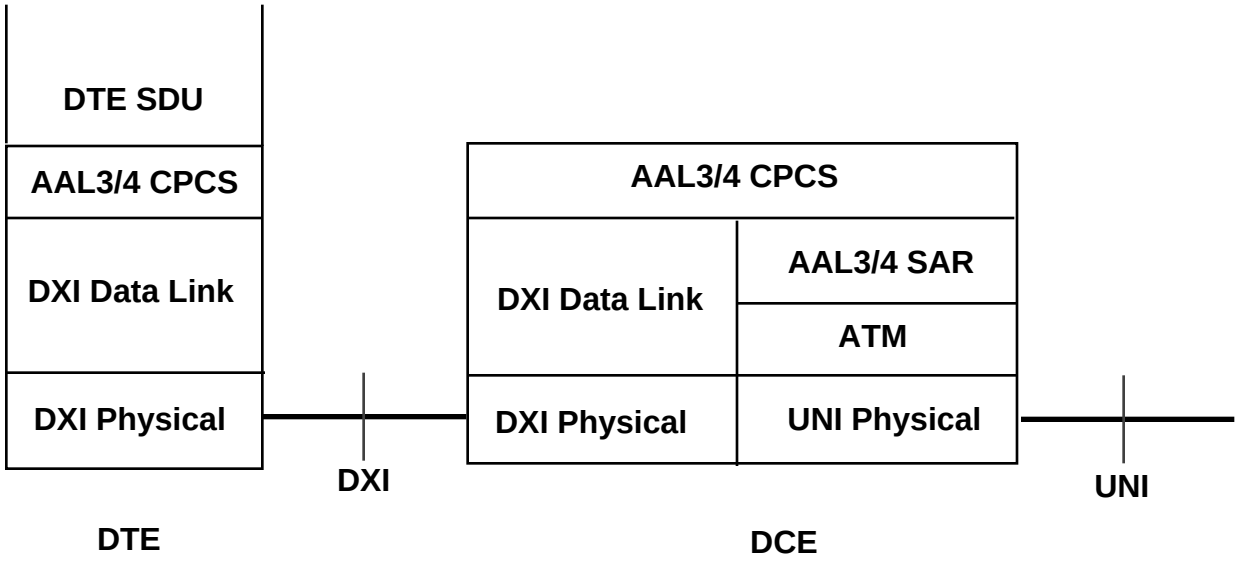


Figure 2.9 Mode 1b Protocol Architecture for AAL3/4

2.3.3 Mode 2

For VPI/VCI values indicating an AAL5 Virtual Connection, the DTE forms the AAL3/4 CPCS_PDU, generates the associated address, encapsulates these in a DXI Frame and transports the resulting frame to the DCE. The DCE strips off the DXI Frame encapsulation to gain access to the AAL3/4 CPCS_PDU and associated DFA. The DCE, then, strip off the header/trailer of the AAL3/4 CPCS_PDU, but retain the DFA, and encapsulates the DTE SDU within an AAL5 CPCS_PDU. Then the AAL5 SAR functions are performed. The DCE translates the DFA to the appropriate VPI/VCI, as noted in Figure 2.13. The DCE then performs the ATM and physical layers processing.

For data transmission from the ATM network to the DTE the reverse process is followed. Figure 2.10 shows the protocol stacks. Figure 2.3 shows the process.

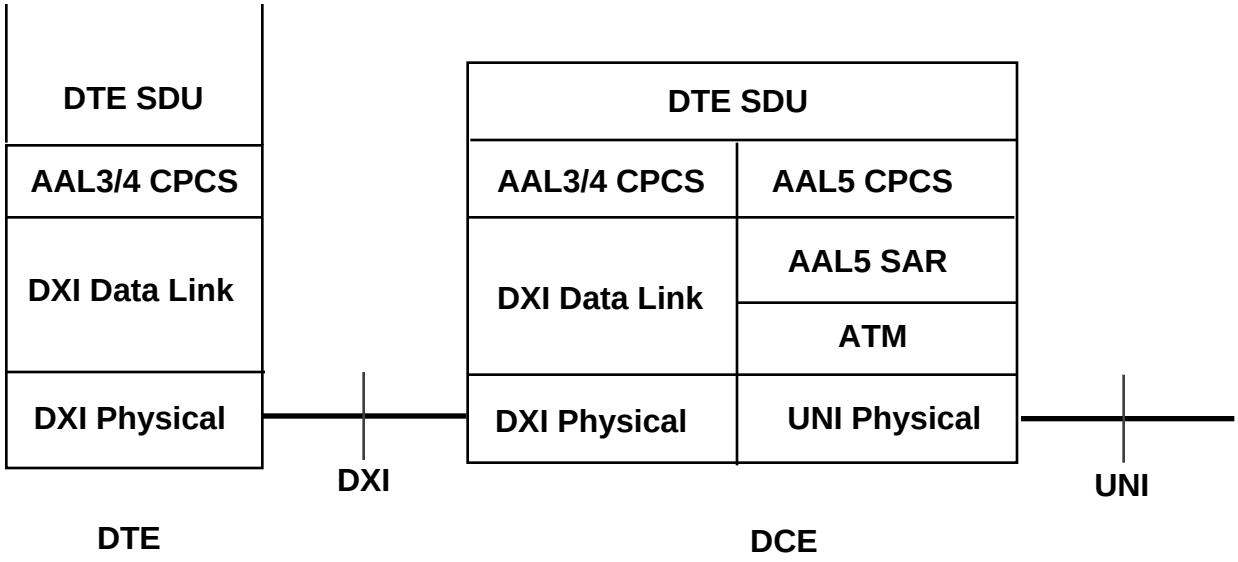


Figure 2.10 Mode 2 Protocol Architecture for AAL5

For VPI/VCI values indicating an AAL3/4 Virtual Connection, the DTE forms the AAL3/4 CPCS_PDU, generates the associated address, encapsulates these in a DXI Frame and transports the resulting frame to the DCE. The DCE strips off the DXI Frame encapsulation to gain access to the AAL3/4 CPCS_PDU and associated DFA, then, performs the AAL3/4 SAR functions. The DCE, then, translates the DFA to the appropriate VPI/VCI, as shown in Figure 2.13, followed by performing the ATM and physical layers processing.

For data transmission from the ATM network to the DTE the reverse process is followed. Figure 2.11 shows the protocol stacks. Figure 2.4 shows the process.

