

Specification for

ControlNet

Part 4. Network and transport layer

ICS 25.040.40; 35.100.40

Committees responsible for this Draft for Development

The preparation of this Draft for Development was entrusted to Technical Committee AMT/7, Monitoring and control aspects of AMT, upon which the following bodies were represented:

- ABB Kent plc
- Cegelec Industrial Controls Ltd.
- Department of Trade and Industry
- Fisher-Rosemount Ltd.
- Foxboro Great Britain
- Institution of Electrical Engineers
- Measurement Technology Ltd.
- Phoenix Contact Ltd.
- Process Automation and Computer Systems Ltd.
- Rockwell Automation Ltd.
- Schneider Ltd.
- Siemens Automation Ltd.
- University of Warwick
- Wago Ltd.

This Draft for Development, having been prepared under the direction of the Advanced Manufacturing Technology Assembly, was published under the authority of the Standards Board and comes into effect on 15 November 1997

© BSI 1997

The following BSI reference relates to the work on this publication:
Committee reference AMT/7

ISBN 0 580 28676 2

Summary of pages

This document comprises a front cover, an inside front cover, pages i to iv, pages 1 to 139 and a back cover.

Amendments issued since publication

Amd. No.	Date	Text affected

Contents

	Page
Committees responsible	Inside front cover
Foreword	iii
0 Introduction	1
1 Scope	2
2 Unconnected Message Manager (UCMM)	2
2.1 General	2
2.2 External Interface	3
2.3 PDU format	4
2.4 Client UCMM	6
2.5 Server UCMM	8
2.6 Example (informative)	11
2.7 Keeper UCMM	13
3 Message Router	13
3.1 External Interface	13
3.2 Path	13
4 Connection Manager Service Descriptions	20
4.1 External interface (SDU)	20
4.2 PDU	26
5 Connection Manager Parameters	36
5.1 Scope	36
5.2 Network connection parameters	36
5.3 Packet interval	37
5.4 Connection Time-out Multiplier	38
5.5 Connection request priority/tick time	38
5.6 Connection request time-out ticks	40
5.7 Connection serial number	42
5.8 Vendor ID	42
5.9 Originator serial number	42
5.10 Connection number	42
5.11 Connection path size	42
5.12 Network Connection ID	42
5.13 Transport class and trigger	42
5.14 Error codes	43
6 Transport Connections	51
6.1 What is a transport?	51
6.2 Tools Used to Discuss Transport Services	52
6.3 Components of Transport Connections	52
6.4 Creating Transport Connections	56
6.5 Binding Transports to Producers and Consumers of Network Connections	57
7 Transport Classes	59
7.1 Classes	59
7.2 Transport Class 0 (Null or Base)	59
7.3 Transport Class 1 (Duplicate Detection)	63
7.4 Transport Class 2 (Acknowledged)	70

7.5 Transport Class 3 (Verified)	79
7.6 Transport Classes 4-6 Common Elements	88
7.7 Transport Class 4 (Non-Blocking)	94
7.8 Transport Class 5 (Non-Blocking, Fragmenting)	104
7.9 Transport Class 6 (Multicasting, Fragmenting)	116
7.10 Retry Timers	136
8 Application Connections	138
8.1 Connections	138
8.2 Production trigger, transport class and EPR	138
8.3 Polling	139

Foreword

The DD 241 specification was developed by the Allen-Bradley Company, Inc. as "ControlNet". Allen-Bradley has granted BSI a right to publish the specification as an open communications standard for digital communications networks in industrial control applications.

This part of DD 241 describes the General Description of the deterministic control network.

This publication consists of the following parts:

Part 1:	General description
Part 2:	Physical layer and media
Part 3:	Data link layer
Part 4:	Network and transport layer
Part 5:	Data management
Part 6:	Object model
Part 7:	Device profiles
Part 8:	Electronic data sheet
Part 9:	Station management
Part 10:	Object library

This Draft for Development is not to be regarded as a British Standard.

It is being issued in the Draft for Development series of publications and is of a provisional nature because the subject matter is still under development requiring wider exposure. It should be applied on this provisional basis so that information and experience of its practical application may be obtained.

A review of this Draft for Development will be carried out not later than 2 years after its Publication. Notification of the start of the review period, with a request for the submission of comments from users of this Draft for Development, will be made in an announcement in *Update Standards*.

According to the replies received the responsible BSI Committee will judge whether the Draft for Development can be converted into a British Standard or what other action should be taken.

Observations which it is felt should receive attention before the official call for comments will be welcomed.

These should be sent to the Secretary of BSI Technical Committee AMT/7 at BSI, 389 Chiswick High Road, London, W4 4AL, giving the document reference and clause number and proposing, where possible, an appropriate revision of the text.

Disclaimer of warranties

This document is provided on an "as is" basis without warranty by BSI and Allen-Bradley as to the accuracy or completeness of the information it contains or as to the results which may be achieved through the use of this information. Further, BSI and Allen-Bradley are not providing any warranties as to whether products produced based on DD 241 may infringe the intellectual property rights of any third parties except that, at the date of publication, to the best of BSI's and Allen-Bradley's knowledge, no written claims have been made against BSI and/or Allen-Bradley alleging that any such products would infringe the intellectual property rights of any third parties. In no event shall BSI and/or Allen-Bradley be responsible for any losses, damages, costs or expenses incurred as a result of the information contained in DD 241.

0 Introduction

The role of the Network and Transport Layer is to establish and maintain connections. A connection is similar to a telephone circuit. When a call is placed, the telephone system selects a path for your call the sets up each switching station in the route to handle it. As long as the call continues, the resulting virtual circuit remains open, carrying data or voice traffic. In the telephone system, a single call can travel over multiple, different type links, and the format of the data can change from link to link, but the connection appears the same to both the caller and the party called. Sound in one end becomes sound out the other.

A connected message assumes previously negotiated resources and parameters at its source its destination(s) and any intermediate transit points. These resources are referenced by a unique connection identifier and do not need to be contained in each message, only the connection ID is needed to identify the message and refer to its related parameters, thus giving significant savings in message efficiency.

An unconnected message provides a means to communicate without previously negotiated resources at the destination so it shall carry full destination ID details, internal data descriptors and full source ID details if a reply is requested. Unconnected messages are used to create connections.

1 Scope

This part of the standard defines the requirements for the Network and Transport Layer of a node on the deterministic control network.

The Network and Transport Layer of the deterministic control network corresponds to the definition of Layers 3 and 4 of the seven layer OSI model in accordance with ISO 7498. Figure 3-1 shows the location of the Network and Transport Layer within the OSI model.

NOTE: Most of the terms and models for the Network and Transport Layer were adopted from those in ISO / IEC 8802-4 or ISO 8802-3. The data types used are described in part 5, Data Management.

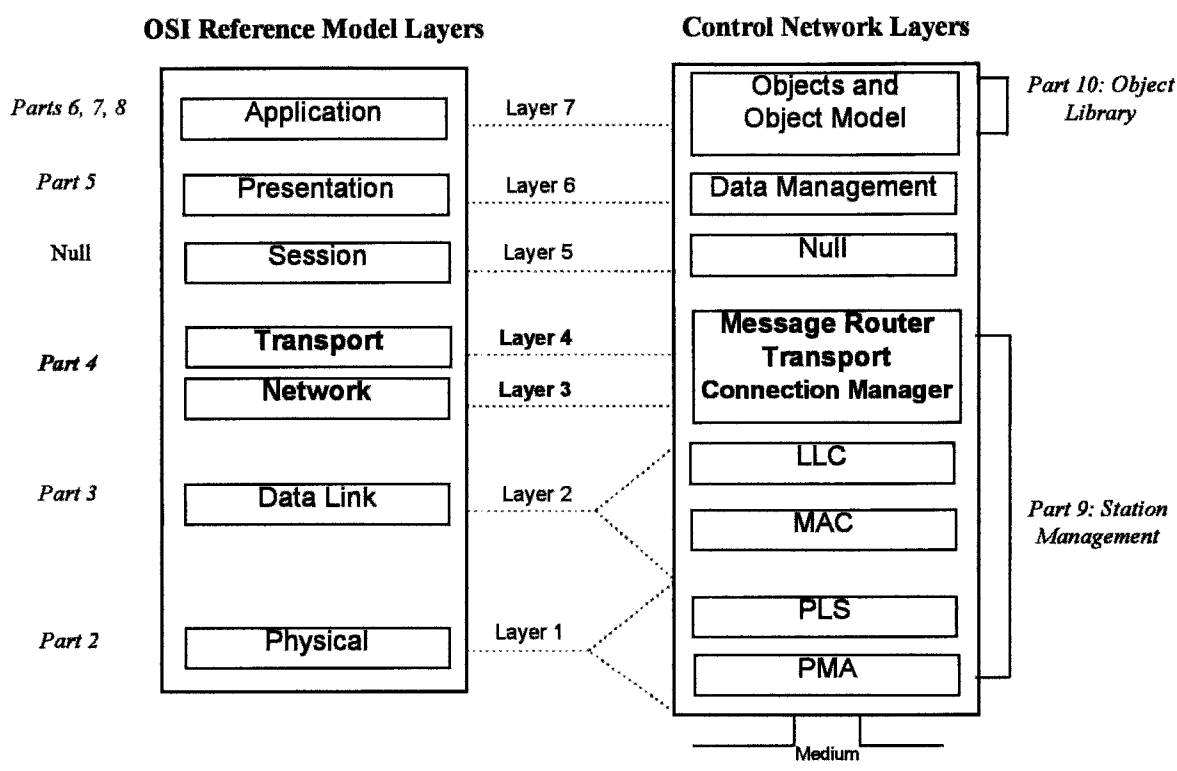


Figure 3-1: Relationship to the ISO/OSI model

2 Unconnected Message Manager (UCMM)

2.1 General

The UCMM shall provides an unconnected request/response message service, limited to a single link, that supports multiple outstanding messages. The required number of outstanding messages shall be implementation specific. The UCMM shall be present in all nodes, and shall support at least one outstanding message.

NOTE: Most I/O modules need only support a single message while controllers and adapters support multiple outstanding messages. Setting up a connection requires communication between devices. These initial communications, require an unconnected message service. The unconnected message service, which only operates on a single link, is different from other transports. That is, packets are queued at each node instead of the normal overwrite. Multiple connections can, therefore, be established at the same time. This reduces the overall time needed to set up all the connections in a typical system.

The UCMM provides a single-link unconnected message service that can support multiple outstanding messages.

This service shall be bound to the message router. All incoming requests shall be forwarded to the message router.

2.2 External Interface

The request, confirmation, indication, and response services used to communicate using the UCMM of the Network and Transport Layer shall be of the form:

```

UCMM_create_request (
    USINT    retries,
    USINT    fixedtag );

UCMM_create_confirm (
    RECORD    record,
    STATUS    status );

UCMM_delete_request (
    RECORD    record );

UCMM_request (
    RECORD    record,
    USINT    destID,
    USINT    service,
    UDINT    timeout,
    USINT    packet[],
    UINT    packet_size );

UCMM_confirm (
    RECORD    record,
    STATUS    status,
    USINT    packet[],
    UINT    packet_size );

UCMM_abort_request (
    RECORD    record );

UCMM_indication (
    RECORD    record,
    USINT    packet[],
    UINT    packet_size );

UCMM_response (
    RECORD    record,
    USINT    packet[],
    UINT    packet_size );

```

UCMM_create_request shall attempt to create a UCMM transaction record. The resulting UCMM_create_confirm shall return the newly created transaction record. The status parameter for these services shall be one of

- SUCCESS;
- CANNOT_CREATE.

