DRAFT INTERNATIONAL STANDARD ISO/IEC 14908-1

Attributed to ISO/IEC JTC 1 by the Central Secretariat (see page iii)

Voting begins on
2007-12-21

Voting terminates on
2008-05-21

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОММИСИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

FAST-TRACK PROCEDURE

Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol —

Part 1:
Protocol Stack

Communication de données ouverte en immotique, contrôles et gestion d'immeuble — Protocole de réseau de contrôle —

Partie 1: Pile de protocole

ICS 35.200; 35.240.99

In accordance with the provisions of Council Resolution 21/1986 this DIS is circulated in the English language only.

Conformément aux dispositions de la Résolution du Conseil 21/1986, ce DIS est distribué en version anglaise seulement.

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENT AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© International Organization for Standardization, 2007
© International Electrotechnical Commission, 2007

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

Copyright notice

This ISO document is a Draft International Standard and is copyright-protected by ISO. Except as permitted under the applicable laws of the user's country, neither this ISO draft nor any extract from it may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, photocopying, recording or otherwise, without prior written permission being secured.

Requests for permission to reproduce should be addressed to either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Reproduction may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

© ISO/IEC 2007 — All rights reserved

Not for Resale

NOTE FROM ITTF

This draft International Standard is submitted for JTC 1 national body vote under the Fast-Track Procedure.

In accordance with Resolution 30 of the JTC 1 Berlin Plenary 1993, the proposer of this document recommends assignment of ISO/IEC 14908-1 to JTC 1/SC 25.

"FAST-TRACK" PROCEDURE

1 Any P-member and any Category A liaison organization of ISO/IEC JTC 1 may propose that an existing standard from any source be submitted directly for vote as a DIS. The criteria for proposing an existing standard for the fast-track procedure are a matter for each proposer to decide.

2 The proposal shall be received by the ITTF which will take the following actions.

2.1 To settle the copyright and/or trade mark situation with the proposer, so that the proposed text can be freely copied and distributed within JTC 1 without restriction.

2.2 To assess in consultation with the JTC 1 secretariat which SC is competent for the subject covered by the proposed standard and to ascertain that there is no evident contradiction with other International Standards.

2.3 To distribute the text of the proposed standard as a DIS. In case of particularly bulky documents the ITTF may demand the necessary number of copies from the proposer.

3 The period for combined DIS voting shall be six months. In order to be accepted the DIS must be supported by 75 % of the votes cast (abstention is not counted as a vote) and by two-thirds of the P-members voting of JTC 1.

4 At the end of the voting period, the comments received, whether editorial only or technical, will be dealt with by a working group appointed by the secretariat of the relevant SC.

5 If, after the deliberations of this WG, the requirements of 3 above are met, the amended text shall be sent to the ITTF by the secretariat of the relevant SC for publication as an International Standard.

If it is impossible to agree to a text meeting the above requirements, the proposal has failed and the procedure is terminated.

In either case the WG shall prepare a full report which will be circulated by the ITTF.

6 If the proposed standard is accepted and published, its maintenance will be handled by JTC 1.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Contents

page

Foreword 7

Introduction 8

1 Scope 9

2 Normative references 9

3 Terms and definitions 9

4 Symbols and abbreviations 11
4.1 Symbols and Graphical Representations 11
4.2 Abbreviations 13

5 Overview of Protocol Layering 13

6 MAC Sublayer 15
6.1 Service Provided 15
6.2 Interface to the Link Layer 15
6.3 Interface to the Physical Layer 16
6.4 MPDU Format 17
6.5 Predictive *p*-persistent CSMA — Overview Description 18
6.6 Idle Channel Detection 19
6.7 Randomising 19
6.8 Backlog Estimation 19
6.9 Optional Priority 20
6.10 Optional Collision Detection 21
6.11 Beta1, Beta2 and Preamble Timings 22

7 Link Layer 24
7.1 Assumptions 24
7.2 Service Provided 24
7.3 CRC 24
7.4 Transmit Algorithm 25

8 Network Layer 25
8.1 Assumptions 25
8.2 Service Provided 27
8.3 Service Interface 27
8.4 Internal Structuring of the Network Layer 28
8.5 NPDU Format 28
8.6 Address Recognition 29
8.7 Routers 29
8.8 Routing Algorithm 30
8.9 Learning Algorithm — Subnets 30

9 Transaction Control Sublayer 30
9.1 Assumptions 30
9.2 Service Provided 31
9.3 Service Interface 31
9.4 State Variables 32
9.5 Transaction Control Algorithm 32

10 Transport Layer 32
10.1 Assumptions 32
10.2 Service Provided 33
10.3 Service Interface 34
10.4 TPDU Types and Formats 34

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.5 Protocol Diagram 35
10.6 Transport Protocol State Variables 36
10.7 Send Algorithm 36
10.8 Receive Algorithm 37
10.9 Receive Transaction Record Pool Size and Configuration Engineering 37
10.9.1 General 37
10.9.2 Number of Retries 37
10.9.3 Transport Layer Timers 39

11 Session Layer 39
11.1 Assumptions 39
11.2 Service Provided 40
11.3 Service Interface 40
11.4 Internal Structure of the Session Layer 41
11.5 SPDU Types and Formats 41
11.6 Protocol Timing Diagrams 43
11.7 Request-Response State Variables 46
11.8 Request-Response Protocol — Client Part 46
11.9 Request-Response Protocol — Server Part 46
11.10 Request-Response Protocol Timers 47
11.11 Authentication Protocol 47
11.12 Encryption Algorithm 47
11.13 Retries and the Role of the Checksum Function 48
11.14 Random Number Generation 49
11.15 Using Authentication 49

12 Presentation/Application Layer 49
12.1 Assumptions 49
12.2 Service Provided 49
12.3 Service Interface 50
12.4 APDU Types and Formats 51
12.5 Protocol Diagrams 52
12.6 Application Protocol State Variables 54
12.7 Request - Response Messaging in Offline State 55
12.8 Network Variables 55
12.8.1 General 55
12.8.2 Network Variable Processing 55
12.9 Error Notification to the Application Program 56
12.9.1 General 56
12.9.2 Error Notification for Messages 56
12.9.3 Error Notification for Network Variables 56

13 Network Management & Diagnostics 57
13.1 Assumptions 57
13.2 Services Provided 57
13.3 Network Management and Diagnostics Application Structure 57
13.4 Node States 57
13.5 Using the Network Management Services 58
13.5.1 General 58
13.5.2 Addressing Considerations 58
13.5.3 Making Network Configuration Changes 59
13.5.4 Downloading an Application Program 59
13.5.5 Error Handling Conditions (Informative) 60
13.6 Using Router Network Management Commands 62
13.7 NMPDU Formats and Types 63
13.7.1 General 63
13.7.2 Query ID 63
13.7.3 Respond to Query 64
13.7.4 Update Domain 64
13.7.5 Leave Domain 64
13.7.6 Update Key 64

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.7.7 Update Address 64
13.7.8 Query Address 64
13.7.9 Query Network Variable Configuration 65
13.7.10 Update Group Address 65
13.7.11 Query Domain 65
13.7.12 Update Network Variable Configuration 65
13.7.13 Set Node Mode 65
13.7.14 Read Memory 66
13.7.15 Write Memory 66
13.7.16 Checksum Recalculate 66
13.7.17 Install 66
13.7.18 Memory Refresh 82
13.7.19 Query SI 82
13.7.20 Network Variable Value Fetch 82
13.7.21 Manual Service Request Message 82
13.7.22 Network Management Escape Code 82
13.7.23 Router Mode 83
13.7.24 Router Clear Group or Subnet Table 83
13.7.25 Router Group or Subnet Table Download 83
13.7.26 Router Group Forward 83
13.7.27 Router Subnet Forward 84
13.7.28 Router Do Not Forward Group 84
13.7.29 Router Do Not Forward Subnet 84
13.7.30 Router Group or Subnet Table Report 84
13.7.31 Router Status 84
13.7.32 Router Half Escape Code 84
13.8 DPDU Types and Formats 84
13.8.1 General 84
13.8.2 Query Status 84
13.8.3 Proxy Status 88
13.8.4 Clear Status 88
13.8.5 Query Transceiver Status 88

Annex A Reference Implementation (Normative) 89
A.1 General 89
A.2 Predictive CSMA Algorithm 89
A.3 LPDU Transmit Algorithm 148
A.4 LPDU Receive Algorithm 150
A.5 Routing Algorithm 153
A.6 Learning Algorithm 153
A.7 Transaction Control Algorithm 154
A.8 Network Layer Algorithm 161
A.9 TPDU and SPDU Send Algorithm with Authentication 177
A.10 Application Layer 232
A.11 Network Management Commands 287
A.12 Configuration Data Structures 324
A.13 Include Files for the Reference Implementation 343
A.14 Application Protocol State Variables and Address Recognition Structures 373
A.15 Query-id Data Structures 375
A.16 Respond to Query Data Structure 376
A.17 Update Domain Data Structures 376
A.18 Leave Domain Data Structures 376
A.19 Update Key Data Structures 376
A.20 Update Address Data Structures 377
A.21 Query Address Data Structures 378
A.22 Query NV Cnfg Data Structures 378
A.23 Update Group Address Data Structures 378
A.24 Query Domain Data Structures 378
A.25 Update Network Variable Configuration Data Structures 379
A.26 Set Node Mode Data Structures 379

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.27 Read Memory Data Structures 380
A.28 Write Memory Data Structures 380
A.29 Checksum Recalculate Data Structures 380
A.30 Install Command Data Structures 381
A.31 Memory Refresh Data Structures 389
A.32 Query SI Data Structures 389
A.33 NV Fetch Data Structures 390
A.34 Manual Service Request Message Data Structures 390
A.35 Product Query Data Structures 390
A.36 Router Mode Data Structures 390
A.37 Router Table Clear Group or Subnet Table Data Structures 391
A.38 Router Group or Subnet Download Data Structures 391
A.39 Router Group Forward Data Structures 391
A.40 Router Subnet Forward Data Structures 391
A.41 Router Group No-Forward Data Structures 392
A.42 Router Subnet No-Forward Data Structures 392
A.43 Group / Subnet Table Report Data Structures 392
A.44 Router Status Data Structures 392
A.45 Query Status Data Structures 393
A.46 Proxy Status Data Structures 393
A.47 Clear Status Data Structures 394
A.48 Query Transceiver Status Data Structures 394

Annex B Additional Data Structures (Normative) 395
B.1 General 395
B.1.1 The System Image 395
B.1.2 The Application Image 395
B.1.3 The Network Image 396
B.2 Read-Only Structures 396
B.2.1 Fixed Read-Only Data Structures 396
B.2.2 Read-only Structure Field Descriptions 398
B.3 Domain Table 401
B.3.1 Domain Table Field Descriptions 402
B.4 Address Table 402
B.4.1 Declaration of Group Address Format 403
B.4.2 Group Address Field Descriptions 403
B.4.3 Declaration of Subnet/Node Address Format 404
B.4.4 Subnet/Node Address Field Descriptions 404
B.4.5 Declaration of Broadcast Address Format 404
B.4.6 Broadcast Address Field Descriptions 404
B.4.7 Declaration of Turnaround Address Format 405
B.4.8 Turnaround Address Field Descriptions 405
B.4.9 Declaration of Protocol Processor's Address Format 405
B.4.10 Protocol Processor Address Field Descriptions 405
B.4.11 Timer Field Descriptions 406
B.5 Network Variable Tables - Informative 407
B.5.1 Network Variable Configuration Table Field Descriptions - Informative 408
B.5.2 Network Variable Alias Table Field Descriptions - Informative 409
B.5.3 Network Variable Fixed Table Field Descriptions - Informative 409
B.6 Self-Identification Structures 409
B.6.1 SI Structure Field Descriptions 410
B.6.2 NV Descriptor Table Field Descriptions 411
B.6.3 SNVT Table Extension Records 411
B.6.4 SNVT Alias Field Descriptions 412
B.6.5 Version 2 SI Data 412
B.7 Configuration Structure 416
B.7.1 General 416
B.7.2 Configuration Structure Field Descriptions 417
B.8 Statistics Relative Structure 418

Annex C Behavioral Characteristics (Informative) 420

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

C.1 Channel Capacity and Throughput 420
C.2 Network Metrics 421
C.3 Transaction Metrics 422
C.4 Boundary Conditions — Power-Up 423
C.5 Boundary Conditions — High Load 423

Annex D PDU Summary (Normative) 424

Annex E Naming and Addressing (Normative) 426
E.1 Address Types and Formats 426
E.2 Domains 426
E.3 Subnets and Nodes 427
E.4 Groups 427
E.5 Unique_Node_ID and Node Address Assignment 428
E.6 NPDU Addressing 429

Annex F List of patents that pertain to this European Standard (Normative) 431

Bibliography 434

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

This European Standard (EN 14908-1:2005) has been prepared by CEN /TC 247, "Building Automation, Controls and Building Management", the secretariat of which is held by SNV.