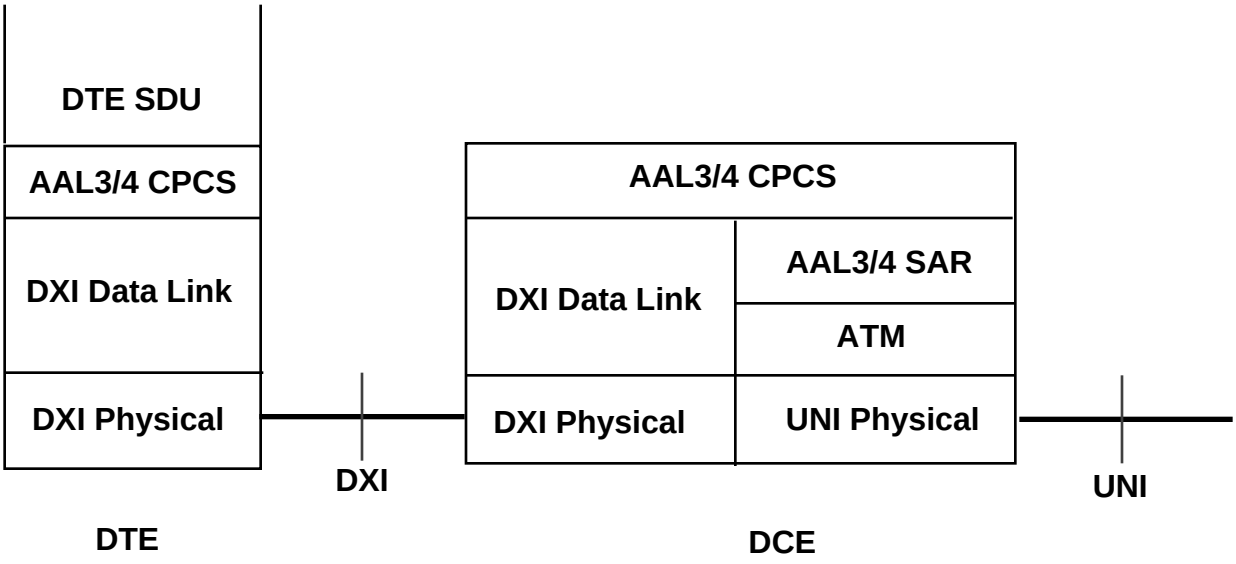


Figure 2.11 Mode 2 Protocol Architecture for AAL3/4

2.3.3.1 DXI Frame Encapsulation

The PDU is encapsulated with the DXI frame header and trailer as shown in Figure 2.12.

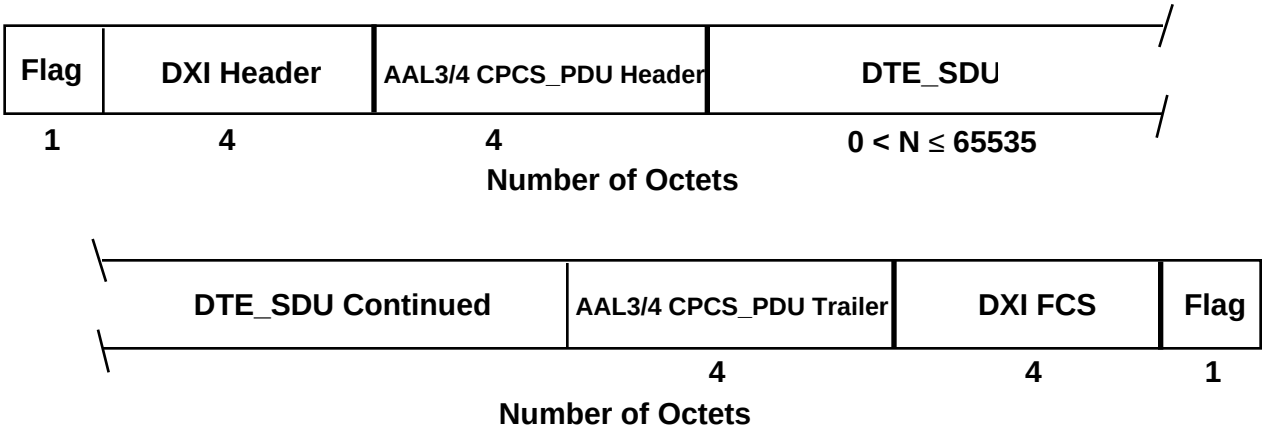


Figure 2.12 Mode 2 Data Link Frame

The flag, DXI header and DXI trailer fields are shown in more detail in Figure 2.13.

	Bit	8	7	6	5	4	3	2	1	Octet
	Flag	0	1	1	1	1	1	1	0	0
DXI Header		DFA						RSVD	0	1
		DFA				CN	RSVD	CLP	0	2
		DFA							0	3
		DFA							1	4
DXI Trailer (FCS)		2^24	2^25	2^26	2^27	2^28	2^29	2^30	2^31	n-3
		2^16	2^17	2^18	2^19	2^20	2^21	2^22	2^23	n-2
		2^8	2^9	2^10	2^11	2^12	2^13	2^14	2^15	n-1
		2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	n
	Flag	0	1	1	1	1	1	1	0	n+1

RSVD Reserved - These bits are set to 0.

DFA Octet 1, bit 8 through bit 3, and Octet 2, bit 6 and bit 5 represent the VPI. Octet 2, bit 8 and bit 7, and Octet 3, bit 8 through bit 2, and Octet 4, bit 8 through bit 2, represent the VCI. The all 0s DFA is not translated to the corresponding VPI/VCI and is not transmitted over the UNI. (See Annex A for the bit-to-bit mapping between DFA and VPI/VCI).

FCS 32 bit FCS is shown. (Ref: ISO 9314-2;1989 (E)).

CN If PTI = 01x in the last ATM cell composing the DXI frame, then DCE sets CN (Congestion Notification) equal to one for that DXI frame, otherwise the DCE sets CN equal to zero. The DTE always sets the CN to zero.

CLP The DCE copies the CLP Bit sent from the DTE into the ATM cell header CLP bit. The CLP bit from the DCE to the DTE is always set to zero.

NOTE: The left most bit of the octet (i.e., bit 8) is the Most Significant Bit (MSB). In diagrams with multiple octets the left most bit in the top most octet is the MSB. These bits are transmitted right to left, top to bottom.

Figure 2.13 Mode 2 Data Link Frame Details

3.0 LOCAL MANAGEMENT INTERFACE

3.1 INTRODUCTION

The ATM Data Exchange Interface (DXI) Local Management Interface (LMI) operates in conjunction with the DXI and it defines the protocol for exchanging management information across the DXI. The ATM DXI LMI supports exchange of DXI-specific, AAL-specific, and ATM UNI-specific management information. The DXI is the interface between a Data Terminal Equipment (DTE, e.g., a router or hub) and the ATM Data Communications Equipment (DCE, e.g., a Data Service Unit (DSU)). The ATM UNI is the User Network Interface originated and terminated at the ATM DCE.

This definition of the LMI assumes a one to one relationship between the DXI and UNI. Other configurations require other management techniques.

The ATM DXI LMI is designed to support a management station running Simple Network Management Protocol (SNMP) and/or switch running ILMI (Interim Local Management Interface). Therefore, it is assumed that an SNMP proxy agent resides in the DTE which is responsible for determining when to query the ATM DCE in response to an SNMP request received from the management station. It is also assumed that an ILMI proxy agent resides in the DTE which is also responsible for determining when to query the ATM DCE in response to an ILMI request received from the switch. The DXI LMI protocol is a simple, adaptable, and generic means of providing the end user with network management information that meets the needs of the application while minimizing effort and delay in realizing these requirements. This approach to the LMI does not imply compliance to the SNMP network management requirements.

3.2 LOCAL MANAGEMENT INTERFACE (LMI) PROTOCOL

This section defines the contents of the information field of the management Logical Link.

There are five defined LMI Protocol Data Units (PDU) types: GetRequest, GetNextRequest, SetRequest, GetResponse, and Trap. The definition assumes that requests originate in the DTE and the DCE responds. The DCE only originates Trap messages. GetRequest, GetNextRequest, and SetRequest messages initiate GetResponses from the DCE. The Trap is autonomously sent from the DCE to the DTE in Alert conditions.