UCMM_delete_request shall delete an existing transaction record.

UCMM_request shall send a message to the MAC ID specified by the destID parameter. When the message arrives at the target device, this shall result in a UCMM_indication at the Message Router (MR) in that device. All UCMM_indication(s) shall be received by the MR. The MR, in turn, shall forward the message to the appropriate object. When the object responds, the MR shall forward the response via UCMM_response. When the response arrives at the device which originated the request, a confirmation, UCMM_confirm, shall be returned to layer which sent the request.

The service parameter shall specify the delivery mechanism to use for this message and shall be one of

- Requests_With_Acknowledge = 2;
- Request_With_No_Response = 4;
- Request_With_No_ACK = 7.

NOTE: These values correspond to command codes contained in the UCMM header.

The UCMM_request packet parameter shall be less than 504 bytes since UCMM communication fits into a single Lpacket. The status parameter for the UCMM_confirm service shall be one of

- SUCCESS;
- RECORD_NOT_CREATED
- PACKET_TOO_BIG;
- NO_ACKNOWLEDGE;
- ABORTED;
- TIMEOUT.

The timeout parameter shall specify the duration of the transaction in microseconds. If no response is received before the time-out expires, the transaction is aborted and the status parameter of the confirmation shall be TIMEOUT.

When called by the client, the UCMM_abort_request shall abort the transaction corresponding to the specified transaction record which shall remove the packet from the local DLL if it has not yet been transmit. When called by the server, this request shall abort the transaction which shall suppress the response from being sent. This service need not have a corresponding confirmation since the confirmation of the original request shall serve this purpose by returning a status of ABORTED.

2.3 PDU format

All UCMM PDUs shall be sent and received used fixed tag 0x83. When a device becomes powered, the UCMM shall invoke the DLL_enable_fixed_request service of the Data Link Layer to request that fixed tag 0x83 Lpackets be routed to the UCMM. All sending and receiving of PDUs shall use the DLL_xmit_fixed_request and DLL_rcv_fixed_indication services of the Data Link Layer, respectively. The packet parameter of these services shall be prefixed with the following header before sending to the Data Link Layer:

```
typedef struct
{
    USINT CommandCode;
    USINT TimeOut;
    UINT TransactionID;
}
UCMM_Header;
```

The CommandCode shall specify the type of UCMM PDU as shown in Table 4-1. UCMM packets received with a reserved CommandCode shall be discarded.

Table 4-1: UCMM command codes

CommandCode	Description
0	reserved
1	acknowledge a request
2	request with retry until acknowledged
3	response with retry until acknowledged
4	request with no acknowledge and no response
5	acknowledge a response
6	response which shall not retry (no acknowledge)
7	request with retry until response (no acknowledge)
8 – 255	reserved

The TimeOut field shall specify the duration of the transaction in an 8 bit floating point format. The most significant 5 bits of the field shall be an unsigned exponent that is not biased. The least significant 3 bits shall be the least significant 3 bits of a 4 bit unsigned mantissa. The most significant bit of the mantissa shall be 1 and shall not appear explicitly in the 8 bit representation. The binary point shall be positioned between the implied 1 and the rest of the mantissa. The units of the TimeOut field shall be in milliseconds.

NOTE: Using ANSI C precedence rules, the number of 0.125ms ticks is $(8 | \text{TimeOut} \& 7) \ll ((\text{TimeOut} \gg 3) \& 31)$.

The TransactionID field shall be composed of two sub-fields:

- the least significant 10 bits, RECORD, shall identify a specific transaction;
- the most significant 6 bits, SEQUENCE, shall be used for duplicate detection. It shall be incremented for every unique message on this RECORD; however it shall not be incremented on a retry.

Only one message shall be outstanding for a given RECORD.

If multiple outstanding messages are required then multiple RECORDs are required.

2.4 Client UCMM

2.4.1 Client Transaction State Transition Diagram (STD)

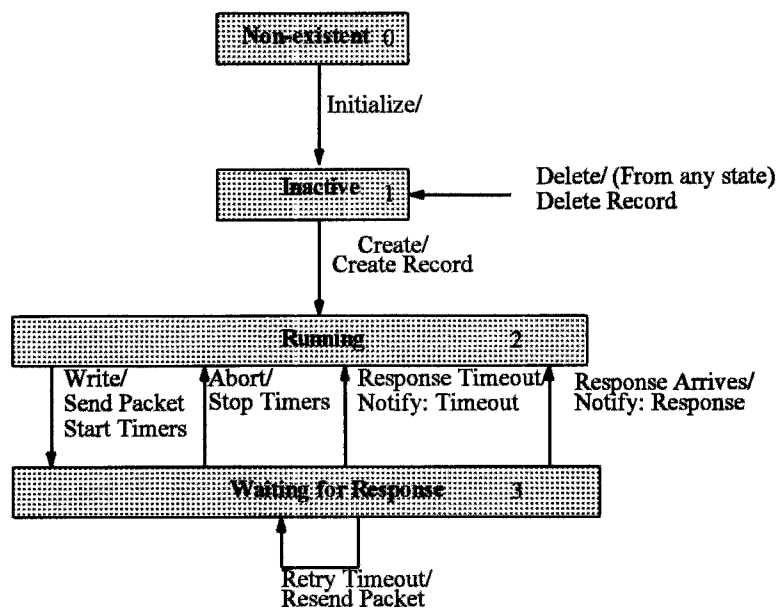


Figure 4-1: Client Transaction State Transition Diagram

2.4.2 Client Transaction State Event Matrix

Table 4-2:UCMM command codes

State Event	Inactive	Running	Waiting for Response
Create	- create record - transition to Running		
Delete	no action	- delete record - increment SEQUENCE - transition to Inactive	- Delete record - increment SEQUENCE - transition to Inactive
Write	confirm status = INVALID_RECORD	- send packet - start Response Timer - initialise Retry Count - start Retry Timer - transition to Waiting	confirm status = BUSY
Abort	confirm status = INVALID_RECORD	no action	- stop timers - increment SEQUENCE - transition to Running
Retry Timeout Request packet available			1) Decrement Retry Count IF Retry Count expired 2) Notify: Timeout-No ACK 3) Free request buffer 4) Increment Sequence # 5) Stop Response Timer 6) Transition to Running ELSE. 1) Resend packet 2) Restart Retry Timer
Response Timeout			- confirm with status = TIMEOUT-No Response - increment SEQUENCE - transition to Running
Response Arrives		no action	- confirm with status = SUCCESS - Send ACK_RESP, unless response indicates no ACK_RESP desired - increment SEQUENCE - transition to Running
ACK_REQ Arrives, sequence number matches stored value		No Action	- Free request buffer - stop retry timer
ACK_REQ Arrives, sequence number not equal to stored value		No Action	No Action

The sequence number shall be set to zero at initialisation. The sequence number value shall be maintained in the inactive state, to be used when the record transitions to running. The retry timeout value shall be fixed for the link

at a value which guarantees that both the client and server nodes have an opportunity to transmit an unscheduled packet.

The response time-out is the response time provided when the record is created.

2.5 Server UCMM

2.5.1 High-end server UCMM

When a packet arrives at the server an instance of a transaction shall be created if resources are available. If resources are not available the packet shall be dropped. When the instance is created, a transaction record shall be created for that instance. This record shall be active for the life of this transaction. The transaction shall end

- when an ACK_RESP is received after a response was sent;
- when the response time timer expires;
- or, when a new packet is received after a response was sent.

2.5.2 High-end Server Transaction State Transition Diagram (STD)

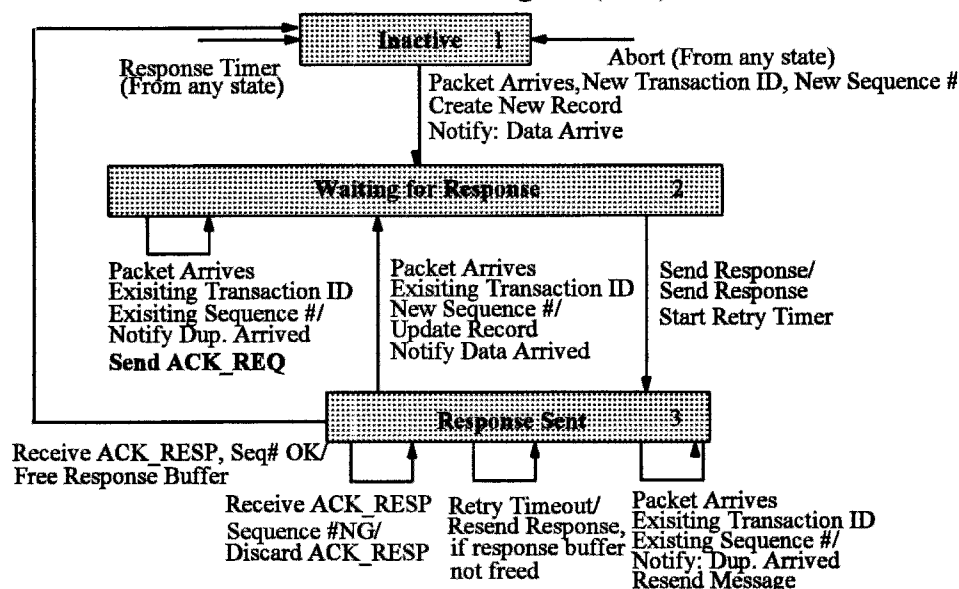


Figure 4-2: High-end Server Transaction State Transition Diagram (STD)

2.5.3 Transaction State Event Matrix

The sequence number shall be set to zero at initialisation. The sequence number value shall be maintained in the Inactive state, to be used when the record transitions to Running. The retry time-out value shall be fixed for the local link.

The response time-out shall be the response time provided when the record is created.

2.5.4 High-end Server Transaction State Event Matrix

Table 4-3: High-end Server Transaction State Event Matrix

State Event	Inactive	Waiting for Response	Response Sent
Packet arrives New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) Send ACK_REQ 4) Start Response Timer 5) Transition to Waiting for App Response	Not Applicable	Not Applicable
Packet arrives Existing Transaction ID and source address New Sequence #	Not Applicable	No Action	1) Update Record 2) Notify: Data Arrived 3) Send ACK_REQ 4) Transition to Waiting for App Response
Packet arrives Existing Transaction ID Existing Sequence #	Not Applicable	1) Notify: Dup. Arrived (Optional) 2) Send ACK_REQ	1) Notify: Dup. Arrived (Optional) 2) Resend Response Message
Response Timer Expires	Not Applicable	1) Notify: Timeout 2) Delete record 3) Transition to Inactive	1) Notify: Timeout 2) Delete record 3) Transition to Inactive
Abort	Error	1) Delete record 2) Transition to Inactive	1) Delete record 2) Transition to Inactive
Send Response	Error	1) Send Response 2) Start Retry Timer 3) Transition to Response Sent	Error
Retry Timeout	Not Applicable	Not Applicable	1) Decrement Retry Count IF Retry Count expired 2) Free Response buffer 3) Stop Response Timer 4) Transition to Inactive ELSE 1) Resend Response message 2) Start Retry Timer
ACK_RESP Attrives, sequence number matches stored value	Error	No Action	1) Delete record 2) Transition to Inactive
ACK_RESP Attrives, sequence number not equal to stored value	Error	No Action	No Action

The response time is the response time received in the message header.