This European Standard shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by May 2006, and conflicting national standards shall be withdrawn at the latest by May 2006.

This European Standard supersedes ENV 13154—2:1998.

This publication is copyright under the Berne Convention and the Universal Copyright Convention. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means, electronic, mechanical, photocopying, recording, or otherwise, without the permission of the European Committee for Standardization (CEN), the European Committee for Electrotechnical Standardization (CENELEC), their National Standards Bodies and their Licensees to reproduce this European Standard in full and including this copyright notice for the purposes of European standardisation.

CEN draws attention to the fact that it is claimed that compliance with this European Standard may involve the use of patents. The patents that pertain to this European Standard are listed in Annex F.

CEN/TC247 confirms that this European Standard contains patents and like rights claiming by Echelon Corporation. The Echelon Corporation declared to CEN its willingness to negotiate licenses under patents or rights with applicants throughout the world on reasonable terms and conditions without any discrimination.

NOTE For licensing information, contact: Echelon Corporation, 4015 Meridian Avenue, San Jose, CA 94304, USA, phone +1-408-938-5234, fax: +1-408-790-3800 http://www.echelon.com.

This European Standard is part of a series of European Standards for open data transmission in building automation, control and in building management systems. The content of this standard covers the data communications used for management, automation/control and field functions. This European Standard is based on the American standards EIA/CEA-709.1-B Control Network Protocol Specification.

The EN 14908-1 is part of a series of European Standards under the general title *Control Network Protocol (CNP)*, which comprises the following parts:

Part 1: *Protocol Stack*

Part 2: *Twisted Pair Communication*

Part 3: *Power Line Channel Specification*

Part 4: *IP-Communication*

Part 5 : *Project Implementation Guideline*

According to the CEN/CENELEC Internal Regulations, the national standards organizations of the following countries are bound to implement this European Standard: Austria, Belgium, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Slovakia, Slovenia, Spain, Sweden, Switzerland and United Kingdom.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

This European Standardhas been prepared to provide mechanisms through which various vendors of building automation, control, and building management systems may exchange information in a standardised way. It defines communication capabilities.

This European Standard is to be used by all involved in design, manufacture, engineering, installation and commissioning activities.

This European Standard has been made in response to the essential requirements of the Constructive Products Directive.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

1 Scope

This specification applies to a communication protocol for networked control systems. The protocol provides peer-to-peer communication for networked control and is suitable for implementing both peer-to-peer and master-slave control strategies. This specification describes services in layers 2 - 7. In the layer 2 (data link layer) specification, it also describes the MAC sub-layer interface to the physical layer. The physical layer provides a choice of transmission media. The interface described in this specification supports multiple transmission media at the physical layer. In the layer 7 specification, it includes a description of the types of messages used by applications to exchange application and network management data.

2 Normative references

Not applicable

3 Terms and definitions

For the purposes of this European Standard, the following subclause introduces the basic terminology employed throughout this European Standard. Most of it is commonly used and the terms have the same meaning in both the general and the standard context. However, for some terms, there are subtle differences. For example, in general, bridges do selective forwarding based on the layer 2 destination address. There are no layer 2 addresses in this standard protocol, so bridges forward all packets, as long as the domain address in the packet matches a domain of which the bridge is a member. Routers, in general, perform network address modification so that two protocols with the same transport layer but different network layers can be connected to form a single logical network. Routers of this standard may perform network address modification, but typically they only examine the network address fields and selectively forward packets based on the network layer address fields.

3.1
Channel
physical unit of bandwidth linking one or more communication nodes. Refer to Annex E for further explanation of the relationship between a channel and a subnet

3.2
Physical Repeater
device that reconditions the incoming physical layer signal on one channel and retransmits it on to another channel

3.3
Store-and-Forward Repeater
device that stores and then reproduces data packets on to a second channel

3.4
Bridge
device that connects two channels (x and y); forwards all packets from x to y and vice versa, as long as the packets originate on one of the domain(s) that the bridge belongs to

3.5
Configuration
non-volatile information used by the device to customise its operation. There is configuration data for the correct operation of the protocol in each device, and optionally, for application operation. The network configuration data stored in each device has a checksum associated with the data. Examples of network configuration data are node addresses, communication media parameters such as priority settings, etc. Application configuration information is application specific

3.6
Domain
virtual network that is the network unit of management and administration. Group and subnet (see below) addresses are assigned by the administrator responsible for the domain, and they have meaning only in the context of that domain

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.7
Flexible Domain
used in conjunction with Unique_Node_ID and broadcast addressing. A node responds to a Unique_Node_ID-addressed message if the address matches, regardless of the domain on which the message was sent. To respond so that the sender receives it, the response must be sent on the domain in which it was received. Furthermore, this domain must be remembered for the duration of the transaction so that duplicate detection of any retries is possible. This transitory domain entry at a node is called the flexible domain. How many flexible domain entries a node supports is up to the implementation. However, a minimum of 1 is required

3.8
Subnet
set of nodes accessible through the same link layer protocol; a routing abstraction for a channel; in this standard subnets are limited to a maximum of 127 nodes

3.9
Node
abstraction for a physical node that represents the highest degree of address resolvability on a network. A node is identified (addressed) within a subnet by its (logical) node identifier. A physical node may belong to more than one subnet; when it does, it is assigned one (logical) node number for each subnet to which it belongs. A physical node may belong to at most two subnets; these subnets must be in different domains. A node may also be identified (absolutely) within a network by its Unique_Node_ID

3.10
Group
uniquely identifiable set of nodes within a domain. Within this set, individual members are identified by their member number. Groups facilitate one-to-many communication and are intended to support functional addressing

3.11
Router
device that routes data packets to their respective destinations by selectively forwarding from subnet to subnet; a router always connects two (sets of) subnets; routers may modify network layer address fields. Routers may be set to one of four modes: repeater mode, bridge mode, learning mode, and configured mode. In repeater mode, packets are forwarded if they are received with no errors. In bridge mode, packets are forwarded if they are received with no errors and match a domain that the router is a member of. Routers in learning mode learn the topology by examining packet traffic, while routers that are set to configured mode have the network topology stored in their memory and make their routing decisions solely upon the contents of their configured tables

3.12
(Application) Gateway
interconnects networks at their highest protocol layers (often two different protocols). Two domains can also be connected through an application gateway

3.13
Beta1
period immediately following the end of a packet cycle. A node attempting to transmit monitors the state of the channel, and if it detects no transmission during the Beta1 period, it determines the channel to be idle

3.14
Beta2
randomising slot. A node wishing to transmit generates a random delay T. This delay is an integer number of randomising slots of duration Beta2

3.15
Network Variable
variable in an application program whose value is automatically propagated over the network whenever a new value is assigned to it

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.16
Standard Network Variable Types (SNVTs)
variables with agreed-upon semantics. These variables are interpreted by all applications in the same way, and are the basis for interoperability. Definition of specific SNVTs is beyond the scope of this European Standard

3.17
Manual service request Message
network management message containing a node's Unique_Node_ID. Used by a network management device that receives this message to install and configure the node. May be generated by application or system code. May be triggered by external hardware event, e.g., driving a "manual service request" input low

3.18
Transaction
sequence of messages that are correlated together. For example, a request and the responses to the request are all part of a single transaction. A transaction succeeds when all the expected messages from every node involved in the transaction are received at least once. A transaction fails in this European Standard if any of the expected messages within the transaction are not received. Retries of messages within a transaction are used to increase the probability of success of a transaction in the presence of transient errors

4 Symbols and abbreviations

4.1 Symbols and Graphical Representations

Figure 1 shows the basic topology of networks based on this protocol and the symbolic representations used in this European Standard.

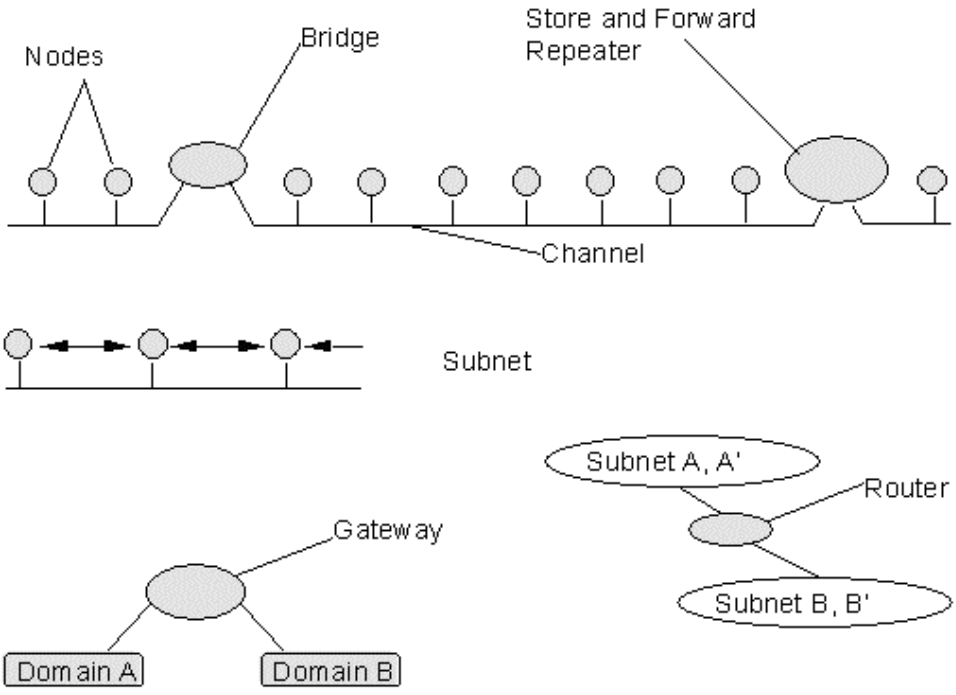

Figure 1 — Network Topology & Symbols

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The layering of this protocol is described using standard OSI terminology, as shown in Figure 2.

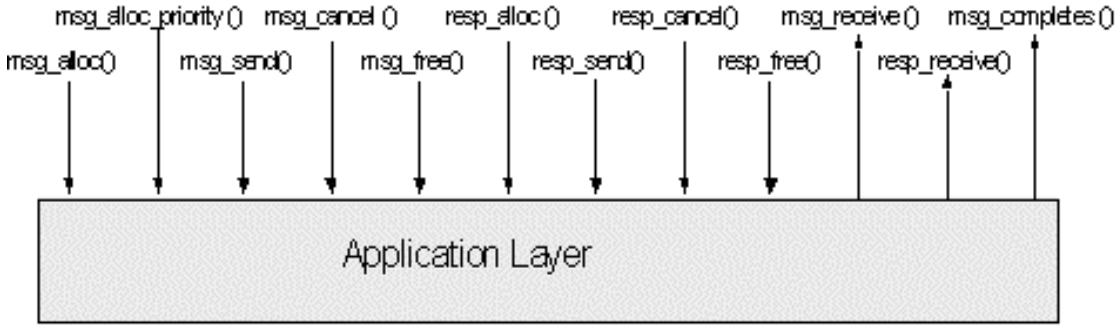

Figure 2 — Protocol Terminology

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.2 Abbreviations

— CNP Control Network Protocol

The Protocol Data Unit (PDU) abbreviations used throughout this European Standard are:

— PPDU Physical Protocol Data Unit, or frame

— MPDU MAC Protocol Data Unit, or frame

— LPDU Link Protocol Data Unit, or frame

— NPDU Network Protocol Data Unit, or packet

— TPDU Transport Protocol Data Unit, or a message/ack

— SPDU Session Protocol Data Unit, or request/response

— NMPDU Network Management Protocol Data Unit

— DPDU Diagnostic Protocol Data Unit

— APDU Application Protocol Data Unit

— FSM Finite State Machine (diagram)

Annex D (PDU Summary) contains the details of these PDUs.

5 Overview of Protocol Layering

The protocol specified by this European Standard consists of the layers shown in Figure 3. Each layer is described below.

Multiple physical layer protocols and data encoding methods are allowed in systems based on this European Standard. Each encoding scheme is medium-dependent.