3.2.1 Order of Bit Transmission

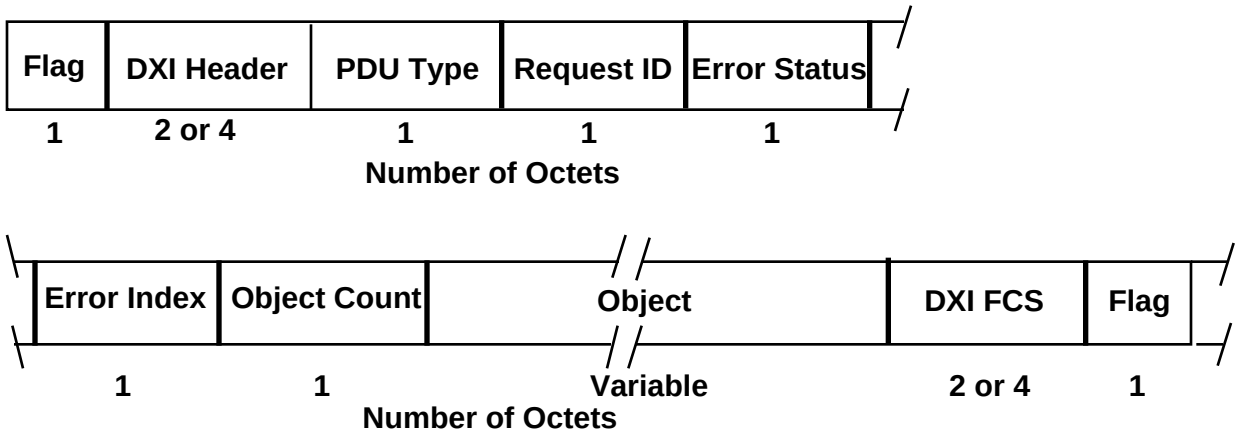
The right most or least significant bit (LSB) of each octet, is the first bit transmitted and the most significant octet is the first to be transmitted.

3.2.2 Non-Trap PDU Structure

The non-Trap PDU types have the following overall structure. All fields are binary encoded unsigned integers except where specified otherwise.

FIELD NAME	LENGTH
PDU type	1 octet
Request ID	1 octet
Error Status	1 octet
Error Index	1 octet
Object Count	1 octet
Object(s)	Variable

The LMI_PDU is encapsulated with the DXI frame header and trailer as shown in Figure 3.1.



NOTES: 1. Modes 1a and 1b use 2 (two) octets in the DXI HDR and DXI FCS fields. Mode 2 uses 4 (four) octets in those fields.

2. The DFA is zero.

Figure 3.1 LMI PDU Structure

The PDU may contain one or more objects, up to the limits imposed by the maximum PDU size or 255, whichever is fewer. The Object Count field specifies the number of objects which follow. Each object is variable in length and has the following structure.

FIELD NAME	LENGTH
Object ID Length	Variable (one octet minimum)
Object ID	Variable
Object Value Tag	One octet
Object Value Length	Variable (one octet minimum)

Object Value Variable

The length fields for the ID and value specify the number of octets that follow in the Object ID and Object value fields respectively. A length field value of zero is legal and specifies that a null field (no octets) follows. If the length is less than 128, the definite short length encoding is used (from the Basic Encoding Rules for ASN.1). If the length is greater than or equal to 128, the definite long length encoding is used. The definite short length encoding is illustrated below in Figure 3.2 and the definite long length encoding is illustrated below in Figure 3.3.

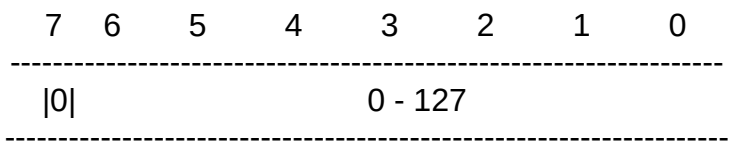
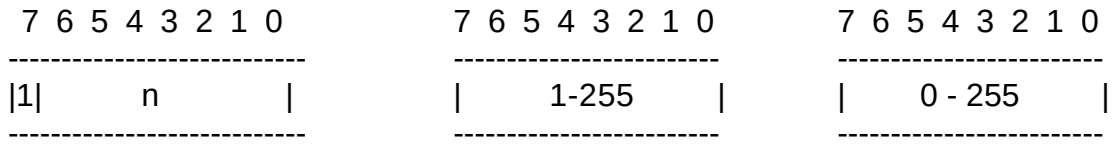


Figure 3.2



n is the number of subsequent octets used to denote the length.

Figure 3.3

As an example, to encode the length 1024, three bytes would be needed as illustrated by Figure 3.4 below.

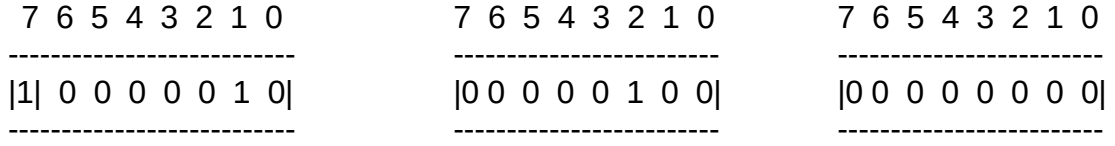


Figure 3.4

The Object ID is represented as an ordered list of encoding of subidentifiers concatenated together with each subidentifier represented as a series of one or more octets. In most cases, a subidentifier is less than 128 and can be encoded in a single octet as a binary number. If a subidentifier is greater than or equal to 128, however, then it is split into a series of 7 bit pieces, each in its own octet. The most significant bit

of each octet within the series indicates whether it is the last in the series, where MSB=0 indicates the terminal octet and all other octets have the MSB=1. The first octet of each subidentifier series must not equal 80H.

As shown in the example illustrated by Figure 3.5, if a subidentifier had the value 129, two bytes are required to encode this value.

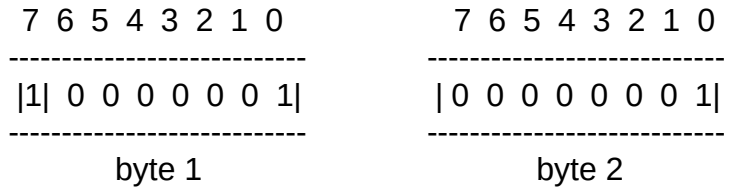


Figure 3.5

Bit 7 of byte 1 indicates that there are more bytes to follow. Bit 7 of byte 2 indicates that it is the last byte for this subidentifier. Concatenating the two 7-bit pieces (and compressing out the leading zeroes), we end up with the value 129 Decimal represented as shown in Figure 3.6.

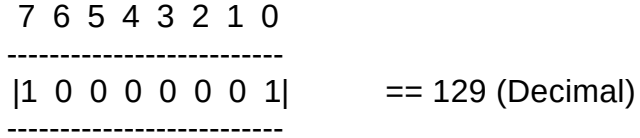


Figure 3.6

Note that each object may have an odd or even number of octets, and that these objects are directly concatenated within the LMI PDU. They are not aligned to any particular octet boundary. The format of the ID and value fields is described later in this section.

The Object value should be represented using the ASN.1 tag, length and value elements appropriate to the value. These elements should then be encoded according to the Basic Encoding Rules for ASN.1 .

3.2.2.1 PDU Type Field

VALUE	MEANING
0	GetRequest
1	GetNextRequest
2	GetResponse

3	SetRequest
4	Trap

3.2.2.2 Request ID Field

The Request ID field is used to distinguish between outstanding requests. The Request ID field is one octet and over the sequence of requests sent by the DTE holds a modulo-256 sequence incremented by the DTE. The GetResponse PDU sent by the DSU contains the value of the Request ID field of the request which generated the response.