2.5.5 Simple server UCMM

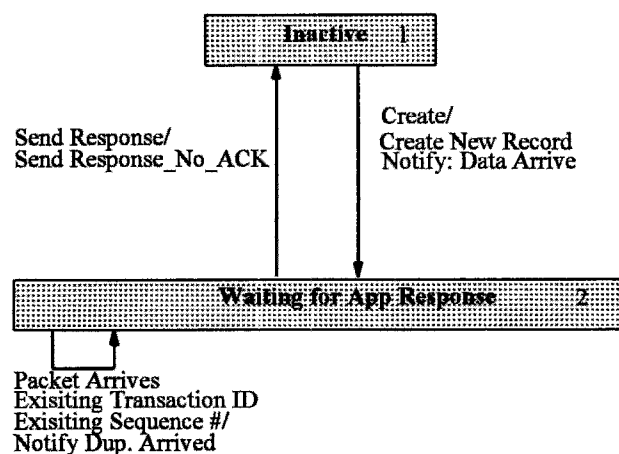


Figure 4-3: Low-end Server Transaction State Transition Diagram

The low-end cannot support the following features:

- perform response message retries
- detect duplicate messages

2.5.6 Low-end Server Transaction State Event Matrix

The state transitions for the low end server UCMM shall be as specified in Table 4-4.

Table 4-4: Low-end Server Transaction State Event Matrix

State Event	Inactive	Waiting for Response
Packet arrives New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) IF Response not immediately available, Send ACK_REQ 4) Transition to Waiting for App Response	Not Applicable
Packet arrives Existing Transaction ID and source address New Sequence #	Not Applicable	No Action
Packet arrives Existing Transaction ID Existing Sequence #	Not Applicable	1) Notify: Dup. Arrived (Optional)
Send Response	Error	1) Send Response_No_ACK 2) Transition to Inactive
ACK_RESP Arrives	Error	No Action

2.6 Example (informative)

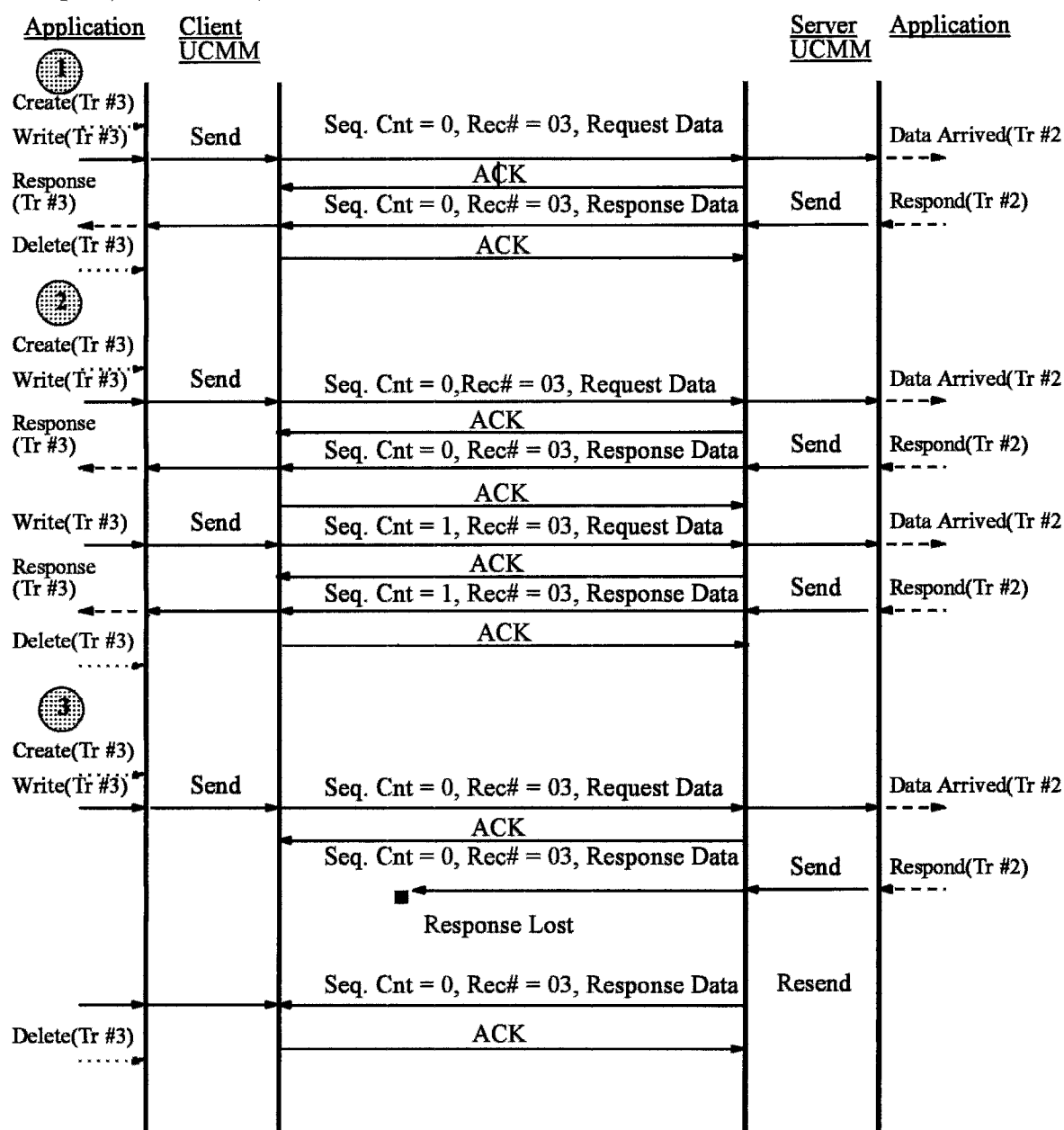


Figure 4-4: Sequence Diagram for a UCMM with One Outstanding Message

Typical transaction: Instance is created, request is written, response is returned, instance is deleted.

Two transactions/same instance: In this case the instance is opened and two requests are sent. Note that the client waits for the first response to return before issuing a second request using the same transaction number.

Lost response: In this case the response is lost and the server issues a retry.

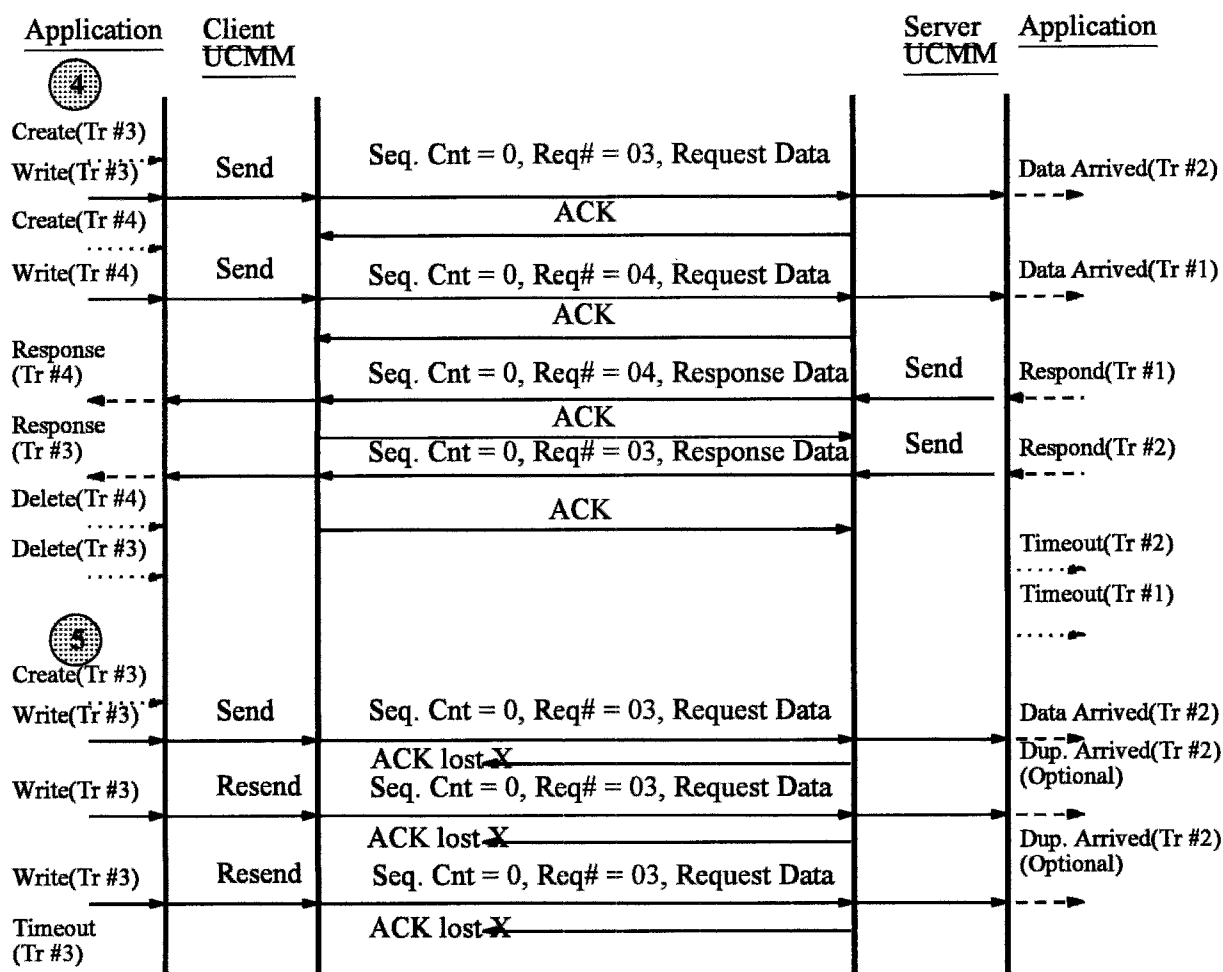


Figure 4-5: Sequence diagram for a UCMM with multiple outstanding messages

Multiple outstanding: In this case the client creates two instances and sends a second request before the first request has completed. Note that the client uses a different transaction number for the second request.

Timeout: In this case the server application does not respond and the client issues a timeout. The timeout causes an error status to be returned. Delete shall be issued to close the transaction.

Transaction	Description
1) Typical transaction	An instance is created. The request is written. The response is returned. The instance is deleted.
2) Two transactions same instance	In this case the instance is opened and two request are sent. Note that the client waits for the first response to return before issuing a second request using the same transaction number.
3) Lost response	In this case the response is lost and the server issues a retry.
4) Multiple outstanding	In this case the client shall create two instances and send a second request before the first request has completed. Note: The client uses a different transaction number for the second request.
5) Time-out	In this case the server application does not respond and the client issues a time-out. Note: A delete is not required to close the transaction.

2.7 Keeper UCMM

Any device that has implemented the Keeper object shall also implement the Keeper UCMM. Except for the fixed tag used, the Keeper UCMM shall be identical to the fixed tag 0x83 UCMM. Transactions for the Keeper UCMM shall be transmit on fixed tag 0x88.

NOTE: Messages to the Keeper object are broadcast since all devices are permitted to implement this object. To facilitate communication to the Keeper object without impacting non-Keeper devices, these messages are transmit on a different fixed tag.