The *MAC* (Medium Access Control) sublayer employs a collision avoidance algorithm called Predictive *p*-persistent CSMA (Carrier Sense, Multiple Access). For a number of reasons, including simplicity and compatibility with the multicast protocol, the link *layer* supports a simple connectionless service. Its functions are limited to framing, frame encoding, and error detection, with no error recovery by re-transmission.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

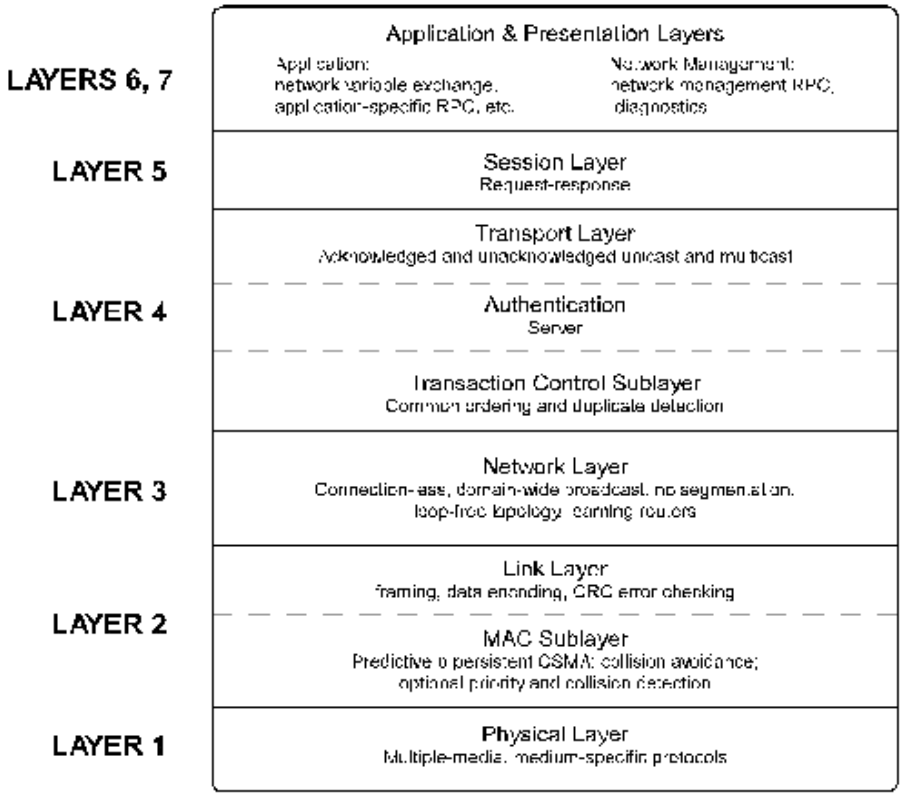

Figure 3 — Protocol Layering

The *Network* layer handles packet delivery within a single domain, with no provisions for inter-domain communication. The Network service is connection-less, unacknowledged, and supports neither segmentation nor re-assembly of messages. The routing algorithms employed by the network layer to learn the topology assumes a tree-like network topology; routers with configured tables may operate on topologies with physical loops, as long as the communication paths are logically tree-like. In this topology, a packet may never appear more than once at the router on the side on which the packet originated. The unicast routing algorithm uses learning for minimal overhead and no additional routing traffic. Use of configured routing tables is supported for both unicast and group addresses, although in many applications a simple flooding of group addressed messages is sufficient.

The heart of the protocol hierarchy is the *Transport* and *Session* layers. A common *Transaction Control Sublayer* handles transaction ordering and duplicate detection for both. The *Transport* layer is connection-less and provides reliable message delivery to both single and multiple destinations. Authentication of the message sender's identity is included as a transport layer service, for use when the security of sender authentication is required. The authentication server requires only the Transaction Control Sublayer to accomplish its function. Thus Transport and Session layer messages may be authenticated using all of the addressing modes other than broadcast.

The session layer provides a simple Request-Response mechanism for access to remote servers. This mechanism provides a platform upon which application specific remote procedure calls can be built. The network management protocol, for example, depends upon the Request-Response mechanism in the Session layer.

A transport layer acknowledged message expects indication of message delivery from remote destination(s). A session layer request message expects indication that application-specific remote task(s) have been completed. A given message uses only one or the other type of service, but not both.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This specification includes the *Presentation Layer* and the lowest level of the *Application Layer*. These layers provide services for sending and receiving application messages including network variables, and other types of messages such as network management and diagnostic messages and foreign frames (see clause 13). For a network variable update, the APDU header provides information on how to interpret the APDU. This application-independent interpretation of the data allows data to be shared among nodes without prior arrangement.

6 MAC Sublayer

In this European Standard the following Media Access Control sublayer is defined. If there is a need for other MAC sublayers they are defined in additional parts of this European Standard.

6.1 Service Provided

The Media Access Control (MAC) sublayer facilitates media access with optional priority and optional collision detection/collision resolution. It uses a protocol called Predictive *p*-persistent CSMA (Carrier Sense, Multiple Access), that has some resemblance to the *p*-persistent CSMA protocol family.

Predictive *p*-persistent CSMA is a *collision avoidance* technique that randomises channel access using knowledge of the expected channel load. A node wishing to transmit always accesses the channel with a random delay in the range (0..w). To avoid throughput degradation under high load, the size of the randomising window, w, is a function of estimated channel backlog BL:

$$w = (BL \times W_{base}) - 1, \quad (1)$$

where

W_{base} is the base window size. W_{base} is measured in time. Its duration, derived from Beta2 (see 6.7), equals 16 Beta2 slots.

6.2 Interface to the Link Layer

The MAC sublayer is closely coupled to the Link layer, described in clause 7. With the MAC sublayer being responsible for media access, the Link layer deals with all other layer 2 issues, including framing and error detection. For explanatory purposes, the interface between the two layers is described in the form shown in Figure 4.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

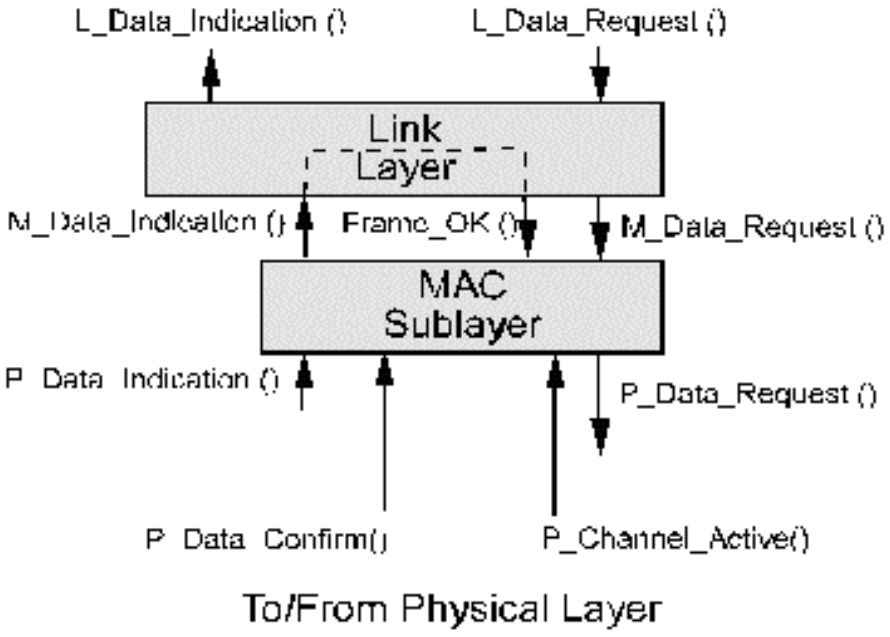

Figure 4 — Interface between the MAC and the Link layers

Although the service interface primitives are defined using a syntax similar to programming language procedure calls, no implementation technique is implied. Frame reception is handled entirely by the Link layer, that notifies the MAC sublayer about the backlog increment via the Frame_OK() primitive.

The following service interface primitives facilitate the interface between the Link and the MAC layers:

M_Data_Request (Priority, delta_BL, ALT_Path, LPDU)

This primitive is used by the Link layer to pass an outbound LPDU/MPDU to the MAC sublayer. Priority defines the priority with which the frame is to be transmitted; delta_BL is the backlog increment expected as a result of delivering this MPDU. ALT_Path is a binary flag indicating whether the LPDU is to be transmitted on the primary or alternate channel, baud rate, etc. See 6.4 for how ALT_Path is set.

Frame_OK (delta_BL)

On receiving a frame and verifying that its CRC is correct, the Link layer invokes this primitive to notify the MAC sublayer about the backlog increment associated with the frame just received.

M_Data_Indication()

The MAC sublayer provides this indication to the link layer once per incoming LPDU/MPDU.

6.3 Interface to the Physical Layer

The Physical layer handles the actual transmission and reception of binary data. Multiple physical layer protocols are supported by the control network protocol. The bit error rate presented to the link layer must be equal to or

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

better than 1 in 10^{-4}. For compatibility with the higher layers, all physical protocols must support the defined service interface (see figure 4):

P_Data_Indication (Frame)

Physical layer provides this indication to the MAC sublayer and the link layer once per in-coming LPDU/MPDU.

P_Data_Request (Frame)

The MAC sublayer uses this primitive to pass the Frame, the encoded LPDU/MPDU, to the physical layer for immediate transmission. The bit transmission order is defined in Annex D.

P_Data_Confirm (Status)

The physical layer returns Status as to whether the frame was transmitted. Status has three possible values: success—indicating the frame was transmitted, request_denied—indicating that activity was detected on the line prior to transmission, and collision—indicating that transmission began, but a collision was detected. Whether or not the transmission is aborted depends on when the collision is detected (see 6.10).

P_Channel_Active ()

The physical layer uses this primitive to pass the status of the channel to the MAC sublayer. This is an indication of activity, not necessarily of valid data.

6.4 MPDU Format

The combined MPDU/LPDU format is shown in Figure 5. (Annex D contains the details of the NPDU frame).

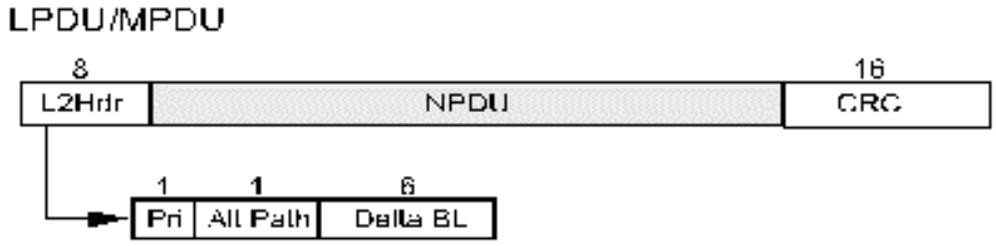

Figure 5 — MPDU/LPDU Format

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The MAC sublayer uses the L2Hdr field, that has the following syntax and semantics:

Pri 1-bit field specifying the priority of this MPDU: 0 = Normal, 1 = High.

Alt_Path 1-bit field specifying the channel to use. This is a provision for transceivers that have the ability to transmit on two different channels and receive on either one without the need to instruct the transceiver to explicitly receive on a specific channel. The transport layer sets this bit for the last two attempts (for acknowledged and request/response services), unless requested to specify the alternate path for every transmission. For any packet received that has the alt_path bit set and that requires an acknowledgement, response, challenge, or reply, the alt_path bit shall be set in the corresponding acknowledgement, response, challenge, or reply.

Delta_BL 6-bit unsigned field (≥ 0); specifies channel backlog increment to be generated as a result of deliver-ing this MPDU.

6.5 Predictive *p*-persistent CSMA — Overview Description

Like CSMA, Predictive *p*-persistent CSMA senses the medium before transmitting. A node attempting to transmit monitors the state of the channel (see Figure 6), and determines the channel to be idle if it detects no transmission during the Beta1 period. Nodes without a packet to transmit during this Beta1 period shall remain in synchronisation for the duration of the priority slots (see 6.10), and at least W_{base} randomising slots. This maintenance of synchronisation allows a packet that arrives in the output queue of the MAC sublayer after the end of the Beta1 time to be transmitted in a valid slot according to the other nodes with a packet to transmit.

Next, the node generates a random delay T (transmit) from the interval (0..(BL x W_{base})-1, where W_{base} is the number of randomising slots within the basic randomising window and BL is an estimate of the current channel backlog. T (transmit) is defined as an integer number of randomising slots of duration Beta2 (see 6.7 and 6.8). If the channel is idle when the delay expires, the node transmits; otherwise, the node receives the incoming packet, and then repeats the MAC algorithm. In Figure 6, D_{mean} is the average randomisation delay between packets, and, since the random delay T is uniformly distributed, D_{mean} is given as (W_{base}-1)/2 for small values of BL.

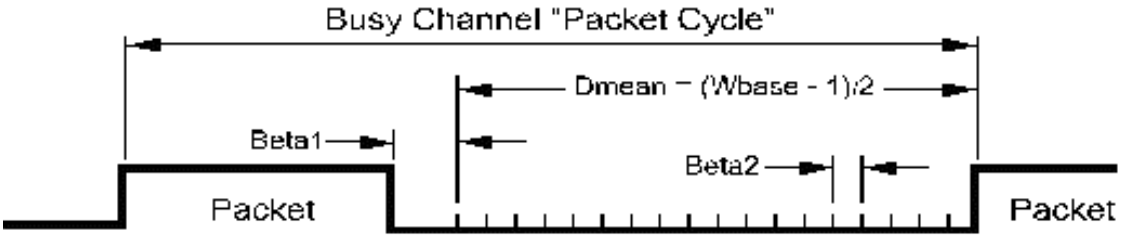

Figure 6 — Predictive *p*-persistent CSMA Concepts and Parameters

By adjusting the size of the randomising window as a function of the predicted load, the algorithm keeps the collision rate constant and independent of the load. Provided that the estimated backlog is greater than or equal to the real back-log, the following holds:

$$\text{Collision Rate} = \text{Error Pkt Cycles} / \text{Error Free Pkt Cycles} \leq 1 / 2W_{base}$$

A base window size of 16 is used. This implies that there are an average of 8 randomising slots of width Beta2 and one slot of width Beta1 between each packet. Also, the width of the Beta2 period is crucial to efficient utilisation of the channel.