3.2.2.3 Error Status Field

In request PDUs this field should be set to zero.

In response PDUs, any non-zero value of the Error Status field indicates that an exception in processing an LMI PDU has occurred. The error index indicates the error indicated by the error type and the first object in error. The prescribed Error Status values are as follows.

Error Status = 0 (noError)

No error on processing the request.

Error Status = 1 (tooBig)

The size of the GetResponse PDU would exceed the maximum allowable. The receiver replies with a GetResponse PDU identical in form to the received message except that the Error Status field value is tooBig(1) and the Error Index field is the index of the Object which exceeded the maximum size.

Error Status = 2 (noSuchName)

One of the objects named in the GetRequest or SetRequest message did not exactly match the name of some object in the MIB. The receiver replies with a GetResponse PDU identical in form to the received message except that the value of the error status field is noSuchName(2) and the value of the error index field is the index of the object name component in the received message.

Error Status = 3 (badValue)

The contents of a value field in a set request message, does not manifest a type, length, and value that is consistent with the requested variable. The receiver replies with a GetResponse PDU identical in form to the received message except that the value of the error status field is badValue(3) and the value of the error index field is the index of the object name component in the received message.

Error Status = 5 (genErr)

The PDU Type was not recognized, or one of the objects named cannot be altered or retrieved for reasons not covered by any of the foregoing rules. The receiver replies

with a GetResponse PDU identical in form to the received message except that the value of the error status field is genErr(5) and the value of the error index field is the index of the object name component in the received message.

3.2.2.4 Error Index Field

In request PDUs this field should be set to zero.

In response messages, the Error Index field is used to indicate which of the objects in the message was in error. If one or more objects are in error, the Error Index field will be set to the index (1 to N) of one of the objects in error. If there are no errors, the value will be set to zero. The other error conditions are determined by the Error Status code.

3.2.2.5 Object ID Field

The Object ID field within an object is the OBJECT IDENTIFIER as defined in RFC 1155. This provides a simple method for accessing information relative to the MIB.

3.2.2.6 Object Value Field

The Object Value is encoded according to the Basic Encoding Rules (BER). The Object Value field is normally null (i.e., Object Value Tag = null(5)) for GetRequest and GetNextRequest PDU types.

3.2.3 Trap PDU Structure

The Trap PDU type has the following overall structure. All fields are binary encoded unsigned integers except where specified otherwise.

FIELD NAME	LENGTH
PDU type	1 octet
Generic Trap Type	1 octet
Enterprise Trap Type	1 octet
Object Count	1 octet
Object(s)	Variable

Similar to the non-trap PDU types, the Trap PDU may contain zero or more objects, up to the limits imposed by the maximum PDU size. The Object Count field specifies the number of objects which follow. Each object is variable in length and has the same structure as objects in the non-Trap PDU types.

3.2.3.1 Trap PDU Type Field

The Trap PDU type field value is four (4) as stated in a previous section.

3.2.3.2 Generic Trap Index Field

The Trap PDU provides a simple method for asynchronous problem notification or asynchronous command completion notification. The Generic Trap Type provides the

basic reason for the trap. Defined values for this field are shown below. The value 'enterpriseSpecific' allows additional trap types to be defined through use of the Enterprise Trap Type field.

VALUE	MEANING
0	coldStart
1	warmStart
2	linkDown
3	linkUp
6	enterpriseSpecific

3.2.3.3 Enterprise Trap Type Field

The value 'enterpriseSpecific' in the Generic Trap Type field allows for additional types of trap to be defined. Such additional types can be defined by any enterprise which can be identified by an OBJECT IDENTIFIER value. The value of the Enterprise Trap Type field indicates a particular additional type of trap according to the values defined by a particular enterprise. The particular enterprise is identified by the first object in the Trap Objects field.

3.2.3.4 Trap Objects

The object fields provide more information on the reason the trap was issued. For example, a Loss of Signal at the DS3 layer (ds3AlarmState with ds3receivedLOS) would cause a linkdown trap.

When the value of the Generic Trap Type field is 'enterpriseSpecific', the first Object ID-Value pair in the Objects field has an ID of atmDxiEnterprise.0, and a Value which identifies the enterprise. For example, the value 1.3.6.1.4.1.353 would identify the ATM Forum.

3.3 LOCAL MANAGEMENT INTERFACE (LMI) PDU TYPES

The SNMP or ILMI proxy agent is resident on the DTE. When it receives an SNMP or ILMI command, it determines whether the information resides in itself or in the DCE (see Section 4). If in the DCE, the DTE issues an appropriate request LMI PDU to the DCE. Upon receiving the request the DCE parses the LMI-PDU and replies with a GetResponse.

Traps are issued autonomously by the DCE in response to an extraordinary event such as an Alarm State.

3.3.1 GetRequest

A GetRequest message originates from the DTE. The Request ID field is incremented (mod 256) from the previous request message sent to the DCE. The Object count field indicates the number of objects in the GetRequest PDU. The Object ID field of each

object specifies an object (variable) for which a value is to be retrieved. The Object ID Length field indicates the number of octets occupied by the Object ID field. The Object ID field contains the full SNMP Object ID. The Object Value field is not required in a GetRequest message and is not included, i.e., the Object Value Tag is null(5) and the Object Value Length field is zero with no octets following within the object.

Upon receipt of a GetRequest PDU the DCE responds according to the following:

1) If the PDU type is not defined, the DCE responds with a GetResponse PDU identical in form to the GetRequest except that the value of the Error Status field is genErr(5) and the value of the Error Index field is zero.

2) A GetResponse header is built, with the Request ID field and Object Count fields copied from the GetRequest PDU. For each Object ID (variable) in the request, the object's value is retrieved, and the ID and value are together appended as a object to the response PDU.

3) If any Object ID is not in the currently defined information base, the DCE responds with a GetResponse identical to the GetRequest PDU except that the Error Status field is noSuchName(2) and the value of the Error Index is the index (1 to N) of one of the Object IDs in error.

4) If the size of the GetResponse PDU would exceed the maximum allowable for a DCE implementation, the DCE replies with a GetResponse PDU identical in form to the received message except that the error status field value is tooBig(1) and the Error Index field value will be the index of the Object which would have exceeded the buffer size.

5) If some other error occurs, the DCE responds with a GetResponse identical to the GetRequest PDU except that the Error Status field is genErr(5) and the value of the Error Index is zero.

6) If after assembling all Objects into the response no errors are found, the GetResponse PDU Error Status field is set to noError(0), the Error Index is set to zero, and the GetResponse PDU is sent to the DTE.

3.3.2 GetNextRequest

A GetNextRequest message originates from the DTE. The Request ID field is incremented (mod 256) from the previous request message sent to the DCE. The Object count field indicates the number of objects in the GetRequest PDU. The Object ID field of each object specifies an object (variable) for which the next object in the MIB (according to SNMP's rules for ordering of objects within the MIB) and its value are to be retrieved. The Object ID Length field indicates the number of octets occupied by the Object ID field. The Object ID field contains the full SNMP Object ID. The Object Value field is not required in a GetNextRequest message and is not included, i.e., the Object Value Tag is null(5) and the Object Value Length field is zero with no octets following within the object.

Upon receipt of a GetNextRequest PDU the DCE responds according to the following:

1) If the PDU type is not defined, the DCE responds with a GetResponse PDU identical in form to the GetNextRequest except that the value of the Error Status field is genErr(5) and the value of the Error Index field is zero.