3 Message Router

3.1 External Interface

The indication and response services used alert an object of incoming communication explicitly addressing an object shall be of the form:

TNL_message_indication(		TNL_message_response(	
IDENTIFIER	id,	IDENTIFIER	id,
USINT	service_code;	USINT	status_code,
USINT	param[];	UINT	ext_stat[],
UINT	param_size;	USINT	ext_stat_size,
UINT	path[];	USINT	reply[],
USINT	path_size;);	UINT	reply_size);

When a packet arrives at the message router, either via UCMM or via a direct connection, the first part of the message shall be parsed to determine the object class to send the packet. The first part of a packet sent to the message router, called the path, shall contain addressing information for the packet.

3.2 Path

3.2.1 Contents

The path shall contain multiple segments which can either indicate the route to the next node or what to connect to in the target device. Each segment shall be comprised of a segment descriptor byte which specifies the segment type and segment information whose size is dependent on the segment type. Each segment shall be 16 bit word aligned. If a pad byte is required to achieve the alignment, then the segment shall specify the location of the pad byte.

The possible segment types shall be as follows:

- Path segment (where);
- Logical segment (what);
- Network segment (when);
- Symbolic segment;
- Data segment.

3.2.2 Segment type

3.2.2.1 Interpreting the segment type

Each segment of the connection path shall contain a segment type to indicate how the segment is to be interpreted.. The segment type is contained in the first byte of the segment.

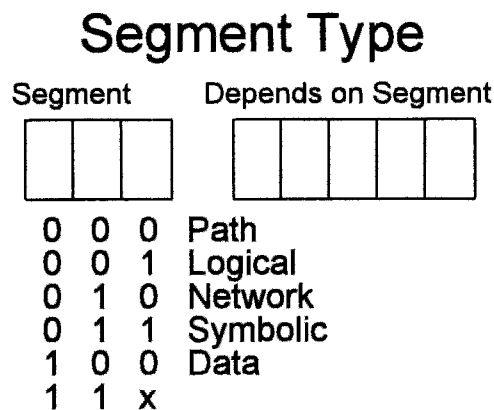


Figure 4-6: Segment type

3.2.2.2 Path segment

The path segment shall indicate the port and link address for the next hop in the path. The first byte shall contain the segment type and port identifier. The port identifier shall specify a port type and port number or an escape to an extended port identifier when the module can support more than 7 ports. The port type and number shall be followed by a link address whose format depends on the port type. For the common port types, the link address shall be a single byte. Other port types, such as TCP/IP Ethernet shall require a larger 32 bit link address. If the size of the link address is greater than a single byte, the Node/Slot Address Size bit shall be set and then the next byte shall give the length of the address.

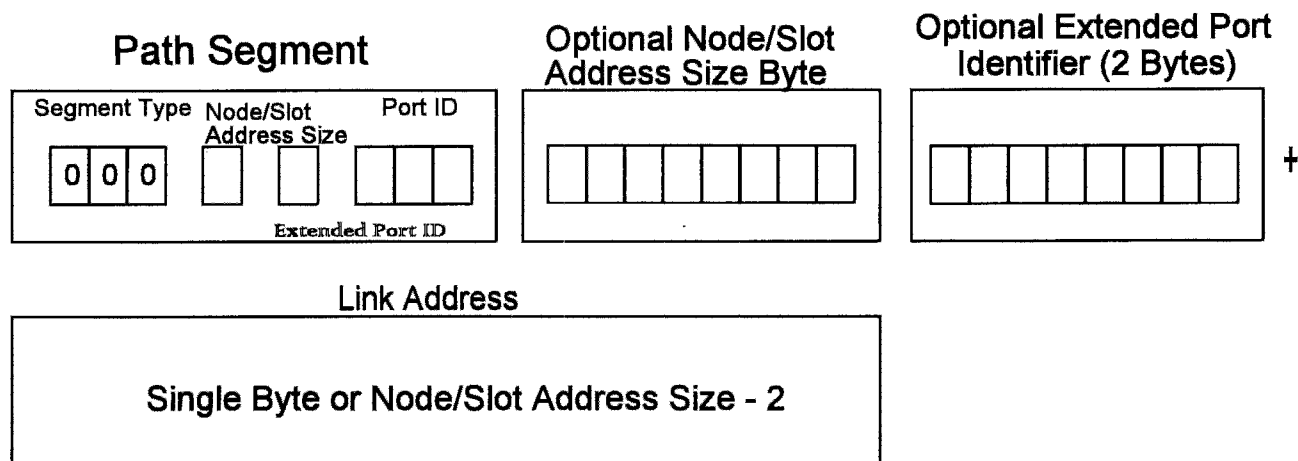


Figure 4-7: Path segment

If the Node/Slot Address Size bit is 0, then the Address shall be specified by a single byte which follows the Port ID. If the Node/Slot Address Size bit is 1, then the size of the address shall follow the Path Segment Byte and shall be contained in a single byte. This bit shall be set if the Node/Slot Address Size is greater than 1 byte.

If the Extended Port ID bit is 0, then the Port ID shall be specified by the next three bits. If the Extended Port ID bit is 1, then the Port ID shall be specified by the next two bytes. If the optional Node/Slot Address Size Byte is used, it shall also include the size of the Extended Port ID if it is also included in the segment. The Extended Port ID shall be represented in Little Endian format.

NOTE: The following example show a Path Segment without the Extended Port Identifier:

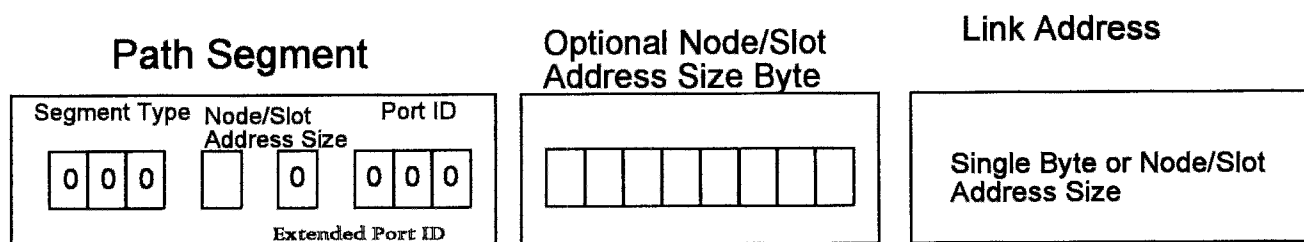


Figure 4-8: Non-extended path segment

Table 4-5: Port type

Port Number			Port Type
0	0	0	Reserved. Use when specifying Extended Port Identifier.
0	0	1	Reserved
0	1	0	Port 2 : typically the first network port
0	1	1	Port 3
1	0	0	Port 4
1	0	1	Port 5
1	1	0	Port 6
1	1	1	Port 7

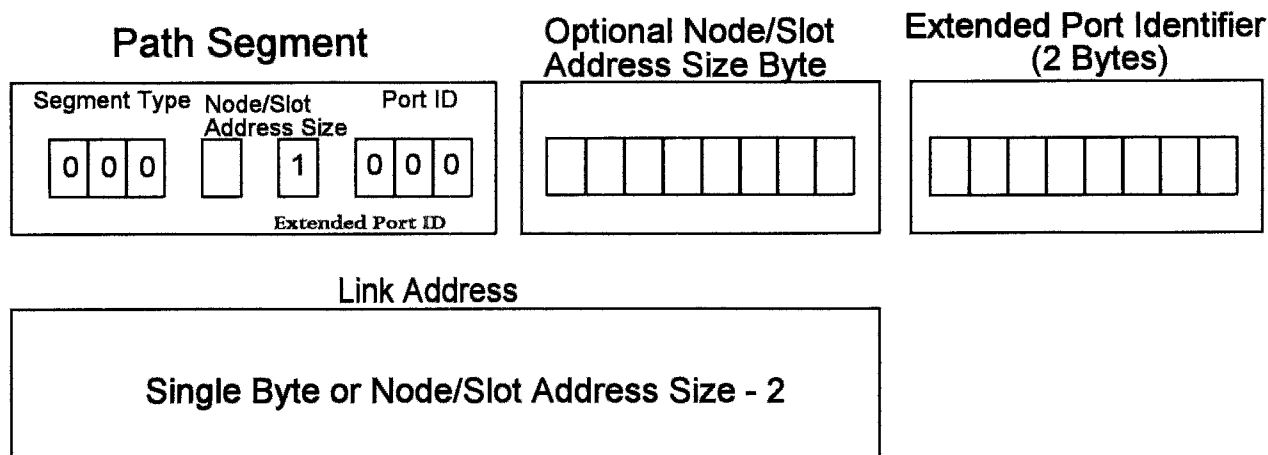


Figure 4-9: Extended path segment

The Extended Port Identifier format of the Path Segment shall only be used when there are more than 7 ports possible on the device. The Path Segment shall always be packed into the smallest Path Segment format possible. Examples of possible path segments are as shown in Table 4-6.

Table 4-6: Possible path segment examples

Path Segment	Notes
[02][06]	Port 2, MAC ID 6
[15][0E][31][33][31][2E] [31][35][31][2E][31][33] [32][2E][35][35]	Multi-Byte Address for Ethernet Port 5, Link Address 131.151.132.55 (IP Address).
[18][08][12][00][01][23] [45][67][89][AB]	Port 18, Link Address 0123456789AB. The Address is defined as a character array.
[08][08][00][01]	Port 8, MAC ID 1

3.2.2.3 Logical segment

The logical segment shall select a particular class, instance, or structure element within a device. The first byte shall indicate the type and format of the logical segment.

Logical Segment Type

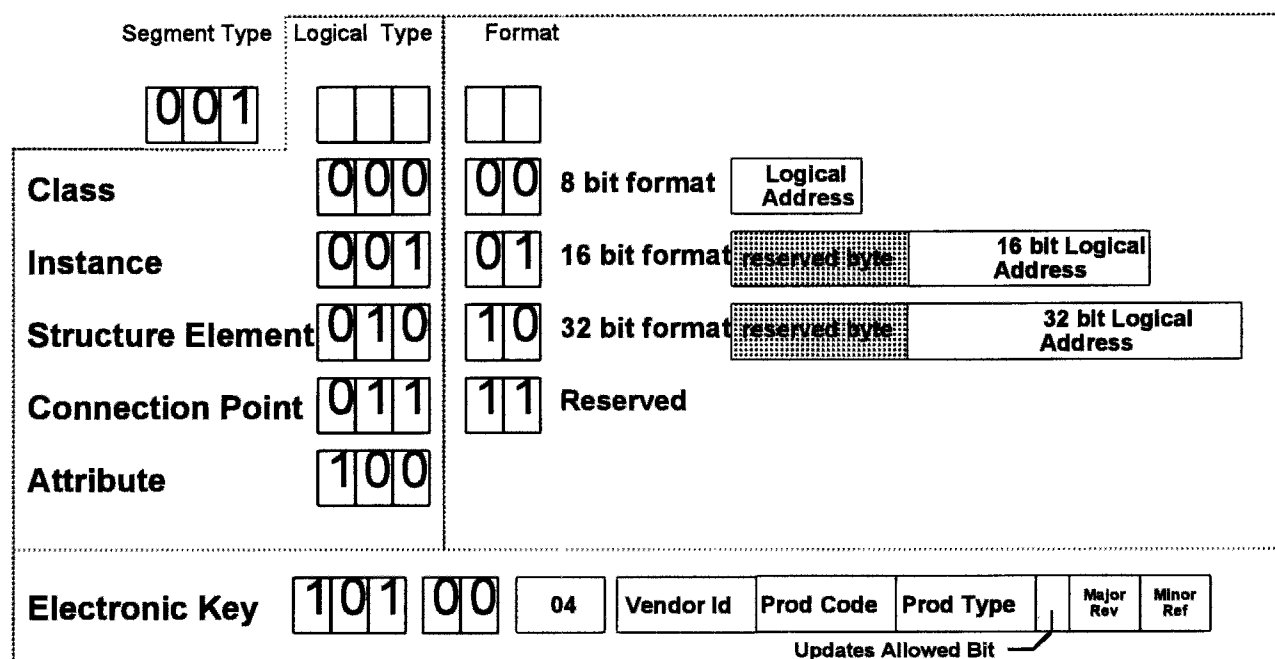


Figure 4-10: Logical segment

The class and instance shall be numeric values which the device can map to internal objects or data structures

Table 4-7: Example logical segments

Segment	Logical	Format	Notes
[21][rb][0500]	Class	16-bit	Register class
[21][rb][0100]	Class	16-bit	Identity object class
[25][rb][2000]	Instance	16-bit	Instance 20
[20][01]	Class	8-bit	Identity object class
[30][01]	Attribute	8-bit	Attribute 1 of an object
[2C][05]	Conn. Pt.	8-bit	Connection point 5

The logical segment shall be used to indicate that the target device has been reached. Additional segments beyond the logical segment shall be used by the target device to select the desired transport for a data structure or function.