The algorithm for Predictive CSMA is in Annex A.2.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

6.6 Idle Channel Detection

The idle channel condition is asserted whenever the following two conditions are met:

1) The current channel state reported by the physical layer via the P_Data_Indication () primitive is low; and

2) No transition has been detected during the last period of Beta1. Note that the MAC sublayer may be configured to ignore transitions during a portion of the Beta1 period. This portion of time that transitions are ignored (the channel is assumed to be idle during this time even in the presence of transitions) is called the indeterminate time (see 6.11 for the details).

The length of the Beta1 period is defined by the following constraint:

$$\text{Beta1} > 1\,\text{bit time} + (2 \times \text{Tau}_p + \text{Tau}_m) \quad (2)$$

The first term assumes a data encoding method that guarantees a transition and/or carrier during every bit time. If encoding methods are used that do not meet this constraint, then the first term must be adjusted to be the longest time that the channel may appear idle without being idle, i.e., the longest run in legal data transmission without a transition and/or carrier asserted on the medium. The second term takes care of propagation and turnaround delays, that are:

Tau_p is the physical propagation delay defined by the media length;

Tau_m is the detection and turnaround delay within the MAC sublayer; this is the period from the time the idle channel condition is detected, to the point when the first output transition appears on the output. On media where there is a carrier, this time must include the time between turning on the carrier, and it being asserted as a valid carrier on the medium.

6.7 Randomising

At the beginning of the randomising period, a node wishing to transmit generates a random delay T (transmit) from the interval (0.. (BL x W_{base})-1. The node then waits for this period, while continuing to monitor channel status; if the channel is still idle when the delay expires, the node transmits.

The transmit delay T (transmit) is an integer number of randomising slots of duration Beta2; the length of the randomising slot must meet the following constraint:

$$\text{Beta2} > 2 \times \text{Tau}_p + \text{Tau}_m \quad (3)$$

Parameters Tau_p and Tau_m are defined in 6.6.

6.8 Backlog Estimation

The predictive aspect of the MAC algorithm is based on backlog estimation. Each node maintains an estimate of the current channel backlog BL, that is incremented as a result of sending or receiving an MPDU and decrements periodically — once every packet cycle. The increment to the backlog is encoded into the link layer header, and represents the number of messages that the packet shall cause to be generated upon reception. The backlog is initially set to one. After sending or receiving a packet with a non-zero backlog increment, the node's backlog estimation is incremented by the backlog increment. The maximum backlog value is 63. If the backlog exceeds 63, then the backlog overflow statistic is incremented (see 13.7.14 and B.8).

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The backlog decrements under one of the following conditions:

— On waiting to transmit: If W_{base} randomising slots go by without channel activity.

— On receive: If a packet is received with a backlog increment of '0'.

— On transmit: If a packet is transmitted with a backlog increment of '0'.

— On idle: If a packet cycle time expires without channel activity.

The packet cycle timer is reset to its initial value whenever the backlog is changed. It is started (begins counting down at its current value) whenever the MAC layer becomes idle. An idle MAC layer is defined as:

1) not receiving;

2) not transmitting;

3) not waiting to transmit;

4) not timing Beta1;

5) not waiting for priority slots; and

6) not waiting for the first W_{base} randomising window to complete.

On transition from idle to either transmit or receive, the packet cycle timer is halted.

The backlog always has a value ≥ 1. The algorithm post-increments rather than pre-increments the backlog by the amount associated with the MPDU being transmitted, because the number of expected responses is of no importance until after transmitting the MPDU.

6.9 Optional Priority

On a channel by channel basis, the protocol supports optional priority. Priority slots, if any, follow immediately after the Beta1 period that follows the transmission of a packet (Figure 7). The number of priority slots per channel ranges from 0 to 127. Priority slots are typically not contended for, but rather are uniquely assigned to nodes on the channel. Nodes that have been assigned a priority slot do not have to use it with every message; the node decides on a message by message basis whether or not to use the assigned priority slot. This determination is made by examining the priority bit within the LPDU header (Figure 5).

It is possible to assign all the nodes on the channel the same priority slot. An example of an architecture where this makes sense would be one where there is a background of peer-to-peer activity, but a single master that cycles around doing something to each node (such as network management, polling, etc.). By giving each node the same slot, and having it used only for this purpose, these transactions (from the single master to multiple slaves) would tend to be completed ahead of the background traffic.

An application may decide that a message is high priority and attempt to send it as such. If the node does not have a priority slot assigned to it, the message shall go out in the usual way, except that the priority bit in the layer 2 header shall be set. If, subsequently, the packet passes through a router that has a priority slot on its destination channel, the packet shall be sent using the priority of the router. If the router does not have a priority slot, the message shall be for-warded in the usual way, with the priority bit in the layer 2 header remaining set.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

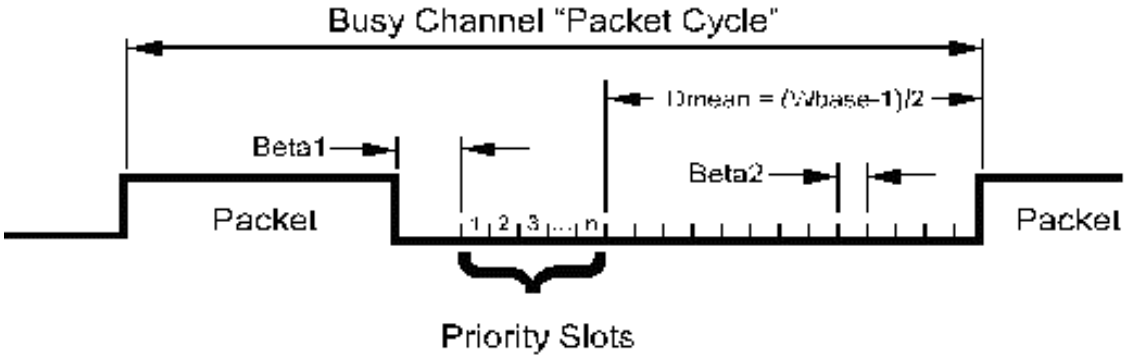

Figure 7 — Allocation of Priority Slots within the Busy Channel Packet Cycle

The protocol provides no synchronisation among the nodes. Therefore, if the channel has been idle for longer than the randomising period (Beta1 + number of priority slots + D_{mean} above), access to the link is random without regard to priority. Once the link returns to the busy state, access to the link shall be in priority order.

If a priority message requires:

- acknowledgement (acknowledged messaging);
- response (request/response messaging);
- challenge (in response to an authenticated acknowledged message or authenticated request); or
- reply (in response to an authentication challenge).

The responding node shall attempt to send a priority acknowledgement/response/challenge/reply by setting the priority bit in the layer 2 header. If a high priority message is generated within a node, it is sent prior to any queued packets of normal priority. Multiple high priority packets are sent in FIFO order. If the application attempts to send a high priority message while its node is sending a packet, the packet in progress completes first.

If a channel has priority slots and a node has multiple messages queued within it, it shall not send them in consecutive packet cycles. In the case where a node has a packet to send, and it has sent a packet in the previous packet cycle, the node does not use its normal medium access algorithm in the current cycle. Instead it attempts to access the medium using a slightly modified version of the non-priority MAC algorithm. The modified version of the algorithm is that the transmitter node waits out the Beta1 time after the priority packet it just transmitted, then it waits for all the priority slots to go by, then it waits for one window of randomising slots to go by (16 Beta2 slots). At this point it randomises its access to the network using the current channel backlog estimate. If the node is not successful in the current packet cycle, it may use its priority slot in the subsequent packet cycle.

6.10 Optional Collision Detection

The physical layer may optionally notify the MAC sublayer of a detected collision. The MAC sublayer can be configured to check for collision detection notification any time during packet transmission.

Collision resolution may be implemented by means of collision detection. A bit pattern unique to each node on the channel may be inserted before the preamble field to resolve collisions. Such a physical layer must define a sufficiently long preamble after the arbitration pattern so that transmitting nodes that lose the arbitration for a packet cycle can turn around and receive the incoming packet from the node that just won the arbitration.

Collision detection, if used, shall only be enabled upon transmission of a packet. When a collision is detected in the MAC sublayer, the backlog is incremented. Receivers do not detect collisions, rather they see them as packets that are too short, or have incorrect CRCs or sometimes they are seen as just spurious transitions for which bit

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

synchronisation cannot be achieved. When a collision is detected, the backlog is incremented, and the collision statistic is incremented (see 13.7.14 and B.8).

The MAC sublayer attempts to re-transmit the packet upon notification of a collision. It obeys the following rules:

7) If a collision is detected on two successive attempts to transmit a priority packet in the node's priority slot, the next attempt to transmit the priority packet shall not use the con-figured priority slot, but rather shall be in a slot picked ac-cording to the non-priority MAC algorithm. The priority bit in the layer 2 header of the packet shall remain set in this case.

8) Whenever a collision is detected by a transmitting node, that transmitting node increments its estimate of the channel backlog by 1.

9) Whenever a collision is detected on 256 successive attempts to transmit a packet, the packet is discarded.

6.11 Beta1, Beta2 and Preamble Timings

A node may be implemented to support one or more media. If implemented to support multiple media, its preamble length and Beta1 and Beta2 times must be configurable to support the relevant physical communication media. Beta2 is the duration in time of a media access time-slot (either randomising or priority slot). The MAC layer timers for Beta1 are provided for both the transmitter in the previous packet cycle as well as the receiver. This is to correct for differences in the time base for the end of the preceding packet between transmitters and receivers. Note that in this family of physical layer standards the timings are with reference to the timings observable on the medium itself, while these timings are in reference to the protocol processor's reference. For example, imagine a transceiver that added bits of error correction code to the actual packet. From the transmitting protocol processor's reference the end of the packet would be sent before these bits were added to the packet, while the receiving protocol processor's time reference would show packet end at the end of the packet including the error correction bits. Thus, the need for the ability to specify a difference in the timing depending upon whether a node is a transmitter or a receiver for a given packet. In addition there are two communication configurations (comm_type in B.7) with different formula for calculating Beta1 based on the type.

A node must be able to generate one of five base time periods to generate Beta1 and Beta2 slots of the right duration.

The formulae for Beta1, Beta2, and the preamble are shown below.

CT is a time base used to generate Beta1 and Beta2 slots and the preamble. Depending on the media type(s) supported, each node must support one or more of the following values for CT: 600 ns, 1.2 µs, 2.4 µs, 4.8 µs, 9.6 µs.

v, a tuning parameter, is in the range 0 to 255 inclusive. There are 3 tuning parameters. These are represented as v_1, v_2 and v_3 in the formula below.

f() is a function that returns a time delay to compensate for other slower nodes on a channel that may generate actual values of CT below the nominal for a channel.

$$\text{Beta2} = \text{CT} \times (40 + 20 \times \text{v}_1) \quad (4)$$

There are four formulae for computing Beta1. The formula to use depends on a bit field in the configuration structure, comm_type, listed in B.7, and whether the timing is done immediately following the receipt or the transmission of a packet Comm_type can have one of two values: 1 or 2. The selection of the comm_type determines the behaviour of the MAC sublayer in the following ways:

Comm_type = 1:

10) An indeterminate time is defined during the Beta 1 period in which all transitions on the channel are ignored. This period starts following the end of any packet (transmitted or received). Its duration is defined below. Following the indeterminate time and before the first Beta2 slot, the MAC sublayer repeatedly waits for a period of duration of one Beta2 slot to pass with no transitions on the channel before

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

proceeding (note that if a valid preamble is detected during this period, an attempt shall be made to receive the incoming packet). This Beta2 slot duration accounts for the Beta2 slot duration component of the total Beta1 time.

11) The transmitted length of the preamble is under the control of the MAC sublayer.

12) The MAC sublayer ignores collisions occurring during the first 25 % of the transmitted preamble. It optionally (according to the cd_tail field described in B.7) ignores collisions reported following the transmission of the CRC but prior to the end of transmission.

13) If a collision is detected during preamble transmission, the MAC sublayer can terminate the packet if so configured. Collisions detected after the preamble has been sent do not terminate transmission.

Comm_type = 2:

14) No indeterminate time is defined at the MAC sublayer. If there is an indeterminate time, it shall be enforced by the physical layer.

15) The MAC sublayer is configured to either control the transmitted length of the preamble or to have the transmitted length of the preamble controlled by the physical layer.

16) The MAC sublayer shall always terminate the packet upon notification of a collision.

17) The MAC sublayer reconfigures the physical layer prior to transmission, if necessary, to inform it of a change in transmission path selection (alternate path versus no alternate path).