2) A GetResponse header is built, with the Request ID field and Object Count fields copied from the GetNextRequest DCE. For each Object ID in the request, the next accessible object in the information base is retrieved along with its value. The new ID and its value are together appended as a object to the response PDU.

3) If any Object ID is not in the currently defined information base, the DCE looks for the first accessible object in the information base whose ID is greater than that of the given ID according to the ordering rules of objects within the MIB. If such an accessible object exists, its ID and value are appended to the response as before. If none exists, the DCE responds with a GetResponse identical to the GetRequest PDU except that the Error Status field is noSuchName(2) and the value of the Error Index is the index (1 to N) of the object containing the Object ID in error.

4) If the size of the GetResponse PDU would exceed the maximum allowable for a DCE implementation, the DCE replies with a GetResponse PDU identical in form to the received message except that the error status field value is tooBig(1) and the Error Index field value will be the index of the Object which would have exceeded the buffer size.

5) If some other error occurs, the DCE responds with a GetResponse identical to the GetNextRequest PDU except that the Error Status field value is genErr(5) and the value of the Error Index is zero.

6) If after assembling all Objects into the response no errors are found, the GetResponse PDU Error Status field is set to noError(0), the Error Index is set to zero, and the GetResponse PDU is sent to the DTE.

3.3.3 GetResponse

The GetResponse originates from the DCE in response to a GetRequest, GetNextRequest, or SetRequest message. The construction of the GetResponse PDU is detailed in sections 3.3.1, 3.3.2 and 3.3.4.

3.3.4 SetRequest

A SetRequest message originates from the DTE. The Request ID field is incremented (mod 256) from the previous request message sent to the DCE. The Object count field indicates the number of objects in the SetRequest PDU. The Object ID field of each object specifies an object (variable) for which a value is to be set (modified). The Object ID Length field indicates the number of octets occupied by the Object ID field. The Object ID field contains the full SNMP Object ID. The Object Value field is the new value to which the object is to be set. The length and format of the object value should be according to the object's description in the MIB.

Upon receipt of a SetRequest PDU the DCE responds according to the following:

1) If the PDU type is not defined, the DCE responds with a GetResponse PDU identical in form to the SetRequest except that the value of the Error Status field is genErr(5) and the value of the Error Index field is zero.

2) A GetResponse header is built, with the Request ID field and Object Count fields copied from the SetRequest PDU. For each Object in the request, a set (write, modify) function is attempted on the given Object ID using the given Value. If successful, the same object ID and value are together appended as an object to the response PDU.

3) If any Object ID is not in the currently defined information base, the DCE responds with a GetResponse identical to the SetRequest PDU except that the Error Status field is noSuchName(2) and the value of the Error Index is the index (1 to N) of one of the Objects in error.

4) If for a given Object, the Object ID cannot be altered because it is designated as non-accessible or read-only, the DCE replies with a GetResponse PDU identical in form to the received message except that the Error Status field value is noSuchName(2) and the Error Index field value is the index of the first Object in error.

5) If for a given Object the contents of the Object value field does not manifest a type, length, and value that is consistent with the requested variable, the DCE replies with a GetResponse PDU identical to the received message except that the value of the Error Status field is badValue(3) and the value of the Error Index field is the index of one of the Objects in error.

6) If the size of the GetResponse PDU would exceed the maximum allowable for a DCE implementation, the DCE replies with a GetResponse PDU identical in form to the received message except that the error status field value is tooBig(1) and the Error Index field value will be the index of the Object which would have exceeded the buffer size.

7) For a SetRequest that specifies multiple objects, all must be set simultaneously. If one or more cannot be set, then none of the objects are set and the DCE responds with a GetResponse identical to the SetRequest PDU except that the Error Status field value is noSuchName(2) and the value of the Error Index is the index (1 to N) of one of the Objects in error.

8) If after setting and assembling all objects into the response no errors are found, the GetResponse PDU Error Status field is set to noError(0), the Error Index is set to zero, and the GetResponse PDU is sent to the DTE.

3.3.5 Trap

Traps are sent to the DTE in the case of a defined alarm condition. The standard traps defined by values of the Generic Trap Type field are as follows.

3.3.5.1 coldStart Trap

A coldStart(0) trap signifies that the DCE is initializing itself such that the DCE configuration may have been altered.

3.3.5.2 warmStart Trap

A warmStart(1) trap signifies that the DCE is reinitializing itself such that the DCE configuration has not been altered.

3.3.5.3 linkDown Trap

A linkDown(2) trap signifies that the DCE's ATM UNI is out of service. The PDU also contains the reason for the linkDown (such as an Alarm State) and the index (ifIndex) of the ATM UNI affected.

3.3.5.4 linkUp Trap

A linkUp(3) trap signifies that the DCE's ATM UNI has been out of service and has returned to service. The PDU also contains the index (ifIndex) of the ATM UNI, if present.

3.3.5.5 enterpriseSpecific Trap

An enterpriseSpecific(6) trap signifies that the DCE recognizes some enterprise specific event has occurred. The Enterprise Trap Type field identifies which enterprise specific event occurred, according to the values defined by the particular enterprise identified by the first object in the Objects field.

4. ATM DXI LMI MIBs

ATM DXI LMI requests from the DTE to the DCE will be stimulated by more global management requests to the DTE. For example, the ATM switch could send an ILMI request to the DTE, or a network management station could send an SNMP request to the DTE. Some of these requests (e.g., for a count of dropped received cells) must be referred to the DCE, as DXI LMI requests. Likewise, ATM DXI LMI traps from the DCE to the DTE could stimulate ILMI traps to the switch or SNMP traps to a distant network management station.

This relationship of ATM DXI LMI with SNMP and ILMI is shown in Figure 4.1.

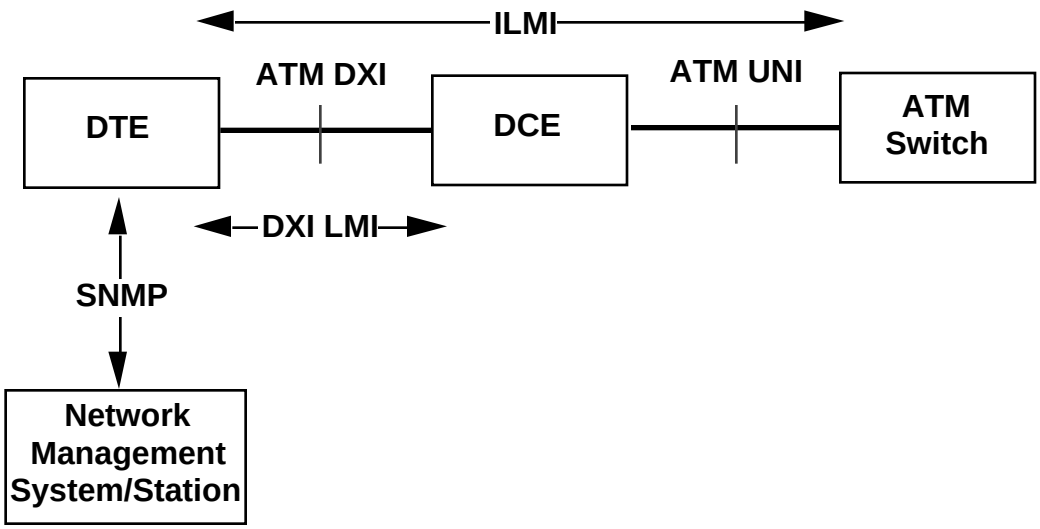


Figure 4.1 ATM DXI LMI

To support more global management mechanisms, such as SNMP and ILMI, the ATM DXI LMI MIB includes at least the MIBs defined in [1]. The MIB defined in [4] is to be supported once it is stable. The DXI also supports physical layer MIBs as appropriate. Vendors may extend the ATM DXI LMI MIB as appropriate specifications are completed. Vendors may also extend their MIB in enterprise specific fashions.