Table 4-8: Segments

Segment	Description
Class segment	The class segment specifies a type of object within a device. For example the class could specify the connection manager, message router, assembly or identity object.
Instance segment	The instance segment specifies a particular instance of an object. For example instance 4 of the assembly object. If an instance number of 0 is used, the class itself is being addressed. This is also called the class instance.
Structure element	The structure element addresses a submember of either an instance or attribute. For example a timer instance contains multiple structure elements, accumulator, preset and done bit.
Connection point	The connection point or secondary instance is used to address multiple instances in a single Open Connection request. For example a single Open can configure a device and connect to both the input and output connection points using a class and instance for the configuration and a connection point for the input and output.
Attribute	The logical segment attribute specifies a single attribute of an instance. These attributes can be a simple data type such as an integer or the attribute may be a complex structure. In the case of a more complex structure the logical segment structure element can be used to further refine the address to a single structure element of the complex attribute.
Electronic key segment	The electronic key segment is used to verify the type of device. The key segment contains the vendor ID, Product Type, Catalog code and revision information. The key is normally sent as the first logical segment in a path and is checked by the message router for the correct device type before any additional address checks are made. The format of the electronic key is as follows:

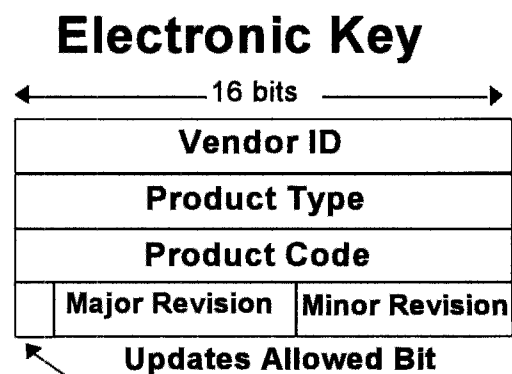


Figure 4-11: Electronic key segment

Table 4-9: Electronic key segment

Key segment	Description
Vendor ID	An UINT which specifies the device vendor desired or the value 0 if no specific vendor is required. A vendor ID is assigned to each registered vendor. For example the Vendor ID for Allen-Bradley is 0001.
Product type	An UINT that specifies a class of product such as digital input or analog inputs. The Product type like the vendor ID can be ignored by specifying a value of 0.
Product code	An UINT that is vendor specific with each vendor having a 16 bit product codes which translate to individual models. The product code, like the vendor ID can be ignored by specifying a value of 0.
The major revision	An USINT that specifies either a 0 where no revision match is desired or a particular major revision of the product. The translating of major revisions to product revisions is vendor specific. For example, if the control logic division used major revision to refer to the series of the device, Series A would be represented by 1 in the major revision field.
The minor revision	An USINT that specifies either a 0 where no revision match is desired or a particular minor revision of the product. The translating of minor revisions to product revisions is vendor specific. For example, if the control logic division used minor revision to refer to the revision of the device, Revision A would be represented by 1 in the minor revision field.
Updates allowed	A bit that modifies the revision matching function. If the Updates Allowed bit is set to a 0, the revision, both major and minor, shall match exactly. If the Updates Allowed bit is set to a 1 any module revision equal to or higher than the revision in the key shall match.

3.2.2.4 Network segment

3.2.2.4.1 Format

The Network Segment shall be used to specify network parameters which may be required by a node to transmit a message across a network. The Network Segment shall immediately precede the path segment of the bridge to which it applies. In other words, the Network Segment shall be the first item in the path which the bridge receives.

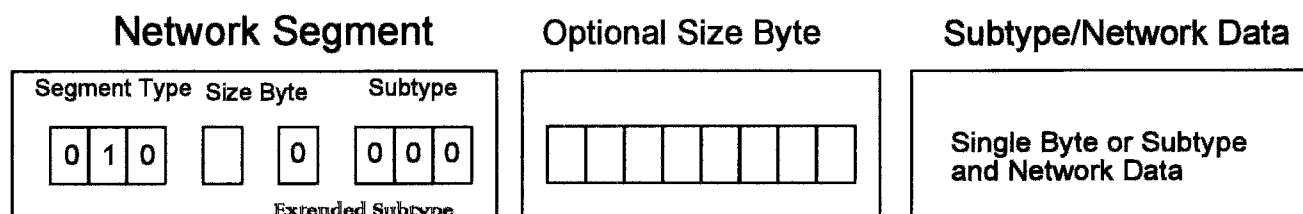


Figure 4-12: Network segment

Table 4-10: Network segment

Subtype			Network Data
0	0	0	Reserved
0	0	1	Schedule Subtype. 1 byte of encoded data is required. The Encoded value specifies the Starting NUT (Network Update Time) and Update Interval. An encoded value of zero is invalid. The value is as follows: Encoded Value = Starting NUT + Update Interval
0	1	0	Fixed Tag Subtype. 1 byte of encoded data is required. The value specifies the Fixed Tag to use for subsequent messages or replies.
0	1	1	Reserved
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

3.2.2.4.2 Schedule segment

The Schedule Network Segment Subtype shall specify both the NUT (Network Update Time) multiplier and the update interval at which to send scheduled data on the link. This segment shall be included in the path to a device so that each intermediary node can schedule bandwidth for subsequent scheduled traffic.

For example, if the connection is meant to be sent every 64th update (NUT) starting on update (NUT) 17, the encoded value shall be:

$$17 + 64 = 81$$

This mapping is reversable.

An encoded value of 0x00 shall be invalid. If a node receives a request for a scheduled connection in which either no Network Segment exists, or the Network Segment exists but the Encoded Value is 0, it shall return a general status of Connection Failure and an extended status of Scheduling Not Specified.

3.2.2.4.3 Fixed tag segment

The fixed tag network segment shall specify the fixed tag which is to be used when sending an unconnected message. This segment subtype shall precede the path segment within the path. If the fixed tag segment is not present, then the default fixed tag shall be used.

3.2.2.5 Data segment

The data segment may contain parameters for a target application. The size of the data segment shall specify the number of words and shall depends on the application. The data segment can be any number of 16-bit words up to 255. A path can contain more than one data segment.

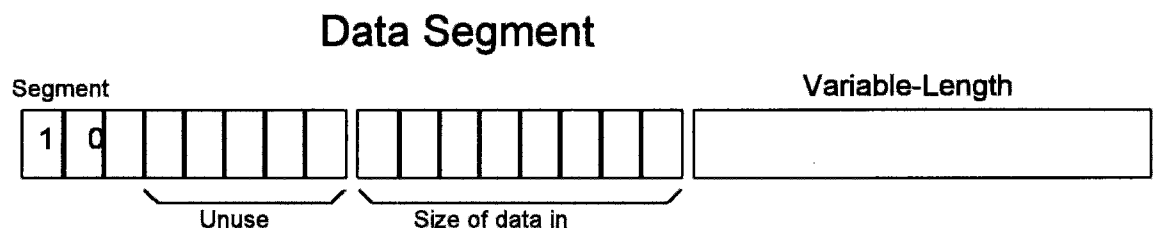


Figure 4-13: Data segment

The size of the data segment shall be specified in words, as shown below:

Table 4-11: Data segment size

Connection path size	Sample connection path	Notes
8	[80][07][0100][0200][0300][0400][0500][0600][0700]	7 words of data in a data segment
9	[21][pad][0500][25][pad][2000][80][04][0100][0200][0300][0400]	Class 5, Instance 20 with 4 words of data

The data can be configuration information for an object, additional parameters necessary for the connection, or any other information. The data segment shall not be interpreted by any other device in the path, so only the originating and target applications need agree on its contents.

4 Connection Manager Service Descriptions

4.1 External interface (SDU)

4.1.1 TNL_open_request

The request service used to originate an application connection via the Network and Transport Layer shall be of the form:

```

TNL_open_request (
    IDENTIFIER      id,
    PRIORITY        est_priority,
    UDINT           est_timeout,
    TNL_ConParm     OtoT,
    TNL_ConParm     TtoO,
    USINT           RPI_multiplier,
    TRIGGER         trigger,
    USINT           trans_class,
    BOOL            is_server,
    USNT            path[],
    USINT           path_size );

```

The TNL_open_request service, sent from the originating application to the local connection manager, shall start the open connection process. The TNL_open_request contains the parameters to select connection types, transports and to specify the requested packet interval in both the originator to target and target to originator direction.

The IDENTIFIER, "id", shall correlate a confirmation with a corresponding request. The est_priority parameter shall determine which DLL priority to use for the messages that establish the connection, and shall be one on LOW or HIGH. The est_timeout parameter shall determine the time to wait for connection establishment. The units for est_timeout shall be milliseconds, and shall be in the range 1 through 8355840

NOTES: The definition of DLL priority is specified in part 3.

The range of "est_timeout" is determined by the format of the message used to establish the connections

The `OtoT` parameter shall specify the characteristics of the originator to target communication on the new connection. The `TtoO` parameter shall specify the characteristics of the target to originator communication. Both parameters shall be of the form:

```
typedef struct
{
    UDINT      RPI;          // requested packet interval, in microseconds
    CONNTYPE   type;         // one of NULL, MULTICAST, or POINT-TO-POINT
    PRIORITY   priority;     // one of LOW, HIGH, or SCHEDULED
    BOOLEAN    variable;     // packet is fixed size(0) or variable size (1)
    UINT       size;         // maximum size packets for connection
}
TNL_ConnParam;
```

The requested packet interval (RPI) shall specify the expected time between packets, and shall be expressed as an integer number of microseconds. The type parameter shall indicate the type of the connection, and shall be one of NULL, MULTICAST, or POINT-TO-POINT. A value of NULL indicates that the specified direction (O⇒T or T⇒O) shall have no transport associated. A value of MULTICAST indicates that other connections may later participate in the transports. Conversely, a value of POINT-TO-POINT indicates that this connection shall not allow other transport to participate.