The formulae for Beta1 for the four cases of transmitting, receiving, comm_type =1 and comm_type = 2 are as follows:

$$\begin{aligned}\text{Beta1(after transmission)} &= \text{CT}\times(583+f(\text{xmit_interpacket}))+\text{Beta2}\{\text{for comm_type}=1\}\\ \text{Beta1(after reception)} &= \text{CT}\times(565+f(\text{recv_interpacket}))+\text{Beta2}\{\text{for comm_type}=1\}\\ \text{Beta1(after transmission)} &= \text{CT}\times(624+f(\text{xmit_interpacket}))+\text{Beta2}\{\text{for comm_type}=2\}\\ \text{Beta1(after reception)} &= \text{CT}\times 602+f(\text{recv_interpacket})+\text{Beta2}\{\text{for comm_type}=2\}\end{aligned} \quad (5\text{-}8)$$

$$f(\text{v}) = (\text{v}<128)\,?\,(41\times \text{v}_2):145\times(\text{v}_2-128) \quad (9)$$

Parameters xmit_interpacket and recv_interpacket have ranges from 0 to 255 and are parameters stored in the Configuration Structure (Annex B.7).

The two formulae for computing the indeterminate time for the MAC sublayer are as follows:

$$\begin{aligned}\text{IDT(after transmission)} &= \text{CT} * (313+f(\text{xmit_interpacket}))\{\text{for comm_type}=1\}\\ \text{IDT (after reception)} &= \text{CT} * (295+f(\text{recv_interpacket}))\{\text{for comm_type}=1\}\end{aligned} \quad (10\text{-}11)$$

CT, *f*(x), xmit_interpacket and recv_interpacket are as defined for the Beta1 calculations. Note that in all cases the indeterminate time is constrained to be less than the total Beta1 time.

Preamble length is either controlled by the protocol processor or by the transceiver. When the protocol processor controls the preamble, the formula for all possible values of preamble length is shown below. For this formula the variable v has a range of from 0 to 253 inclusive. CT is as defined above.

$$\text{Preamble} = \text{CT}\times(219+32\times \text{v}_3) \quad (12)$$

When the transceiver controls the preamble length (an option available when comm_type = 2), the minimum preamble length allowed is 181 µs. This is the value corresponding to a protocol processor with a 600 ns timer value. This value scales with the time base, CT. When the transceiver controls the preamble length, the maximum preamble length is determined by the transceiver.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Beta1, Beta2 and preamble timings for each transceiver type compatible to this standard are noted in the relevant physical layer specification.

7 Link Layer

7.1 Assumptions

The Link layer assumes that CRC errors due to both collisions and transmission errors occur with some probability P_e, and that P_e is small enough so that link level error recovery is not needed.

The above assumption means that successful end-to-end communication is only possible when the sum of error probabilities along the communication path is much less than one.

$$SUM(P_e) << 1$$

In networks where P_e is constant, the maximum communication distance D (network or group diameter) must be:

$$D << 1 / P_e$$

7.2 Service Provided

The Link layer provides subnet-wide, ordered, unacknowledged LPDU delivery with error detection but no error recovery. A corrupted frame is discarded as soon as its CRC check fails.

7.3 CRC

The CRC is computed over the entire NPDU including the L2Hdr field. The CRC is generated using the polynomial:

$X^{16} + X^{12} + X^5 + 1$ (the CCITT CRC-16 standard).

For each NPDU, the CRC register is initialised to all ones. Figure 8 shows an example of the time behaviour of the CRC shift register that implements the above polynomial. The register bits are represented by the columns labelled bit0-bit15. The data, over which the CRC is computed, appears in the column labelled crc_in. The register is initialised to all ones. The data is then presented serially to the register starting with bit7 of byte 1. Each row of the table shows the register state at a particular bit time. The register continues to input data and compute the CRC as long as the compute mode is true (mode compute = 1). The compute mode going false signals the end of data and the end of the CRC computation.

Following data input, the sixteen (16) bits of CRC data are shifted out serially as shown by the column labelled nrz_out.

When the link layer receives a packet with a CRC error or a packet that is less than 8 bytes in length, a transmission error statistic is incremented (see 13.8.2), and the packet cycle timer is started but not reset first, unlike after the reception of a good packet. If a packet is received but is too long for the input buffer, or there are no input buffers, the missed packet statistic is incremented (see 13.8.2).

For a normative description of the CRC-16 computation, refer to the pseudocode example in A.3.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

comment	mode compute	crc_in	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7	bit 8	bit 9	bit 10	bit 11	bit 12	bit 13	bit 14	bit 15	nrz_out
initialize			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
byte1 bit7	1	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	0
6	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1
5	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
4	1	1	0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0	1
3	1	1	1	0	0	0	1	0	1	1	1	0	1	1	0	1	1	1	1
2	1	0	1	1	0	0	0	0	0	1	1	1	0	1	0	0	1	1	0
1	1	0	1	1	1	0	0	1	0	0	1	1	1	0	0	0	0	1	0
0	1	1	0	1	1	1	0	0	1	0	0	1	1	1	0	0	0	0	1
byte2 bit7	1	1	1	0	1	1	1	1	0	1	0	0	1	1	0	0	0	0	1
6	1	0	0	1	0	1	1	1	1	0	1	0	0	1	1	0	0	0	0
5	1	0	0	0	1	0	1	1	1	1	0	1	0	0	1	1	0	0	0
4	1	1	1	0	0	1	0	0	1	1		0	1	0	1	1	1	0	1
3	1	1	1	1	0	0	1	1	0	1	1	1	0	1	1	1	1	1	1
2	1	0	1	1	1	0	0	0	1	0	1	1	1	0	0	1	1	1	0
1	1	0	1	1	1	1	0	1	0	1	0	1	1	1	0	1	1	1	0
0	1	0	1	1	1	1	1	1	1	0	1	0	1	1	0	1	0	1	0
byte3 bit7	1	1	0	1	1	1	1	1	1	1	0	1	0	1	1	0	1	0	1
6	1	1	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	1
5	1	1	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0	1
4	1	0	0	0	1	0	1	1	1	0	1	1	1	0	1	0	0	1	0
3	1	0	1	0	0	1	0	0	1	1	0	1	1	1	1	1	0	0	0
2	1	0	0	1	0	0	1	0	0	1	1	0	1	1	1	1	1	0	0
1	1	0	0	0	1	0	0	1	0	0	1	1	0	1	1	1	1	1	0
0	1	0	1	0	0	1	0	1	1	0	0	1	1	0	0	1	1	1	0
crc bit15	0		0	1	0	0	1	0	1	1	0	0	1	1	0	0	1	1	0
14	0		0	0	1	0	0	1	0	1	1	0	0	1	1	0	0	1	0
13	0		0	0	0	1	0	0	1	0	1	1	0	0	1	1	0	0	0
12	0		0	0	0	0	1	0	0	1	0	1	1	0	0	1	1	0	1
11	0		0	0	0	0	0	1	0	0	1	0	1	1	0	0	1	1	1
10	0		0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0
9	0		0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	0
8	0		0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	1
7	0		0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1
6	0		0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0
5	0		0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
4	0		0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
3	0		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
2	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
1	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8 — CRC Register State Behaviour Example

7.4 Transmit Algorithm

The pseudocode Transmit algorithm is in Annex A.3.

Receive Algorithm

A valid frame starts with the channel active state, and terminates with the channel idle state. Upon reception, valid frames are processed as defined below; invalid frames are discarded. The pseudocode Receive algorithm is in Annex A.4.

8 Network Layer

8.1 Assumptions

This protocol supports a variety of topologies in order that the requirements from many application areas can be met. Within a single channel, the topology can be a bus, a ring, a star, or “free” (see Figure 9). Free topology is defined as a total wire specification with no other rules, and a single termination placed anywhere on the network. Thus, the set of all free topologies includes a ring, a star, a bus and virtually any other combination of these constructs.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

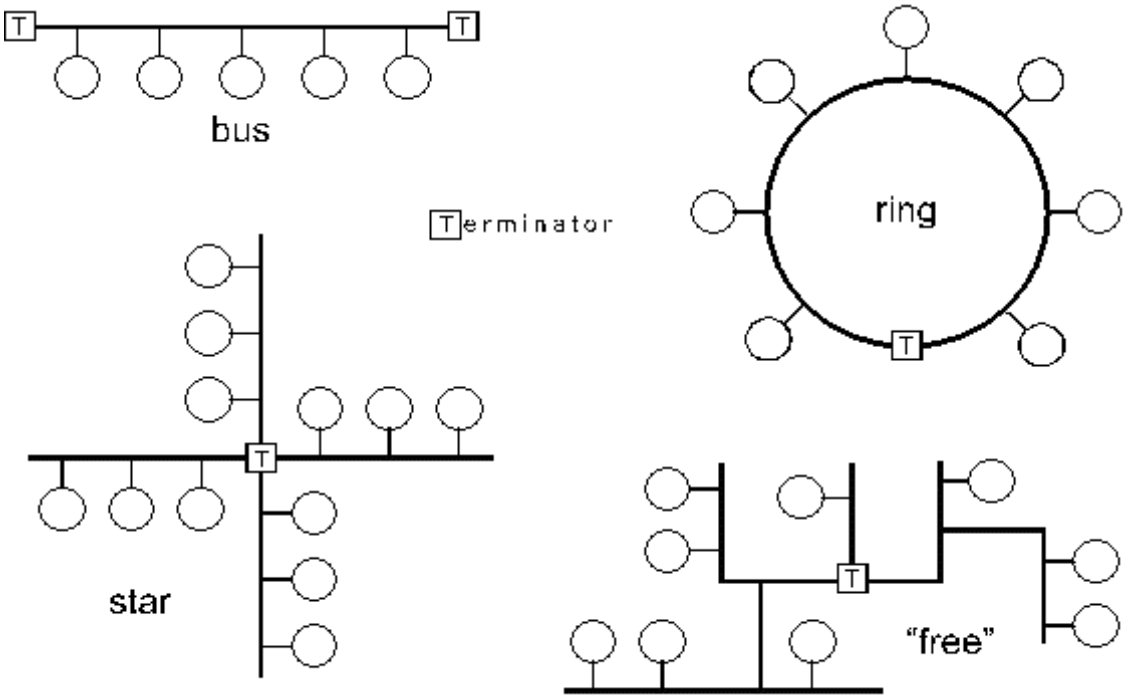

Figure 9 — Single Channel Topologies

The protocol supports physical layer repeaters as well as store-and-forward repeaters to repeat packets from one channel to another. The protocol also supports bridges to repeat all packets on the bridge's domain(s) from one channel to another. Additionally, both learning and configured routers are supported to segment traffic and thus increase total system performance.

In networks where there is a possibility of more than one path from one node to another, there is a danger of packets looping indefinitely. In these networks, configured routers must be used to impose a logical tree structured topology on top of the physical looping topology.

If there is a desire to use repeaters, bridges, and learning routers, then control of the topology must be maintained so that no loops exist. To avoid routing loops within a domain, domain topology must be tree-structured as shown in Figure 10. Store-and-forward repeaters may only be used if they connect two different channels together; store-and-forward repeaters that repeat on the same channel are not supported. This restriction is necessary for an order-preserving network.

If any node receives a packet with a protocol version number other than a version number it supports, the packet is discarded.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

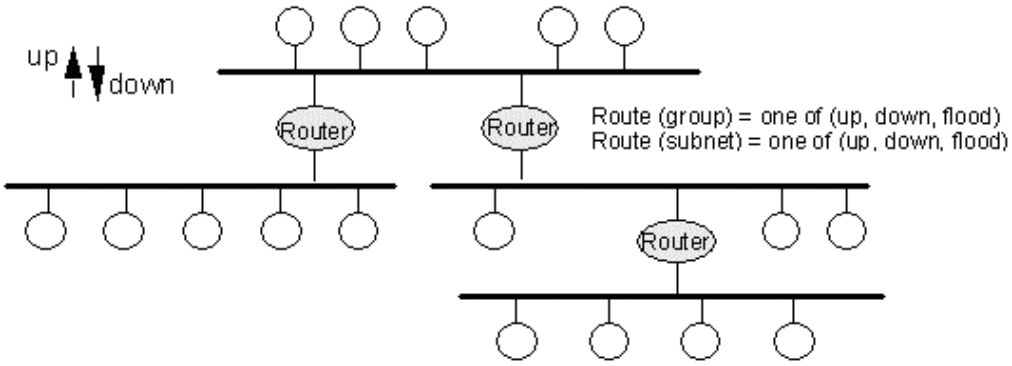

Figure 10 — Typical Tree-Like Domain Topology

8.2 Service Provided

This network layer provides a connection-less network service facilitating domain wide packet delivery with the following attributes:

— *Unacknowledged Unicast, Multicast, and Broadcast.* Depending on its destination address, the packet submitted is delivered to one node, multiple nodes, or all nodes within the domain (or optionally all nodes on a specified subnet within the domain). This delivery occurs with some probability $p \leq 1$;

— *Lossiness.* The network layer supports no re-transmissions or acknowledgments. The probability of delivery decreases with the number of channels traversed;

— *Order Preserving.* Loop-free topology (accomplished logically with configured routers or physically via topological control) coupled with the absence of single channel store and forward repeaters provides natural ordering;

— *No Segmentation.* No message segmentation and/or message re-assembly are performed anywhere within the network layer.