The ATM DXI LMI MIB also includes local information about the DXI itself, as specified in Annex B.

4.1. ATM DXI LMI MIB

The ATM DXI LMI MIB is based upon the DTE/DCE configuration shown in Figure 4.2 below. The primary assumption is that each DXI corresponds to one UNI.

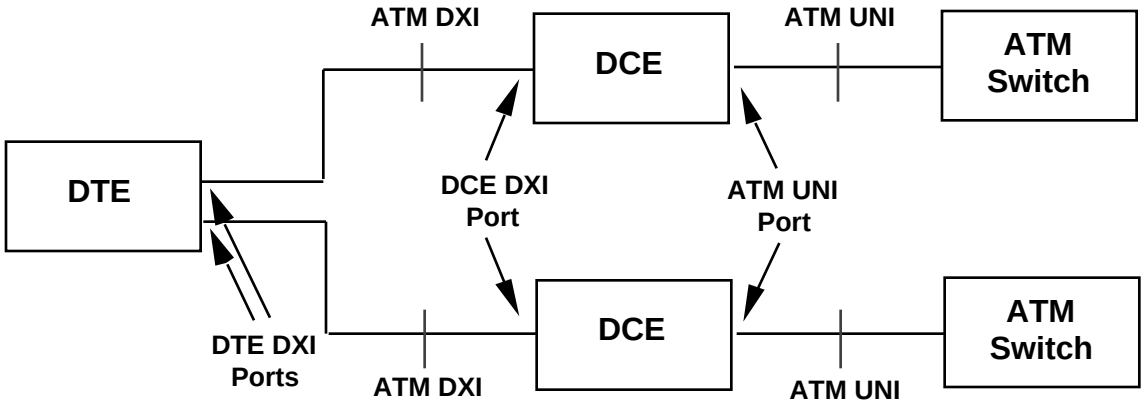


Figure 4.2 DTE/DCE Configuration for the ATM DXI LMI

4.1.1. DCE DXI and ATM UNI Port Management

In order to provide accurate information and control for the DXI link joining a DTE and a DCE, a coherent and thorough method of identifying specific interfaces on specific devices is required.

The conventional method of identifying specific interfaces on specific devices is through the use of transport addresses (the most common example of which is an IP address and UDP port 161), a community-string value, and an index into the interface table from MIB II. By addressing the device at the IP layer, an SNMP management station can get a request to that device. Relative to each device, interfaces can then be identified by numbering them starting from one. SNMP requests can then be sent to a device to examine the interface table, obtaining enough basic information about the interfaces on that device to correlate statistical and configuration information, which is provided relative to an interface number, to physical interfaces.

In order to speed the introduction of manageable DCE to the market, the DTE will act as a proxy for the DCE and relieve it from the burden of participating as an independent IP device. Since the use of the IP layer is no longer an option for identifying the DCE, another mechanism must be found.

4.1.2. MIB-Centric Approach

The proposed approach is to extend the interface indices relative to the DTE to identify interfaces on the DCE. For example, SNMP requests sent to DTE 1 (see the diagram below) might use the following interface indices shown in Figure 4.3 below.

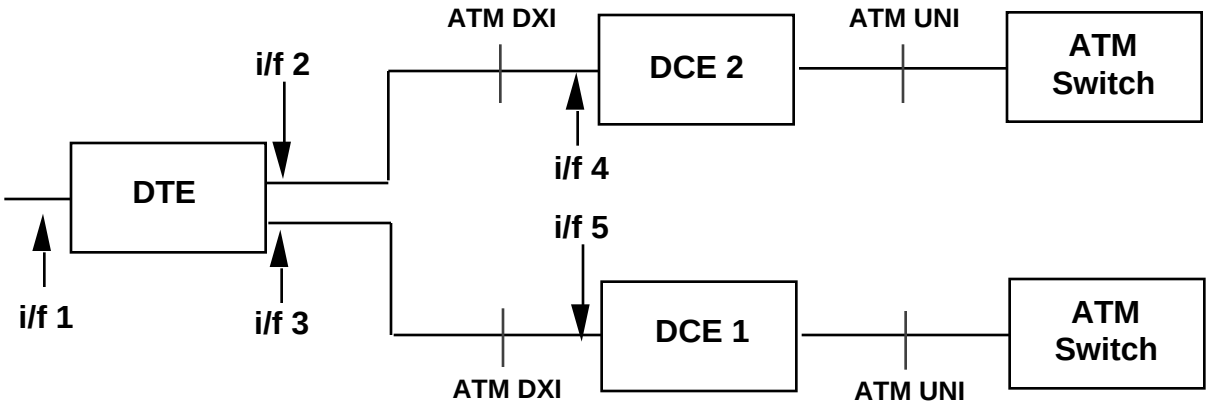


Figure 4.3 Example of Interface Indices for MIB-Centric Approach

(Note, this is almost identical to the indexing scheme described in the DS1/DS3 MIBs).

These extra interfaces will appear to management to be interfaces on the responding DTE, and will be assigned index values by the DTE. In order to simplify the problems

associated with numbering, and the complexities of the DCE, the DCE shall ignore the IfIndex used on any request sent to it. No matter what value of IfIndex is included in the request, the DCE shall return information relative to the interface over which it received the request.

Using this indexing scheme, a management station would send an SNMP request for foo.4 to DTE 1 to get values of the statistic "foo" relative to i/f 4 in the diagram above. In the background, transparent to the management station, the DTE will send a DXI LMI request to DCE 2 (the DCE containing corresponding to the interface for which information was requested) for the exact objects requested in the SNMP request from the management station. (Note that if the DTE has two interfaces to the same DCE, it must send the SNMP request over the one associated with the requested interface, as the DCE will not examine that portion of the request). Once the response is received from the DCE, the DTE, in turn, will send an SNMP response to the management station. (Note, in order to reduce the amount of manual configuration required in the DTE, it is highly desirable that the Object Identifiers used in the SNMP request be identical to the Object Identifiers used in the DXI LMI request. In other words, the ATM DXI LMI MIB is identical in form to the SNMP MIB).

This mechanism works even when there are multiple interfaces on the DCE from multiple DTEs. Following is an example. Suppose the same management station wanted to get information about the interfaces relative to DTE 2. Using this scheme, those interfaces may be numbered as shown in Figure 4.4 below. In reading the diagram, note that I/F represents interfaces to DTE 2, and i/f represents interfaces to DTE 1 (Keep in mind that the DTE assigns the IF index to the DCE interface).