The priority parameter shall determine which DLL priority to use for the new transports, and shall be one of LOW, HIGH, or SCHEDULED. The variable parameter specifies whether every application data which traverses the new transport shall be the same size, and shall be either variable (1) or fixed (0). The size parameter shall specify the size of the application data in bytes.

The product of the `RPI_multiplier` parameter and the actual packet interval (API) shall specify the time-out for the transport. If the application has not used to transport within this time-out, the transports shall close the transport which shall subsequently close the connection.

The `trans_class` parameter shall specify the transport class, and shall be in the range 0 through 6. The `trigger` parameter shall specify the trigger mode, and shall be one of CYCLIC, CHANGE_OF_STATE, or APPLICATION. The `is_server` parameter shall specify if the new transport shall be of server (1) or a client (0). These parameter shall be used to configure the transport(s).

NOTE: The meaning of these parameters to the transports are described in clause 6.

The `path` and `path_size` parameters together shall specify the target of the new connection. The `path` shall contain the node, class, and instance of target application. The `path_size` parameter shall specify the number of UINTs in the `path` array.

4.1.2 TNL_open_confirm

The confirmation service used to respond to the connection request shall be of the form:

```
TNL_open_confirm(  
    IDENTIFIER      id,  
    TRANSPORT       transport,  
    UDINT           OtoT_API,  
    UDINT           TtoO_API,  
    UINT            reply[],  
    USINT           reply_size,  
    USINT           status,  
    USINT           remaining_path,  
    UINT            ext_stat[],  
    USINT           ext_stat_size );
```

After the connection has been established, or if the connection attempt failed, a TNL_open_confirm shall be sent from the local connection manager back to the originating application. The confirmation shall indicate the status of the connection and shall provide information about the connection including the transport parameter which shall serve as a reference to the newly established connection.

The IDENTIFIER, "id", shall correlate a confirmation with a corresponding request. The OtoT_API and TtoO_API shall specify the actual packet interval that shall be used for the new connection, and shall be expressed in microseconds. These value may be different than the requested packet intervals, but shall always be equal or smaller than the requested interval.

If the target has any application specify reply data, it shall be returned in the reply and reply_size parameters. The reply_size parameter shall specify the number of UINTs returned in the reply array.

The rest of the parameters, status, remaining_path, ext_stat and ext_stat_size, shall the type and location of any errors found during connection establishment. The ext_stat parameter is an array of UINTs in which extended status codes shall be placed. The ext_stat_size shall specify the number of UINTs in the ext_stat array, and shall be 0 if no extended status was returned.

In the failure response, the remaining_path parameter specifies the "pre-stripped" size. This is the size of the path when the node first receives the request and has not yet started processing it. A target node may instead return 0 for this parameter.

4.1.3 TNL_open_indication

The indication service which attempts to open a connection to an object shall be of the form:

```
TNL_open_indication(
    IDENTIFIER    id,
    USINT         RPI_multiplier,
    TRIGGER       trigger,
    TN_ConnParm   OtoT,
    TN_ConnParm   OtoT,
    TRIGGER       trigger,
    USINT         trans_class,
    BOOL          is_server,
    UINT          vendor_id;
    UDINT         vendor_ser_num;
    USNT          tgt_path[],
    USINT         tgt_path_size);
```

The TNL_open_indication service shall be sent from the target connection manager to the target application. The parameters of this service shall include any remaining path which could be a data segment used to configure the application, the transport class, and the trigger. The application shall reply to the open_indication service with a TNL_open_response whether the application accepts the connection or not.

4.1.4 TNL_open_response

The TNL_open_response shall be returned by the target application.

```
TNL_open_response (
    IDENTIFIER    id,
    USINT         status,
    USINT         remaining_path,
    UINT          ext_stat[],
    USINT         ext_stat_size,
    UDINT         OtoT_API,
    UDINT         TtoO_API,
    UINT          reply[],
    USINT         reply_size );
```

4.1.5 TNL_close_request

The request service used to close an existing connection via the Network and Transport Layer shall be of the form:

```
TNL_close_request (
    IDENTIFIER      id,
    PRIORITY        est_priority,
    UDINT           est_timeout,
    TNL_ConParm     OtoT,
    TNL_ConParm     TtoO,
    USINT           RPI_multiplier,
    TRIGGER         trigger,
    USINT           trans_class,
    BOOL            is_server,
    USNT            path[],
    USINT           path_size );
```

The close_request service shall be sent by the originator application of a connection to close the connection and de-allocate the resources associated with the connection. The service shall be sent from the originator application to the local connection manager. The service shall indicate the connection to close by using the original path and the connection serial number. The originating application shall save both the connection path required to open the connection and the connection serial number returned by the open_confirm.

4.1.6 TNL_close_confirm

The confirmation service used to indicate a connection is closed shall be of the form:

```
TNL_close_confirm(
    IDENTIFIER      id,
    TRANSPORT       transport,
    UINT            reply[],
    USINT           reply_size,
    USINT           status,
    USINT           remaining_path,
    UINT            ext_stat[],
    USINT           ext_stat_size );
```

The TNL_close_confirm shall be sent from the local connection manager to the originating application to indicate the status of the close connection request.

A successful TNL_close_confirm shall be returned when the close has been acknowledged by all the nodes involved in the original connection. The reply shall indicate the connection serial number of the closed connection. Once the reply indicating success has been received, the connection shall be closed. The target application may optionally pass back application specific information in the close reply. This information shall be object specific. If no application reply information is contained in the service, the application reply size shall be zero. The object specific status words shall contain information about the reason for the failure. The remaining connection path size shall contain the path size from the point at which the close connection failed.

In the failure response, the remaining Connection Path Size shall be the "pre-stripped" size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the "pre-stripped" size or 0 for the remaining Connection Path Size.

4.1.7 TNL_close_indication

The indication service which attempts to close a connection to an object shall be of the form:

```
TNL_close_indication(
    IDENTIFIER    id,
    UINT          vendor_id;
    UDINT         vendor_ser_num;
    USNT          tgt_path[],
    USINT         tgt_path_size);
```

The close_indication service specifies the connection to be closed using the connection serial number, vendor ID, and originator serial number which was associated with the open_request which opened the connection.

4.1.8 TNL_close_response

The TNL_close_response shall be returned to originating application.

```
TNL_close_response(
    IDENTIFIER    id,
    USINT         status,
    USINT         remaining_path,
    UINT          ext_stat[],
    USINT         ext_stat_size,
    UDINT         OtoT_API,
    UDINT         TtoO_API,
    UINT          reply[],
    USINT         reply_size );
```

The TNL_close_response shall be sent from the target application to the target connection manager to indicate the success of the TNL_close_request.. A successful TNL_close_response service shall be returned when the close has been acknowledged by the target application. The reply shall include the connection serial number, vendor ID, and originator serial number of the closed connection. Once the reply indicating success has been received, the connection shall be closed. The target application may optionally pass back application specific information in the close reply. This application reply information shall be object specific. If no application reply information is contained in the service, the application reply size shall be zero.

A target application shall not refuse to close a connection but may not be able to find the specified connection. Once the TNL_close_response is received, the specified connection shall be closed or shall not be present. The object specific status words shall contain information about the reason for the failure. The remaining connection path size shall contains the path size from the point at which the close connection failed.

4.2 PDU

4.2.1 Transmission of PDUs

The connection manager objects at different nodes shall communicate using the UCMM service described in clause 2.

Table 4-12: Common and class specific services

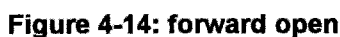
Name	Service code	Need in implementation		Description of service
		Class	Instance	
fwd_close	0x4E	n/a	Required	Close a connection
unconnected_send	0x52	n/a	Optional	Unconnected Send Service. Only Originating and Bridge Nodes need to implement
ni_fwd_open	0x54	n/a	Required	Open a connection

4.2.2 fwd_open

NOTE: The fwd_open Service Request set up all required network, transport, and application connections. An application connection consists of a single transport connection, and one or two network connections which are in turn comprised of multiple link connections. Each path segment in the connection path uses a link connection. The fwd_open service between two devices builds one or two link connections as specified by the network connection parameter and the requested packet intervals (RPI). Since up to two network connections can be required for a single transport connection, they are differentiated by the O⇒T and T⇒O designations; O⇒T means originator to target, and T⇒O means target to originator.

A duplicate fwd_open service shall be defined as a fwd_open service whose Connection Serial Number, Vendor Id, and Originator Serial Number match an existing connection's parameters. If the duplicate fwd_open service is a null fwd_open service (defined as the Connection Type in both the O⇒T and T⇒O Network Connection Parameter fields are NULL), then the fwd_open service shall be forwarded to the application for further processing. null fwd_open Service Requests may be used to reconfigure the connection. The Connection Manager in the intermediate nodes need not allocate additional resources for a duplicate fwd_open service since the resources have already been allocated. If the duplicate fwd_open service is not null, then an error (Connection in Use - see section) shall be returned.

The fields described shall not be used for all combinations of network and transport connections, but shall appear in the fwd_open service PDU.



The format of the fwd_open service is shown in Table 4-13. A duplicate open_indication service is defined as a open_indication service whose Connection Serial Number, Vendor Id, and Originator Serial Number match an existing connection's parameters. This is not necessarily an error condition. If the duplicate open_indication service is a null open_indication service (defined as the Connection Type in both the O⇒T and T⇒O Network Connection Parameter fields are null), then the open_indication service shall be processed by the target application. null open_indication Service Requests may be used to reconfigure the connection. The target application does not need to allocate additional resources for a duplicate open_indication service since the resources have already been allocated. If the duplicate open_indication service is not null, then an error (Connection in Use - see section) shall be returned.

Table 4-13: forward open PDU

Function	Bytes in the Service	Notes
Service	[54]	fwd_open
Size of path	[ss] = [02]	Size of path in words
Class segment for connection manager	[SS][cc]= [20][06]	SS = logical class segment 8-bit format cc = class of connection manager
Class Segment for Connection Manager Instance	[SS][ii]= [24][01]	SS = logical instance segment 8-bit format ii = instance number (1)
Conn. priority/tick time	[pssss]	p = priority of the UCMM msg to set up the connection ssss = Time/Tick 0 = 1 ms, 1 = 2 ms, 2 = 4 ms, etc.
Conn. time-out ticks	[tt]	tt = connection time-out ticks time-out = ticks * time/tick
Net O⇒T connection ID	[tgmmcccc]	t = ID type; g = group; mm = MAC or slot; cccc = connection number
Net T⇒O connection ID	[tgmmcccc]	t = ID type; g = group; mm = MAC or slot; cccc = connection number
Connection serial no.	[cccc]	cccc = connection serial number this was assigned by the originator
Vendor ID	[vvvv]	vvvv = Vendor ID
Originator serial no.	[ssssssss]	ssssssss = originator serial number
Connection Time-out multiplier	[xm]	m = multiplier 0 = 4, 1 = 8, 2 = 16 ...
Reserved	[00][0000]	Reserved byte and word shall be sent as zero
RPI O⇒T	[iiiiiii]	RPI in us. between packets
Net O⇒T Connection Parameters	[tPfS]	t = network connection type f = fixed / variable P = priority S = size
RPI T⇒O	[iiiiiii]	RPI in us. between packets
Net T-O Connection Parameters	[tPfS]	t = network connection type f = fixed / variable; P = priority S = size
Transport Class / Trigger	[tC]	tC is the transport type and trigger mode
Connection path size	[ss]	Size of connection path
Connection path	[seg]...[seg]	seg = address segment

4.2.3 fwd_open_reply

4.2.3.1 Usage

The response to the fwd_open PDU is a fwd_open_reply PDU. The fwd_open_reply shall contains the original connection serial number and owner vendor and serial number and the connection IDs (CID) necessary to complete the connection if successful and error information if the connection failed.