When learning routers are used, the routers discover the topology by examining the network layer address fields in the packets. The learning algorithm imposes no additional traffic overhead on the network. It assumes that the domain is loop free, and it learns about the location of subnets by observing the source addresses of NPDUs being routed. NPDUs addressed to groups are routed by flooding, with the NPDU being propagated through the entire domain.

The network layer suppresses outgoing messages if the node is not in the hard offline state (see 13.4) or configured state (e.g. unconfigured), except for manual service request messages, responses, and ACKs. These excepted messages can be sent in any state.

8.3 Service Interface

The Network service interface consists of the Send_Packet () service request and the Rcv_Packet () indication, as shown in Figure 11. Again this interface is provided for explanation purposes. Actual implementations may, for example, combine this layer with the transport layer and only expose the transport layer interface.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

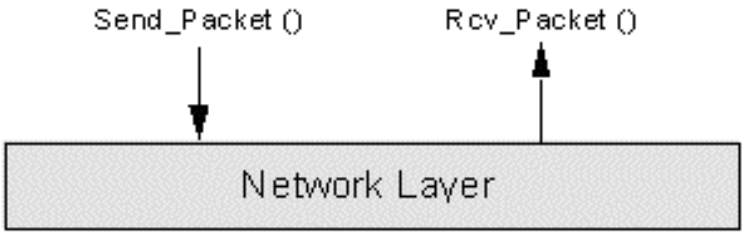

Figure 11 — Network Service Interface

The syntax of these two interface primitives is:

```
Send_Packet (address_pair, pduType, PDU, priority, delta_BL, Alt_path)

Rcv_Packet (address_pair, pduType, PDU, priority)
```

8.4 Internal Structuring of the Network Layer

The network layer performs two functions—address recognition and routing—as shown in Figure 12.

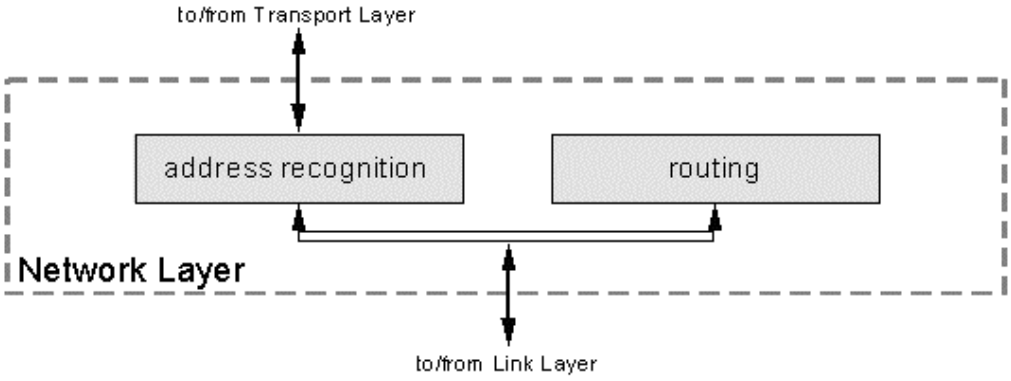

Figure 12 — Network Layer—Internal Structure

8.5 NPDU Format

NPDU format is shown in Figure 13. An NPDU carries and encapsulates either a TPDU, SPDU, AuthPDU or APDU. There are no NPDUs defined for internal network layer use. The numbers above each field in Figure 13 specify the field size in bits. The symbolic field values used in the figure are assigned in the order shown, as enumerated ranges (0, 1, 2, 3, …, n). For example, a value of 3 in the PDU_Fmt field signifies that the enclosed PDU is an APDU. For additional details, including the bit/byte transmission order, refer to Annex D.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

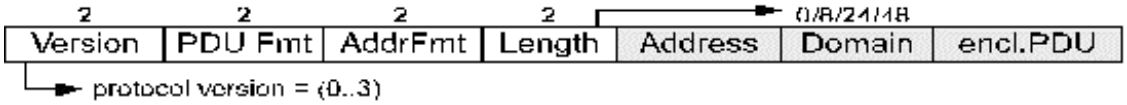

Figure 13 — NPDU Format

8.6 Address Recognition

Each node compliant to this European Standard performs address recognition. Address recognition at a node depends on the node's state. In the Unconfigured state, all broadcast messages and messages that match a node's Unique_Node_ID are received. In the Configured state, messages must either match the Unique_Node_ID or match the domain and be either broadcast, group (i.e., multicast) or unicast messages. To match the domain both the domain length and the domain address must match. For broadcast messages, the subnet must match unless the broadcast message is sent domain-wide. For a group message, the node must be a member of the group, i.e., one of the group addresses must match. For a unicast message, subnet as well as node numbers must match. The match of the node number is a comparison of the entire byte even though the node number is defined as seven bits.

In this subclause the information that must be kept by every node is identified without specifying the address-matching algorithm, that is implementation specific. For an example of an actual implementation of these data structures, refer to Annex A.14.

Any packet received by a node with a protocol version number not supported by the node shall be discarded.

A packet whose source subnet/node address matches that in the matching domain is discarded.

When not in the configured or the hard-offline state, only broadcast (domain-wide or subnet) or matching Unique_Node_ID messages are received. The receive transaction source address is saved based on the actual domain in which the message was received (the flexible domain). Any responses are returned on that domain with a subnet/node number of 0x00/0x80. Either the implementation shall support one flexible domain shared among all receive transactions or the implementation shall support one flexible domain per receive transaction.

8.7 Routers

A router performs the routing function for a specific domain. The routing function is independent of the network layer address recognition function. Thus, a router also has its own network addresses and can be addressed like any other node for purposes of network management and diagnostics. A router connects two sets of subnets. A router is a logical rather than physical entity; more than one router may be housed within a single physical package. Routers have four modes that may be set with network management messages: repeater, bridge, learning, and configured. Routers with their mode set to learning, automatically learn the subnet topology of the network. Routers with their mode set to configured have their routing tables statically configured with network management messages to enforce a specific topology and to provide information on group address topology. Similarly, a router may be set to be a bridge or repeater using the network management messages to set its mode. The same subnets may exist on both sides of a bridge or repeater.

A router uses three routing functions: $ROUTE_{uc}()$, $ROUTE_{mc}()$, and $ROUTE_{bc}()$ to forward NPDUs. The first function specifies how to forward NPDUs addressed to subnets, the second how to forward NPDUs addressed to groups, and the third how to forward NPDUs when they are broadcast. The functions are tables, where each entry has the following form:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
ROUTEuc (DestSubnet) = one of (forward, discard)
ROUTEmc(DestGroup) = one of (forward, discard)
ROUTEbc (DestSubnet) = one of (forward, discard)
```

The entry for an address X specifies whether an NPDU addressed to X should be forwarded or discarded. These functions are called from the side of the router on which the packet was received—that is, the algorithm in Annex A.5 executes independently on each side of the router and makes all routing decisions for packets arriving at the side on which it runs. Each side of the router shall have a different set of tables that have the information as to whether the packet should be passed across the router or not. For configured routers, these tables are initialised during the network configuration of the router and must be updated if the topology changes. Learning routers discover the contents of their subnet tables by observing the source addresses of the packets (see Annex A.6), but cannot learn the topology of a group. Therefore, learning routers always pass group messages to the other side.

8.8 Routing Algorithm

The pseudocode for the routing algorithm is in Annex A.5.

8.9 Learning Algorithm — Subnets

The subnet routing table defining the routing function $ROUTE_{uc}()$ is created by the algorithm in Annex A.6. Upon initialisation, forwarding is used for all subnet addresses. The algorithm subsequently learns about the location of subnets by observing the source addresses of NPDUs being routed. Again, this algorithm executes on each side of the router independently.

9 Transaction Control Sublayer

9.1 Assumptions

The transaction sequencing protocol described in this clause uses 4-bit transaction numbers that are allocated by the sender and used by the receiver to detect duplicate packets. This sublayer provides end to end protocol integrity by allowing the receiver to only act upon a packet once, even if several retries for the initial packet are correctly received by the receiver. It is assumed that the network is either order preserving or, if it is not, that packets are delayed approximately uniformly in the multiple paths that they traverse from source to destination. The difference in packet propagation time when there are multiple paths from a given source to a destination must be less than the time it takes for the source to complete one transaction (via an acknowledgement on the shortest path) and begin a second transaction to the same destination. Stale acknowledgements and responses are detected as duplicates, but stale packets that initiate a transaction and arrive after a subsequent transaction has started will not be detected as duplicates.

The transaction control sublayer may pass a transaction to the higher layers for processing. In some implementations, it may be possible for the upper layers to never respond to the transaction control sublayer. If such a situation exists, the transaction control sublayer can defend itself from such poorly behaved upper layers. During the time that the transaction control sublayer is waiting for a response from the upper layers, the transaction record can be locked and not be de-allocated even if the receive transaction timer expires. Such a transaction is only de-allocated when the upper layers finally respond, or when there are no more transaction records available and one is needed. Then, a locked transaction record may be used for the new transaction. Use of this mechanism is implementation dependent.

In the general case, the number of concurrent outgoing transactions is restricted to a single priority and a single non-priority transaction. This restriction is due to the fact that acknowledgements/responses from either a subnet/node addressed message/request or a Unique Node ID addressed message/request are indistinguishable. Implementations that restrict the use of one of these addressing modes may support multiple, concurrent transactions. Receive transactions are shared among the transport and session layers. Transactions at either layer can perform duplicate detection.

The transaction timer, receive timer, and retry count interact to establish the reliability of the duplicate detection mechanism. It is assumed that the receive timer is set to be long enough to cover the configured number of retries,

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

yet short enough so that the transaction ID does not wrap around causing a new transaction to be falsely detected as a duplicate transaction. In an implementation, it is required to have a transaction space for all priority transactions and a second transaction space for all non-priority transactions. If the sender wishes to send a packet to a given destination and the transaction ID for the last message sent to that destination is identical to the current transaction ID, the transaction ID is incremented. An optional enhancement is to have a separate priority and non-priority transaction space for every destination address. In this second case, the sender remembers the last transaction ID used per destination address. In any case an implementation shall defend against the situation that acknowledgements/responses cannot be differentiated from each other when using subnet/node addressing or Unique Node ID addressing. Other implementations are permitted as long as the duplicate detection mechanism remains robust in all of the error cases and all of the addressing formats in use by the node.

9.2 Service Provided

The transaction control sublayer is responsible for the common functions related to transaction ordering, sequencing, and duplicate detection. It provides the following services:

— Outgoing Sequencing. To guarantee ordering among outgoing transport messages and session layer requests, the transaction control sublayer controls the allocation of send transaction numbers. It limits the number of concurrent transactions to any destination to ≤ 1 priority and ≤ 1 non-priority transactions;

— Incoming Sequencing and Duplicate Detection. The transaction control sublayer provides duplicate detection.

9.3 Service Interface

Access to transaction control services is facilitated by the interface depicted in Figure 14.

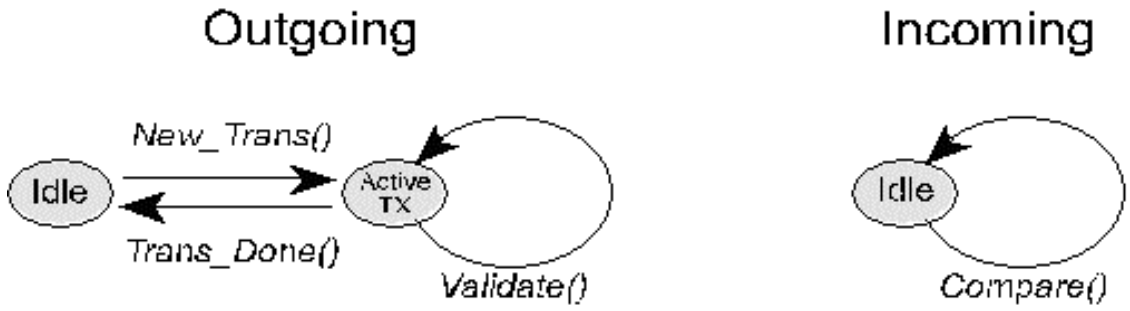

Figure 14 — Transaction Control Service Interface

The syntax and semantics of the interface primitives are:

```
New_Trans (Priority, Dest_addr) -> (Trans_No)
    is used to obtain a transaction number for a new outgoing transaction

Validate (Priority, Trans_No, Dest_addr) -> (result)
    where result = one of (current, not_current), verifies that Trans_No is in the transmit
    window

Trans_Done (Priority, Trans_No, Dest_addr)
    notification of an outgoing transaction completion

Compare (T1, T2) -> (result)
   where result is one of (new, duplicate ); defines the relationship of T1 relative to T2,
    where both T1 and T2 are receive transaction numbers
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4 State Variables

To support the allocation of transaction numbers, the transaction control sublayer uses a number of destination records shown below.

```
Transaction_CTRL_Record = record
    PriTX               True or False
    Trans_No:           (0..15); initial value = 0;
    In_Progress:        boolean;
    end;
```

9.5 Transaction Control Algorithm

The pseudocode algorithm in Annex A.7 is a version of the transaction control algorithm that provides only two transaction spaces – one for all priority transactions and one for all non-priority transactions. In any implementation, the first transaction ID provided by New_Trans() after a reset shall be 0 for every transaction space provided by the implementation. New_Trans() shall then increment the transaction ID within the space from 1 to 15 and continue again with 1 so that the transaction ID 0 is used exclusively by the first transaction per transaction space after a reset.