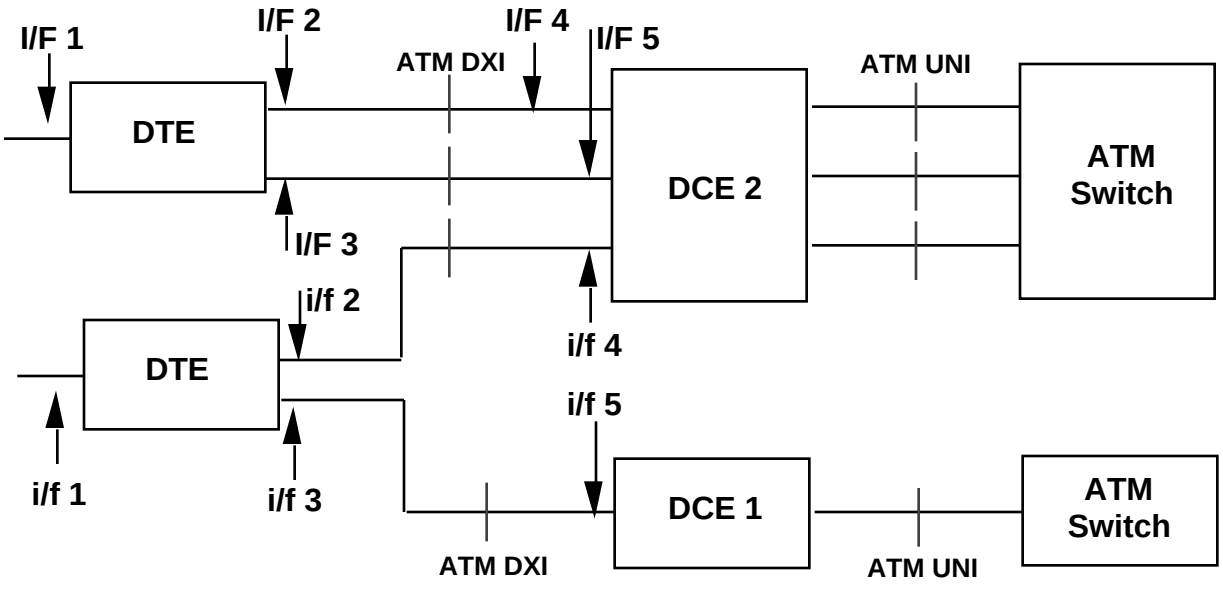


Figure 4.4 Example of Numbering Multiple Interfaces on a DCE from Multiple DTEs

In order to get the value of statistics "foo" relative to I/F 4, the management station would send an SNMP request for foo.4 to DTE 2. If the management wanted to get information about i/f 4 (note, this is relative to DTE 1), the management station would send an SNMP request for foo.4 to DTE 1. When the request arrives at the DCE, one might anticipate confusion in that both requests specify foo.4. However, the DCE ignores the 4, and returns the foo information for interface over which it receives the request. For the purposes of DXI, this value is considered 0.

4.1.3 The Location of Information

To utilize the ATM DXI MIB, it is necessary to understand what data structures are used to reflect certain information. Obviously, the DTE maintains all of the SNMP information for its own side of the DXI interface in the MIB associated with that interface. It also assigns the IfIndex for the DCE side interface. The DTE also maintains the IfSpecific value for the DCE interface. This Object ID shall point to the AtmDxiConfIfIndex.i (where i is the DTE assigned IfIndex of the DCE port). The value of the IfSpecific field of the DTE interface information will be standardized by the IETF.

The DCE shall maintain all of the statistics and information called for by the ifTable in RFC 1213, which includes packets sent and received, errors, etc.

There are two other "tables" maintained by the DCE. Both are indexed by IfIndex. These maintain information about the ATM UNI associated with the DXI interface (as stated earlier, there is a one-to-one relationship between each DXI and ATM UNI). Therefore, from the point of view of SNMP operations, these structures are indexed by the same IfIndex as the associated DXI interface. The fact that the DCE will ignore the index on requests is not visible to management.

The atmDxiConfTable contains only the atmDxiConfMode, to store the mode of operation of the interface. The atmDxiDFAConfTable is indexed by the atmDxiDFAConfDfaIndex, and stores the AAL for each DFA.

5. REFERENCES

[1] ATM Forum, ATM UNI Specification, Version 2.0, June 1992;
Section 4.6, "Actual MIB."

[2] RFC 1407, Definitions of Managed Objects for the DS3/E3 Interface Type;
Section 4, "Object Definitions."

[3] Internet Draft (Work In Progress), Definitions of Managed Objects for the SONET Interface Type.

[4] Internet Draft (Work In Progress), ATM Management Information Base (MIB).

[5] Bellcore Technical Advisory, "Broadband ISDN Switching System Generic Requirements", TA-NWT-001110, Issue 1, August 1992.

ANNEX A
ATM DATA EXCHANGE INTERFACE (DXI) ADDRESS MAPPINGS

MODES 1A AND 1B

10-Bit DFA to VPI/VCI Mapping

See Figure 2.8 in this document and Figure 3.4 in [1].

DFA		VPI	
Octet	Bit	Octet	Bit
16	2	8	
15	2	7	
14	2	6	
13	2	5	

DFA		VCI	
Octet	Bit	Octet	Bit
18	3	2	
17	3	1	
28	4	8	
27	4	7	
26	4	6	
25	4	5	

MODE 2

24-Bit DFA to VPI/VCI Mapping

See Figure 2.13 in this document and Figure 3.4 in [1].

DFA		VPI	
Octet	Bit	Octet	Bit
18	1	4	
17	1	3	
16	1	2	
15	1	1	
14	2	8	
13	2	7	
26	2	6	
25	2	5	

DFA		VCI	
Octet	Bit	Octet	Bit
28	2	4	
27	2	3	
38	2	2	
3	7	2	1
3	6	3	8
3	5	3	7
34	3	6	
3	3	3	5
3	2	3	4
48	3	3	
47	3	2	
4	6	3	1
45	4	8	
44	4	7	
43	4	6	
42	4	5	