4.2.3.2 Format of fwd_open_reply if success

This format shall be used when the connection is successful. Success shall be returned when the connection requested has been established from this point forward in the path. This reply also shall indicate the connection serial number and the actual packet rate of the connection. Once the Success reply has been received, the connection shall be open from this point forward in the path.

Table 4-14: Connection success format of fwd_open_reply

Function	Bytes in the service	Notes
Reply Service	[D4][00]	Reply service with path size 0
Status	[ss][oo]	ss = General status code; oo = size of object-specific status/event data in words both ss and oo are zero for success
Net O-T connection ID	[tgmmeccc]	t = ID type; g = group; mm = MAC or slot; cccc = connection number
Net T-O connection ID	[tgmmeccc]	t = ID type; g = group; mm = MAC or slot; cccc = connection number
Connection serial number	[cccc]	cccc = connection serial number
Vendor ID	[vvvv]	vvvv = Vendor ID
Originator serial number	[ssssssss]	ssssssss = originator serial number
API O-T	[iiiiiii]	APR in us. per packet this may be different than the O-T RPI
API T-O	[iiiiiii]	APR in us. per packet this may be different than the O-T RPI
Application Reply size	[xx][rb]	Size in words of the reply from the application as a result of the data segment rb = reserved byte shall be set to zero
Application Reply	[xxxx]....[xxxx]	Array of words for the application reply

4.2.3.3 Format of fwd_open_reply if failure

This format shall be used for all failures of the connection system. The requested connection shall not be established, and the object specific status words contain information about the reason for the failure. The remaining connection path size shall contain the length from the point at which the connection failed. This information can be used to debug the problem. The format of the fwd_open_reply service for a connection failure shall be as shown in Table 4-15.

Table 4-15: Connection failure format of fwd_open_reply

Function	Bytes in the service	Notes
Reply Service	[D4][00]	Reply service with path size 0
Status	[ss][oo]	ss = General status code; oo = size of object-specific status/event data in words
Object-specific status Reason for failure	[rrrr]	rrrr = Reason for resource failure This field is present only if oo is non-zero
Connection serial number	[cccc]	cccc = connection serial number
Vendor ID	[vvvv]	vvvv = Vendor ID
Originator serial number	[ssssssss]	ssssssss = originator serial number
Remaining Connection Path Size	[ss][rb]	Size of remaining connection path in 16 bit words rb = reserved bytes shall be set to zero

In the failure response, the remaining Connection Path Size shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the “pre-stripped” size or 0 for the remaining Connection Path Size.

4.2.4 fwd_close

4.2.4.1 Usage

The fwd_close PDU shall remove a connection from all the nodes participating in the original connection. The fwd_close shall be sent between connection managers as specified in the connection path. The fwd_close request shall cause all resources in all nodes participating in the connection to be deallocated, including connection IDs, bandwidth, and internal memory buffers.

If an intermediate node cannot find the connection which is to be closed (it may have timed out at the node), the fwd_close PDU shall still be forwarded to downstream nodes or the target application (via the app_close service). The downstream nodes or the target application may still have a record of the connection and may need the fwd_close service to release any resources which were allocated for the connection.

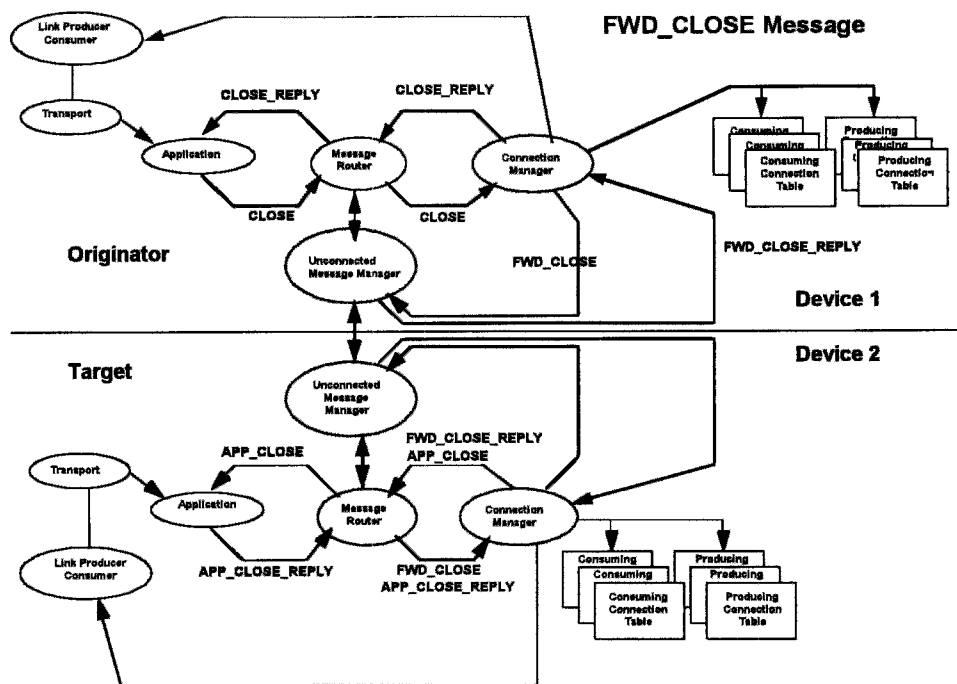


Figure 4-15: forward close PDU

4.2.4.2 Format of fwd_close

The format of the fwd_close service shall be as shown in Table 4-16.

Table 4-16: Format of fwd_close

Function	Bytes in the service	Notes
Service	[4E]	Close Connection
Size of path	[ss] := [02]	size of path in words
Class segment for connection manager	[SS][cc] = [20][06]	SS = logical class segment 8-bit format cc = class of connection manager
Class segment for connection manager instance	[SS][ii] = [24][01]	SS = logical instance segment 8-bit format ii = instance number
Conn. Priority/tick time	[pssss]	p = priority of the UCMM msg to set up the connection ssss = Time/Tick 0 = 1 ms, 1 = 2 ms, 2 = 4 ms, etc.
Conn. time-out ticks	[t]	t = connection time-out ticks time-out = ticks * time/tick
Connection serial no.	[cccc]	cccc = connection serial number
Vendor ID	[vvvv]	vvvv = vendor ID
Originator Serial Number	[ssssssss]	ssssssss = originator serial number
Connection path size	[ss][rb]	Size of connection path rb = reserved byte shall be set to zero
Connection path	[seg]...[seg]	seg = address segment

4.2.5.1 Usage

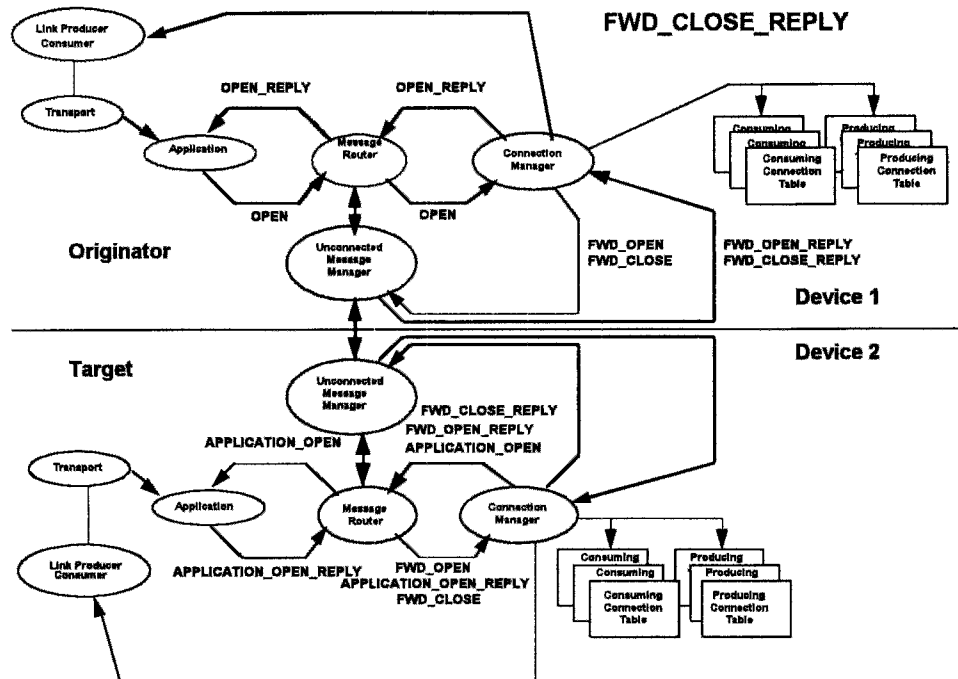


Figure 4-16: forward close reply PDU

4.2.5.2 Format of fwd_close_reply if success

Table 4-17: Connection success format of fwd_close_reply

Function	Bytes in the service	Notes
Reply Service	[CE][00]	fwd_close_reply service with path size 0
Status	[ss][oo]	ss = General status code; oo = size of object-specific status/event data in words both ss and oo are zero for success
Connection serial number	[cccc]	cccc = connection serial number
Vendor ID	[vvvv]	vvvv = Vendor ID
Originator serial number	[ssssssss]	ssssssss = originator serial number
Application Reply size	[xx][rb]	Size in words of the reply from the application as a result of the data segment rb = reserved bytes shall be set to zero
Application Reply	[xxxx]...[xxxx]	Array of words for the application reply

4.2.5.3 Format of fwd_close_reply if failure

This format, shown in Table 4-18, shall be used if the close connection request fails. The object specific status words shall contain information about the reason for the failure. The remaining connection path size shall contain the path size from the point at which the close connection failed..

Table 4-18: Connection failure format of fwd_close_reply

Function	Bytes in the service	Notes
Reply Service	[CE][00]	Reply service with path size 0
Status	[ss][oo]	ss = General status code; oo = size of object-specific status/event data in words
Object-specific status Reason for failure	[rrrr]	rrrr = Reason for resource failure This field is present only if oo is non-zero
Connection serial number	[cccc]	cccc = connection serial number
Vendor ID	[vvvv]	vvvv = Vendor ID
Originator serial number	[ssssssss]	ssssssss = originator serial number
Remaining Connection path size	[ss][rb]	Size of remaining connection path in 16 bit words rb = reserved bytes shall be set to zero

In the failure response, the remaining Connection Path Size shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the “pre-stripped” size or 0 for the remaining Connection Path Size.

4.2.6 unconnected_send

4.2.6.1 Usage

The unconnected_send shall allow an application to send a message to a device without first setting up a connection. The unconnected_send service shall use the UCMM and connection manager in the intermediate nodes to forward the message and remember the return path. The method of forwarding the message shall be the same as the fwd_open but no connection shall be built. The unconnected_send service shall be sent to the local connection manager and shall be sent between intermediate nodes. When an intermediate node removes the last path segment, the message shall be formatted as a simple UCMM message and sent to the port and link address of the last path segment. Therefore, the end node shall never see the unconnected_send service but only a standard message arriving via the UCMM. The message shall be treated as a standard UCMM message by the message router and sent to the object specified in the path of the message.