In general, any transaction control algorithm shall follow the following basic operation:

- When a node has a packet to send using the transaction control algorithm, the sender node picks a transaction ID and sends the packet.
- Upon receipt of that packet, the receiver node searches its currently active receive transactions for a transaction that matches the source and destination address of the packet, the priority attribute, and the transaction ID.
- If there is no match a new record is allocated.
- If everything but the transaction ID matches and the existing transaction is not locked, then the preceding transaction is assumed to have completed successfully and the receive transaction record may be reused with the new transaction ID inserted into it. Processing of the incoming packet cannot proceed until a receive transaction is allocated for it.
- Once the receiver has allocated the transaction record, it starts a receive transaction timer for that record. The value of the receive transaction timer is 8 seconds for a Unique_ID addressed message, the value of the non-group receive timer for a subnet/node addressed message, and, for group addressed messages, the receive transaction timer configured for the group. The record will be de-allocated upon the expiration of this timer unless the transaction record becomes locked (see 9.1).
- Subsequent packets that arrive at the receiver node that match the transaction ID, source address, destination address and priority attribute of an existing transaction will be considered duplicates by the receiver node.
- Packets detected as duplicates are responded to according to the protocol service requested, but are either not delivered to the application, or if they are delivered to the application, they are delivered along with the notification that they are duplicate packets.

10 Transport Layer

10.1 Assumptions

The Transport protocol makes no assumptions apart from relying on the transaction control sublayer for correct TPDU sequencing and duplicate detection.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.2 Service Provided

The transport layer provides the following services:

- Reliable Multicast and Unicast. The transport protocol supports both multicast within a group, and multicast to a group with the sender not being a member of the group. If the service requires acknowledgements or responses, then multicast to a group that the sender is not a member of requires the transport layer to be told that the group size is the actual group size plus one. All reliable services have the following attributes:

- (a) reliable delivery with best effort determined by the number of retries;

- (b) assuming the layer 4 timers are set correctly, duplicate detection is provided in all cases except when the sender or receiver just reset. In this case, the reset node shall start with transaction number 0. This may result in a transaction in progress between the two nodes not to be acted upon at all, but acknowledged/responded to by the receiver;

- (c) partial ordering—ordering is preserved but a message is not delivered when delivery fails within the specified number of retries; and,

- (d) immediate re-synchronisation—following a network partitioning, the very first message is delivered.

- Unacknowledged-Repeated Multicast and Unicast. These services differ from the reliable ones described above only in that no acknowledgement is expected, and that the message is sent repeatedly until the number of repetitions is equal to the retry count. When using this service, the limit of 64 members in a group does not apply—the only limit on the number of members in a group addressed via this service is the number of nodes in a domain.

Groups in this European Standard are symmetric in that every member of the group can both send and receive.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.3 Service Interface

The service interface provided to the session and application layers has the form shown in Figure 15.

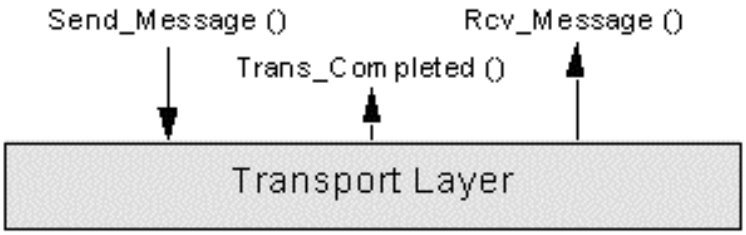

Figure 15 — Transport Interface to Upper Layers

The syntax and semantics of the transport layer interface are:

```
Send_Message (Address, APDU, priority, ServiceType, alt_path) -> (TID)

Trans_Completed (TID, Result)

Rcv_Message (APDU)
```

TID, above, is a unique identifier for the transaction.

10.4 TPDU Types and Formats

TPDU syntax is shown in Figure 16; the number above each field specifies the field size in bits. The symbolic field values shown in the picture are mapped onto numeric ranges (0, 1, 2, 3…) in the order shown. Additional details, such as the bit/byte transmission order, are defined in Annex D. Timers and counters used to transmit retries, count retries, and time out transaction IDs, are described in Annex B.4.11.

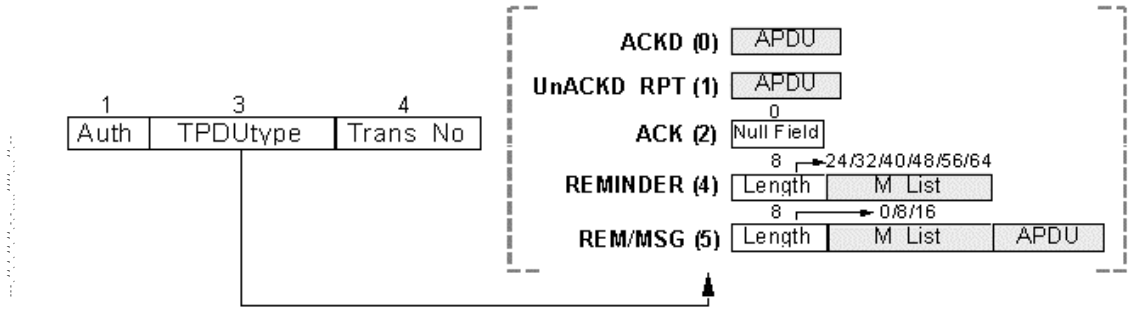

Figure 16 — TPDU Types and Formats

The *Acknowledged Message* (ACKD) TPDU (see Figure 34) is used for the first transmission of a message. It is used with addressing formats #1, #2a, and to a limited extent #3. Unlike the Unacknowledged message TPDU, it must be acknowledged by all addressed recipients, thus the transport layer provides the number of

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

acknowledgements expected to the link layer for encoding into the link layer header. This TPDU is used for acknowledged initial TPDUs (both unicast and multicast) as well as unicast reminders. Multicast reminders are covered below. Refer to Figure 36 or Figure 34 to review the addressing formats.

The *Unacknowledged-repeated Message* (UnACKD_RPT) TPDU is identical to its acknowledged counterpart with one exception: on its reception, no acknowledgements are returned to the sender. This TPDU is used with no modifications for the unacknowledged-repeated service. For this type of message the transport layer provides a backlog increment equal to the number of retries on the first message and a backlog increment of zero on all subsequent transmissions within the transaction. Simple unacknowledged messages are only sent once, have no TPDU header, and thus have no duplicate detection.

The *Message-Reminder* (REM/MSG) TPDU facilitates selective soliciting of acknowledgements for multicast transactions. REM/MSG *type 5* is used when the highest numbered group member from which the sender has received an acknowledgement is < 16; this TPDU contains both the member list (M_List []), that is an array of bits, and the APDU. The Length field specifies (in bytes) the size of the M_List field. A value of 0 in the bit field M_List [X] indicates that member X's acknowledgement has not been received by the sender, whereas a value of 1 indicates that the acknowledgement has been received. Bit locations within a byte are assigned right to left with bit number zero on the right. When Length = 0, the M_List[] field is absent and the meaning is "all members should acknowledge." The backlog increment for this message is the number of acknowledgements that have not yet been received.

Type 4 is a plain *reminder*, without an APDU (see Figure 16). It is used in cases where the highest numbered member that has acknowledged the message is ≥ 16. Acknowledgements are solicited using the TPDU pair (REMINDER, ACKD) and this pair is logically equivalent to a single type 5 REM/MSG TPDU used when the members needing to acknowledge may be encoded within the type 5 format. (This separate solution is provided for large groups because of the need to limit maximum TPDU size.) This message always has a backlog increment of zero. The ACKD message that immediately follows its transmission has its backlog increment set to the number of acknowledgements that are yet to be received.

The *Acknowledgement* (ACK) TPDU is null. It uses addressing format #2a (unicast acknowledgement) or #2b (group acknowledgement). The Trans_No field conveys the transaction being acknowledged. Acknowledgements always inherit the priority and alt_path attributes of the original message.

Any TPDU that requires an acknowledgement can be flagged as an authenticated packet by setting the Auth bit to '1'.

Codes 3, 6, and 7 are reserved. Packets with these codes are discarded.

10.5 Protocol Diagram

The diagram in Figure 17 augments—but doesn't replace—the protocol description in 10.7-10.8.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

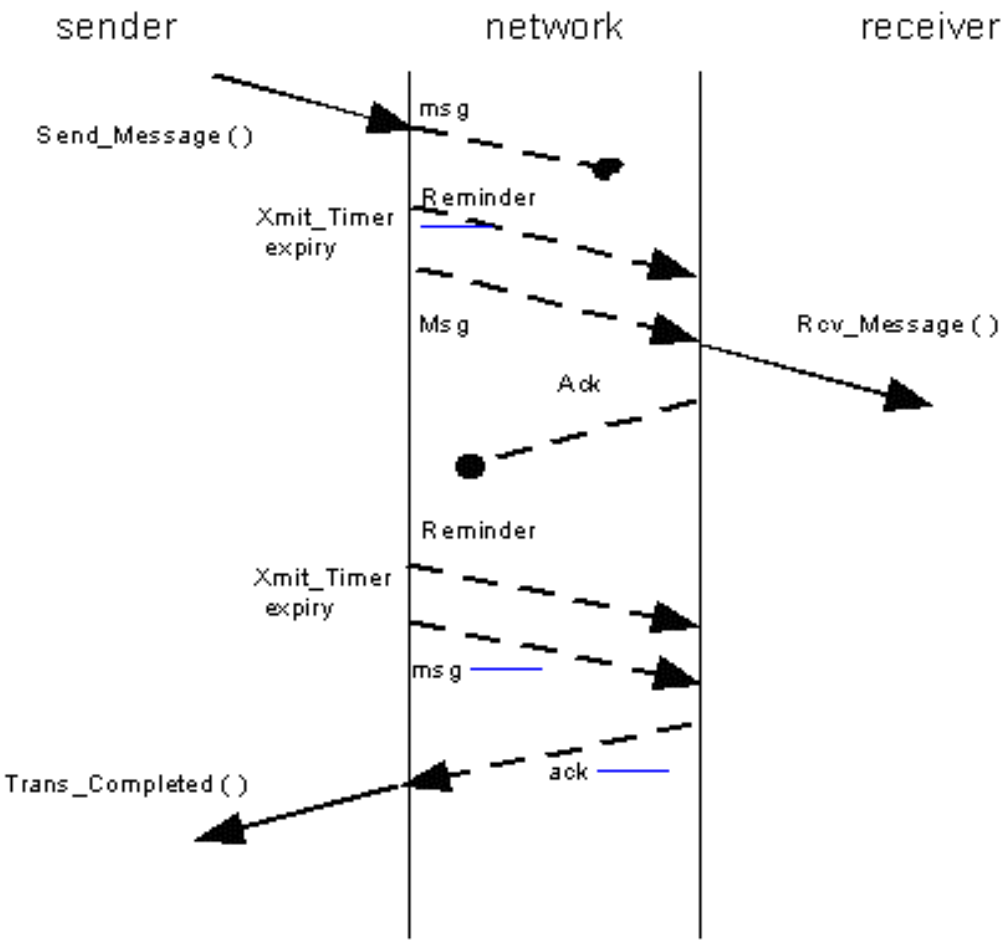

Figure 17 — Transport Protocol Diagram for Multicast Message with a Loss of Both the Message and the ACK TPDUs

10.6 Transport Protocol State Variables

The transport protocol consists of two independent functions—Send and Receive. To support the send function, the transport layer keeps one Transmit record per transaction in progress. A shared pool of Receive records facilitates message reception. The Transmit and Receive data structures are described in Annex A.8 and A.9.