ANNEX B

ATM DATA EXCHANGE INTERFACE (DXI) MIB

ATM-DXI-MIB DEFINITIONS ::= BEGIN

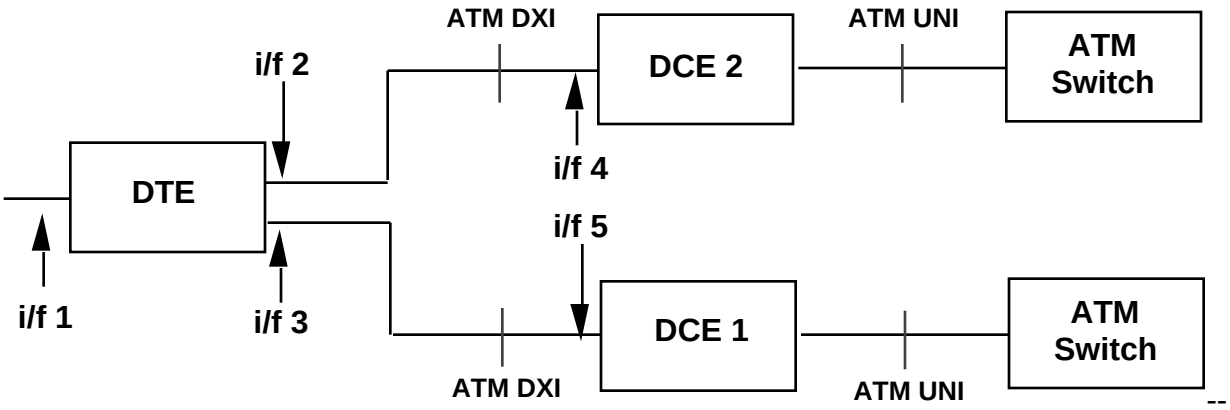
```
IMPORTS

  enterprises, Counter, TimeTicks
    FROM RFC1155-SMI
  OBJECT-TYPE
    FROM RFC-1212;


atmForum        OBJECT IDENTIFIER ::= { enterprises 353 }

atmUniDxi       OBJECT IDENTIFIER ::= { atmForum 3 }

-- This MIB is intended to be used in either of two scenarios.
-- In the first scenario, the DTE acts as a proxy for the DCE(s),
-- while, in the second scenario, the DCE contains an independent
-- SNMP agent and responds to SNMP requests directly.
--
-- In both these scenarios, specific DXI interfaces are identified
-- by a single integer-valued index.  In the first scenario, the
-- DXI interfaces that connect directly to the DTE are identified
-- by the DTE's corresponding value of ifIndex.  The DXI interfaces
-- that connect to the DCE are assigned interface numbers.
-- For example, see the figure below.
-- In this example, the numbers 4 and 5 are assigned to those
-- DXI interfaces that are connected to the DCE(s).  The assignment
-- of numbers to these interfaces is implementation dependent.
-- An SNMP management station will see these as interfaces of a
-- special type (assigned by the IETF).
-- In the second scenario, all the DXI interfaces correspond
-- directly to DCE interfaces.  In this case, all corresponding
-- instances are identified by IfIndices assigned by the DCE.
```


```
-- In the UNI ATM Cell Header, the VPI/VCI fields together
-- comprise a 24-bit field.  The eight most significant bits
-- constitute the VPI subfield, and the least significant
-- sixteen bits constitute the VCI subfield.  All zeros is an
-- invalid value and on ATM interfaces is used to designate a
-- null (or empty or unassigned) cell.  Thus the valid VPI/VCI
-- values are in the range 1 thru 2**24-1 inclusive.  These
-- values are used for indexing certain tables in the ATM
-- MIB(s).

-- The ATM Data Exchange Interface (DXI) object reflects
-- information about an ATM DXI between an ATM DCE (e.g., a DSU)
-- and a DTE (e.g., a Router/Host).  In particular, it contains both
-- configuration and performance information specific to the
-- DXI. Each DXI instance typically corresponds to several DFAs.
-- With a 24-bit DFA, the range of valid DFA index values is
-- 1 thru 2**24 inclusive.  Note that the 24-bit binary DFA
-- field (with values 0 thru 2**24-1) from the DXI
-- header must be incremented by one to yield the DFA index
-- for accessing the tables in the MIB.  The DFA consisting of
-- all zeros (with corresponding MIB index value of 1) is
-- reserved for the DXI Local Management Interface (LMI) with
-- the ATM DTE.

 Dfa ::= INTEGER (1..16777215)

-- *** For the DCEs, the mapping between the DXI
-- *** DFA and the ATM VPI/VCI is fixed.  The LMI DFA has no
-- *** corresponding VPI/VCI. (Refer to Annex A for the bit mapping.)

 atmDxi     OBJECT IDENTIFIER ::= { atmUniDxi 2 }

   ---
```

```
   --- This table is mandatory if supporting ATM DXI.
   ---

atmDxiConfTable   OBJECT-TYPE
   SYNTAX  SEQUENCE OF AtmDxiConfEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
   "This table contains
   ATM DXI configuration information
   including information on the mode supported
   across the DXI."
   ::= { atmDxi 2 }

atmDxiConfEntry  OBJECT-TYPE
   SYNTAX    AtmDxiConfEntry
   ACCESS    not-accessible
   STATUS    mandatory
   DESCRIPTION
   "This list contains ATM DXI configuration information."
   INDEX    { atmDxiConfIfIndex }
   ::= { atmDxiConfTable 1 }

AtmDxiConfEntry  ::=
   SEQUENCE {
   atmDxiConfIfIndex
   INTEGER,
   atmDxiConfMode
   INTEGER
   }

atmDxiConfIfIndex  OBJECT-TYPE
   SYNTAX    INTEGER
   ACCESS    read-only
   STATUS    mandatory
   DESCRIPTION
   "The value of this object identifies, for SNMP, the ATM DXI
   interface for which this entry contains management information.
   This is the same value as used to identify the IfEntry describing
   the DCE interface.  Management uses and expects this value.  In
   the proxy mode of operation, the DCE always treats this as 0, but
   preserves it in its response to the DTE.  0, the DXI value, means
   the interface over which the DXI LMI request was received."
   ::= { atmDxiConfEntry 2 }

atmDxiConfMode OBJECT-TYPE
```

```
      SYNTAX    INTEGER {
            mode1a  (1),
            mode1b  (2),
            mode2   (3)
      }
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
      "This object identifies the dxi mode being
      used at the atm dxi port."
      ::= { atmDxiConfEntry 3 }

   atmDxiDFAConfTable  OBJECT-TYPE
      SYNTAX  SEQUENCE OF AtmDxiDFAConfEntry
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
      "This table contains configuration information about
      a particular DFA."
      ::= { atmDxi 3 }

   atmDxiDFAConfEntry  OBJECT-TYPE
      SYNTAX    AtmDxiDFAConfEntry
      ACCESS    not-accessible
      STATUS    mandatory
      DESCRIPTION
      "This list contains ATM DXI DFA configuration
      information."
      INDEX    { atmDxiDFAConfIfIndex, atmDxiDFAConfDfaIndex }
      ::= { atmDxiDFAConfTable 1 }

   AtmDxiDFAConfEntry  ::=
      SEQUENCE {
         atmDxiDFAConfIfIndex
         INTEGER,
         atmDxiDFAConfDfaIndex
         Dfa,
         atmDxiDFAConfAALType
         INTEGER
      }

   atmDxiDFAConfIfIndex   OBJECT-TYPE
      SYNTAX    INTEGER
      ACCESS    read-only
      STATUS    mandatory
      DESCRIPTION
```

```
      "The value of this object identifies, for SNMP, the ATM DXI
      interface for which this entry contains management information.
      This is the same value as used to identify the IfEntry describing
      the DCE interface.  Management uses and expects this value.  In
      the proxy mode of operation, the DCE always treats this as 0, but
      preserves it in its response to the DTE.  0, the DXI value, means
      the interface over which the DXI LMI request was received."
      ::= { atmDxiDFAConfEntry 1 }

   atmDxiDFAConfDfaIndex   OBJECT-TYPE
      SYNTAX  Dfa
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
      "This object identifies the DFA instance on the DXI
      identified by atmDxiDFAConfIfIndex."
      ::= { atmDxiDFAConfEntry 2 }

   atmDxiDFAConfAALType OBJECT-TYPE
      SYNTAX    INTEGER {
         unknown (1),
         none    (2),
         aal34   (3),
         aal5    (4)
      }
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
      "This object identifies the AAL type supported at this DFA.
      Note, if mode 2 is supported on the DXI identified by the
      corresponding instance of atmDxiDFAConfIfIndex and that
      instance of atmDxiDFAConfIfIndex identifies the DCE side
      of the DXI, then this object contains the AAL Type being
      run on the corresponding VPI/VCI on the corresponding ATM UNI
      interface."
      ::= { atmDxiDFAConfEntry 3 }

   --  The following definition is for use only in Trap PDUs

   atmDxiEnterprise  OBJECT-TYPE
      SYNTAX  OBJECT IDENTIFIER
      ACCESS  not-accessible
      STATUS  mandatory
      DESCRIPTION
         "This object is included as the first ID-Value pair
         in a Trap PDU for which the Generic Trap Type field
```

```
            has the value 'enterpriseSpecific'.  The value of the
            object identifies the enterprise under whose authority
            the value of the Enterprise Trap Type field is defined."
        : := { atmDxi 4 }


    -- End of definitions for ATM DXI-MIB
```

END of ATM DXI Document