Since a message may be an odd number of bytes in length (e.g. a Set_Attribute_Single to an attribute whose size is a single byte), a pad byte may be required. The pad byte shall allow devices to be aligned on a 16 bit boundary. A pad byte shall be added to the unconnected_send Service whenever the Message Size is an odd number of bytes. The Message Size shall not be adjusted to include the size of the pad byte.

The Originator of the unconnected_send shall be responsible for adding the pad byte. The intermediate node which removes the last path segment shall be responsible for also removing the pad byte from the message if it is present. The message shall then be sent to the last port and link address in the path segment (i.e. target node).

4.2.6.2 Format of the unconnected_send service

The format of the unconnected_send service shall be as shown in Table 4-19:

Table 4-19: Format of the unconnected_send service

Function	Bytes in the service	Notes
Service	[52]	Unconnected Send
Size of path	[ss] = [01]	size of path in words
Class segment for connection manager	[SS][cc] = [20][06]	SS = logical class segment 8-bit format cc = class of connection manager
Conn. priority/tick time	[pssss]	1 = high; 0 = low (priority of the UCMM message). ssss = Time/Tick 0 = 1 ms, 1 = 2 ms, 2 = 4 ms, etc.
Conn. time-out ticks	[t]	t = connection time-out ticks time-out = ticks * time/tick
Message Size	[ssss]	Size of Message in bytes not including the optional pad byte. The size of this field is 16 bits.
Message	[mmmm]...[mmmm]	Variable size message of length Message Size. Same format as connected message.
Optional Pad Byte	[pb]	pb = Optional Pad Byte. If the Message Size is an odd number of bytes (not evenly divisible by two), then this pad byte shall be present.
Connection path size	[ss][rb]	Size of connection path rb = reserve byte shall be set to zero
Connection path	[seg]...[seg]	seg = address segment

NOTES:

- 1 Required fields for the Unconnected Message
- 2 Only required in the local interface is sending messages via the message router

There is no logical instance defined in the path for this message. An optional logical instance segment is allowed. However, if there is no instance in the path, instance 1 shall be assumed.

4.2.7 unconnected_reply

4.2.7.1 Usage

The unconnected_reply service shall be generated by the last intermediate node from the UCMM reply generated by the target node or by an intermediate node as the result of a UCMM time-out, a problem with the embedded message, or a problem with the Unconnected Service Request itself. The packet shall be routed from intermediate node to intermediate node using the information stored when the unconnected_send was processed. The reply shall contain a header with status information about the request and a variable length reply generated by the target node.

4.2.7.2 Format of `unconnected_reply` if success

This format shall be used when an unconnected message reply is received. The application reply shall be the format of the Message reply from the target and shall contain error codes which resulted from the execution of the message by the application at the target node.

Table 4-20: Connection success format of `unconnected_reply`

Function	Bytes in the service	Notes
Reply Service	[sv][00]	Reply to service sv with path size 0
Status	[ss][oo]	ss = general status code; oo = size of object specific status/event data in words Note: this status code may be non-zero. This indicates that the message made it to the target node, but that the target node detected an error in the encapsulated Message and is returning that error.
Application Reply	[xxxx]....[xxxx]	Array of words for the application reply (this is the same format as any connected reply message)

4.2.7.3 Format of `unconnected_reply` if failure

This format shall be used if the originating node or any intermediate node found a problem in the message format, message parameters, or the UCMMs timed-out due to no acknowledge or no reply was received. A UCMM time-out may be the result of

- The target device not being connected or powered up. (link specific time-out);
- An intermediate device (bridge) not being connected or powered up. (link specific time-out);
- A target device being out of UCMM buffers or overloaded (Flow Control). (link specific time-out);
- An intermediate device (bridge) being out of UCMM buffers or overloaded (Flow Control). (link specific time-out);
- An invalid path segment type (logical or connection segment);
- An invalid path segment address (link or port address invalid).

Since the Unconnected message system does not implement any flow control the failure of an `unconnected_send` message need not mean a device is not present in the system.. The format of the `unconnected_reply` service for a failure shall be as shown in Table 4-21:

Table 4-21: Failure format of `unconnected_reply`

Function	Bytes in the service	Notes
Reply Service	[D2][00]	Reply service with path size 0
Status	[ss][oo]	ss = general status code; oo = size of object-specific status/event data in words
Object-specific status Reason for failure	[rrrr]	rrrr = Reason for resource failure This field is present only if oo is non-zero
Remaining Connection path size	[ss][rb]	Size of remaining connection path in 16 bit words rb = reserved byte shall be set to zero

In the failure response, the remaining Connection Path Size shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the “pre-stripped” size or 0 for the remaining Connection Path Size.

5 Connection Manager Parameters

5.1 Scope

The following parameters are used in the various PDUs of the connection manager request and reply.

5.2 Network connection parameters

5.2.1 Format

Network connection parameters shall be provided as a single 16 bit word which contains five fields. The fields within the 16 bit word shall indicate

- the type of network connection desired;
- whether the buffer size is variable or fixed;
- the priority of the connection;
- the size of the connection buffer required.

The size of the connection buffer shall include any transport header.

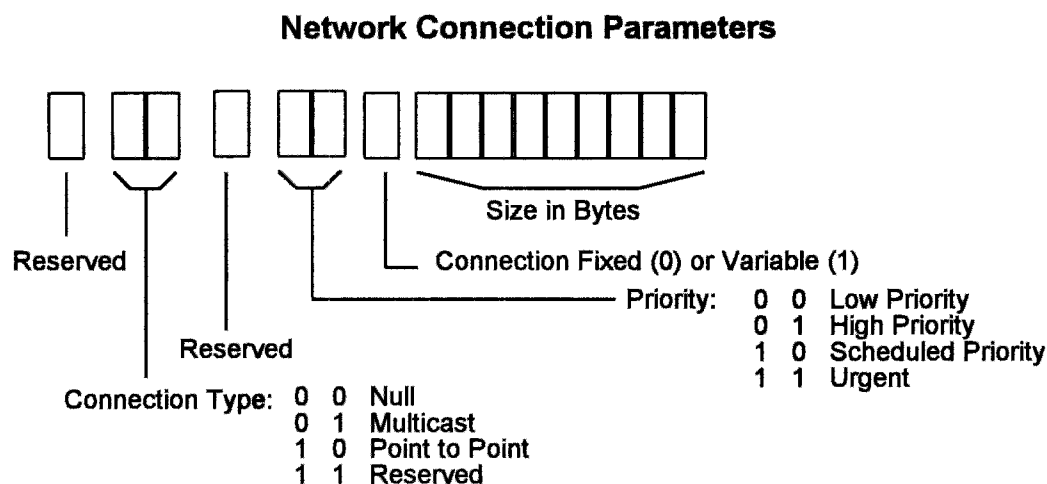


Figure 4-17: Network connection parameters

5.2.2 Connection type

The connection type shall be one of null, multicast, or point-to-point. The null type shall indicate that no network connection is required, while multicast and point-to-point refer to the network connection types as defined in clause 7. The null connection type may be used to reconfigure a connection.

5.2.3 Priority

Priority shall be one of LOW, HIGH, and SCHEDULED as specified in part 3, Data Link Layer. Although SCHEDULED priority is observed on the links, nodes may low priority data while another node has high priority data to send.

5.2.4 Connection fixed/variable

The connection can be setup to use fixed or variable sized connections. With a fixed size connection, each transmission on the connection shall use the same size buffer. Otherwise, either a buffer overrun or underrun condition shall occur. With a variable sized connection, each transmission on the connection may send a variable amount of data up to the maximum size which shall be specified when the connection is opened. Buffer underruns shall not be reported, but a buffer overrun can still occur if too much data is sent.

5.2.5 Connection size

The network connection size shall be the size of the buffer required for the connection which includes the data and any transport header. For a variable sized connection, the size shall be the maximum size of the buffer for any transfer. The actual size of the transfer for a variable connection shall be equal to or less than the size specified for the network connection. The maximum buffer size allowed shall be dependent on the links which the connection traverses.

5.3 Packet interval

5.3.1 Requested Packet Interval (RPI)

The requested packet interval shall be the requested time between packets in microseconds. The format of the RPI shall be a 32 bit integer in microseconds.

5.3.2 Actual Packet Interval (API)

The actual packet interval shall be the actual time between packets in microseconds. The format of the API shall be a 32 bit integer in microseconds.

5.3.3 Usage

The requested packet interval shall be the time between packets requested by the receiving device. The value shall be used to allocate bandwidth at each of the producing nodes. The allocation of bandwidth may have to be adjusted when the actual packet rate or actual packet interval is returned, since it is possible for the two values to differ. The path time-out value at each of the intermediate and target nodes shall also be set to the connection time-out multiplier times the API. The RPI is therefore required for all connections.

5.3.4 Function of priorities

5.3.4.1 Scheduled priority

For scheduled priority, the RPI shall be the packet rate of the repetitive data. On links which support bandwidth allocation, bandwidth shall be reserved for this packet. For scheduled priority, the data shall also be restricted to the specified packet rate, which means that if data arrives at an intermediate node faster than the specified packet rate, the node shall filter the packets to the specified rate. Since each node's scheduled priority update rate is in discrete quanta, the Actual Packet Interval (API), may be higher or lower than the RPI. The path time-out value shall be set to the Connection Time-out multiplier times the API.

5.3.4.2 High priority

For high priority, the RPI shall be used to set the path time-out in the intermediate and target nodes. The RPI requested shall therefore be set to the slowest packet rate expected, which shall preclude having the connection close due to a path time-out. The longer the path time-out, the longer the time required to reclaim resources in the intermediate nodes as a result of faults in the network. Since the high priority is not *quantised* at any of the nodes, the API shall equal the RPI. To maintain consistency, however, the time-out value shall again be set to the Connection Time-out multiplier times the API.

5.3.4.3 Low priority

For low priority, the RPI shall be used to set the path time-out in the intermediate and target nodes. The RPI shall therefore be set to the slowest packet rate expected, which shall preclude having the connection close due to a path time-out. The longer the path time-out, the longer the time required to reclaim resources in the intermediate nodes as a result of faults in the network. Since the low priority is not quantised at any of the nodes, the API shall equal the RPI. To maintain consistency, however, the time-out value shall again be set to the Connection Time-out multiplier times the API.

5.4 Connection Time-out Multiplier

The Connection Time-out Multiplier specifies the multiplier applied to the RPI to obtain the connection time-out value. The multiplier shall be represented by the following table:

Table 4-22 Time-out multiplier

Value	Multiplier	Notes
0	X 4	Default
1	X 8	
2	X 16	
3	X 32	
4	X 64	
5	X 128	
6	X 256	
7	X 512	
8 - 255	reserved	

5.5 Connection request priority/tick time

Two values are packed into this field. The connection request priority determines the priority of the UCMM messages used to establish the connection and has no correlation with the priority of the connection being made. Low priority shall be used for all connection establishment, with High priority reserved for those special circumstances which require a connection be made quickly. The use of low priority was chosen to avoid congestion with time-critical messaging. UCMM messages can not use scheduled priority.

The tick time shall be used in conjunction with the connection request time-out ticks value. This value determines the time between "ticks" in the latter field. Thus, if the tick time is 1ms., then a connection request time-out value of 5 would translate into 5ms. If the tick time was 2 ms., then a connection request time-out value of 5 would translate into 10 ms.