10.7 Send Algorithm

The simplified transmit FSM is shown in Figure 18, with full details in the algorithm in Annex A.9. The algorithm resets the re-transmission timer, Xmit_Timer, whenever an acknowledgement or challenge is received, as opposed to once per Expiration period. The alternate path bit is set in the packet on the last two attempts by this layer notifying the lower layers of the alternate path attribute for each attempt.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

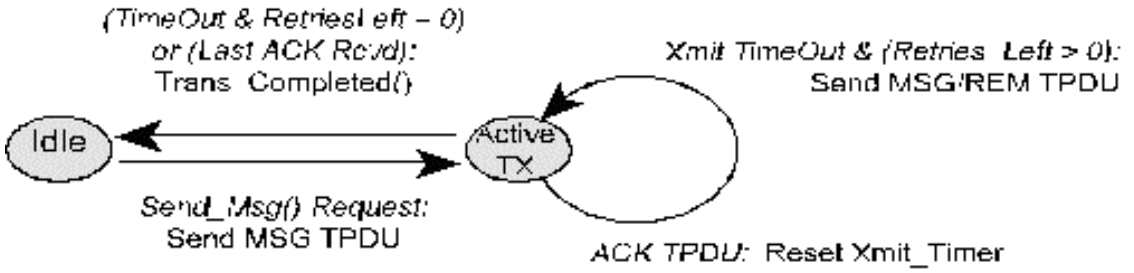

Figure 18 — Transport Protocol—Send FSM

10.8 Receive Algorithm

Message reception uses the timer-based mechanism. Figure 19 shows the receive FSM for a single transaction, while the algorithm in A.9 specifies full details.

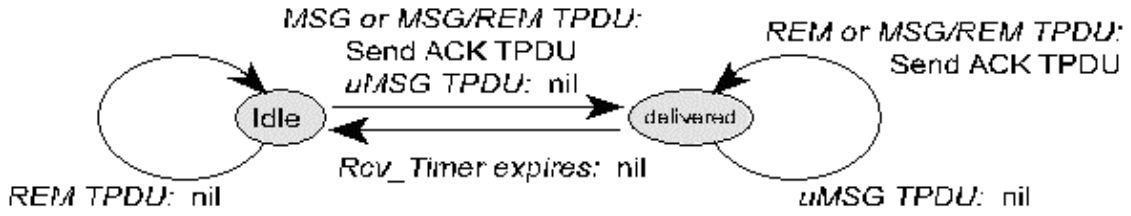

Figure 19 — Transport Protocol—Receive FSM

10.9 Receive Transaction Record Pool Size and Configuration Engineering

10.9.1 General

Space for the Receive Transaction Record (RTR) pool defined in 9.5 and Annex A.9, may be fixed for an implementation, or may be allocated at the time the application program is linked with the protocol code. The size of the RTR pool limits the number of *concurrent receive* operations on a node ("concurrent" means concurrent within the context of Rcv_Timer) and should be engineered according to the number of concurrent transactions expected within the receive timer interval. If an attempt is made to allocate a receive transaction record and none are available, the receive transaction full statistic is incremented (see 13.8.2).

10.9.2 Number of Retries

The number of retries should be large enough to ensure that message delivery is successfully completed with acceptable probability, e.g., ≥ 99 %. However, the upper bound of the number of retries in this European Standard is 15.

If the delivery failure probability of a single attempt is p, then the probability that message delivery within a group of n succeeds in ≤ k attempts is shown in the following formulae. (It is assumed that the single attempt probability p is the same for all destinations.)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$P\{\text{no retry}\} = (1-p)^{n}$$

$$P\{\leq k \text{ retries}\} = \sum_{i=0}^{k} \binom{k+1}{i} p^{i} (1-p)^{(k-i+1)} \left(1-p^{(k-i+1)}\right)^{(n-1)}$$

$$k \geq 0,\ n \geq 2 \qquad (13)$$

NOTE For a single channel $p = (1/2w + p_e)$, where w is the size of the MAC layer randomising window and p_e is the probability of packet loss because of a transmission error.

The above probabilities are graphed in Figure 20. Given a group size and the error probability p, the required number of retries for, say 99.5 % probability of success, can be read off the graph.

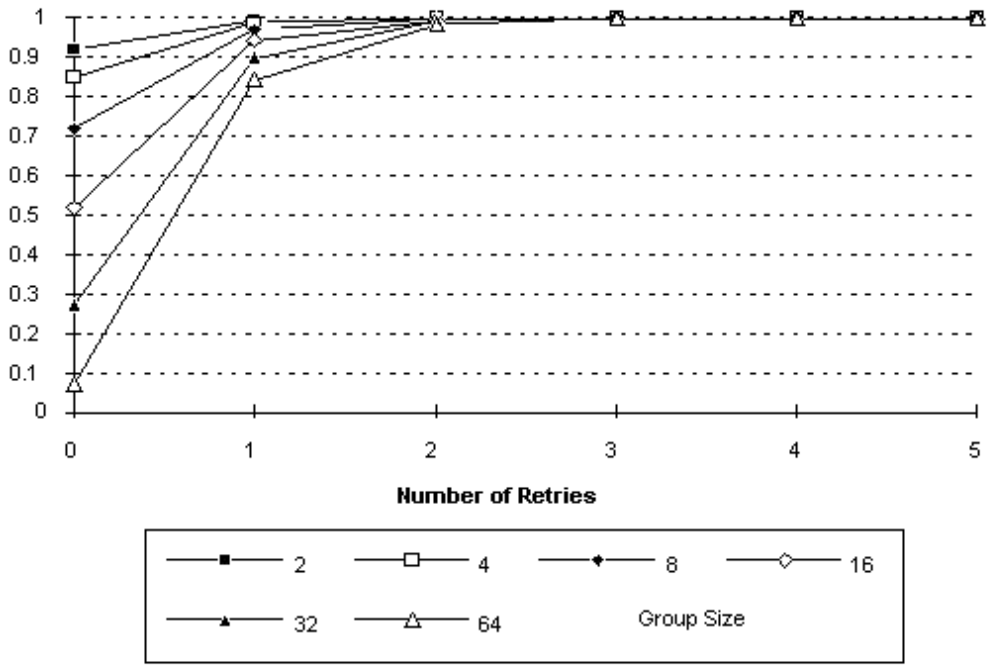

Figure 20 — Probability of Transaction Completion in k Retries

Pragmatic considerations should be respected in any real system design. For example, there is no need for a large number of retries with transactions that are repeated periodically. Additionally, when the acknowledgements to a multicast message would take up a significant percentage of the bandwidth of the channel, it is more likely that a message will be properly delivered using the “unacknowledged-repeated” service rather than the acknowledged multicast service. In practice, it is expected that a Retry_Count in the (2..5) range will cover most situations.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.9.3 Transport Layer Timers

There are three timers used by the transport layer protocol. They are the:

— Xmit_Timer the layer 4 retransmission timer

— Repeat_Interval_Timer the UNACKD_RPT interval timer

— Rcv_Timer the receive record timer (see Annex A.9 and 10.9)

These timers are active for every transaction. On the sender's side, either the Repeat_Interval_Timer or the Xmit_Timer is active, while on the receive side, the Rcv_Timer is active. Xmit_Timer is reset on every ACK TPDU or Authentication challenge reception, while Rcv_Timer is reset whenever a MSG or MSG/REM TPDU that has a new transaction number is received for this destination. The Repeat_Interval_Timer determines the interval between the UNACKD_RPT packets sent by the sender. The recommended methodology for calculating the timer values is shown in Figure 21. These recommendations are for unicast transactions across a single channel. For multi-channel networks, the calculation depends upon the speed of the router and the number of buffers. Multicast transactions need somewhat longer receive transaction values since multicast transactions take longer to complete even when there are no retries needed. Since buffering and clock rates are adjustable, the system designer must make some measurements to set the timers and retry counts correctly within each node.

Retry Count = 2-5	(see 10.9.2 and Figure 20)
Xmit_Timer ≥ 3 * packet cycle time + margin	(margin = best case tx completion time)
Rcv_Timer ≥ Xmit_Timer * (Retry_Count + 2)	
where:	
"3 * packet cycle time" assumes that the average station on the channel must wait two packet cycles of delay to access the network. Then, following network access, the transmission time of the average packet (also the packet cycle time) is added to the timer value. Finally, the time it takes the receiver to process the packet and send the acknowledgement is added. This is a function of the clock speeds of the associated nodes.	

Figure 21 — Methodology for Calculating Timer Values

All messages using the Unique_Node_ID use an 8-second receive timer, regardless of the configured value of the non-group receive timer. All non-group messages use the non-group receive timer.

11 Session Layer

11.1 Assumptions

The session layer makes no assumptions apart from relying on the transaction control sublayer for correct SPDU sequencing and duplicate detection.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

11.2 Service Provided

The session layer provides a single service:

— Request-Response. This service facilitates application communication similar to a remote procedure call. In particular, it allows a client to make a request to a remote server and receive a response to this request.

— *Non-idempotent* transactions are to be executed *"at most once"* (i.e., exactly once or not at all). Non-idempotent transactions are those where the action depends on a prior state, such as "open the valve an additional 10 %." The protocol considers a transaction to be non-idempotent if and only if its response length is ≤ 1 byte.

— Requests with response length > 1 byte are considered *idempotent*; such requests may be executed *"several times"* (i.e., zero or more times). Idempotent transactions are those where the action may be repeated any number of times, and the effect is the same. An example of an idempotent command is "read the first 10 table values".

— The distinction between idempotent and non-idempotent transactions is based upon the size of the response, in order to limit the amount of storage required for transaction records within a node. If a transaction has a response length > 1 byte but may not be executed more than once*, it is the responsibility of the application to save the response and send it again*. The session layer facilitates this in that notification that the request is a duplicate is provided to the application layer by the session layer.

— A request-response transaction fails unless the server response is generated within the limit imposed by the Request-Response timers and re-transmissions (see 10.8).

11.3 Service Interface

The service interface to the application layer has the form shown in Figure 22.

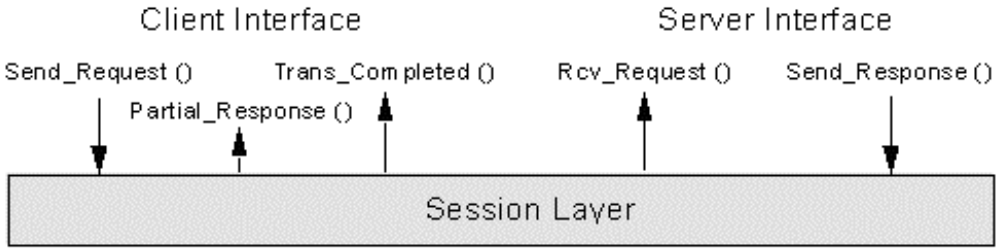

Figure 22 — Session Layer Interface to Application Layer

The syntax of the service interface primitives is given below. TID is a unique identifier for the transaction. See Annex A.7 for implementation details. Duplicate is a Boolean that, when true, indicates that the client is retrying a previously executed request. Result is a Boolean; true and false denotes success and failure respectively.

Send_Request (Address, APDU, priority) -> (TID) {client}

Partial_Response (TID, APDU, priority)

Trans_Completed (TID, Result)

Rcv_Request (TID, APDU, duplicate, priority) {server}

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Send_Response (TID, APDU, priority)

11.4 Internal Structure of the Session Layer

The session layer is internally structured as shown in Figure 23.

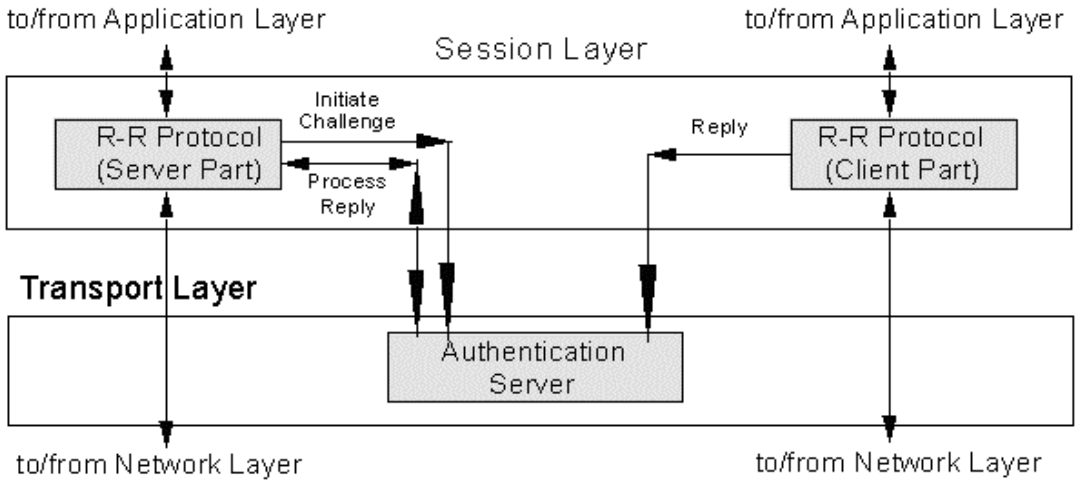

Figure 23 — Session Layer—Internal Structuring

The Request-Response protocol accesses the Authentication server via the following three calls (details in 11.11). The transport layer also accesses the Authentication server with the same three calls.

Initiate_Challenge(RR,PDU) -> (null) { challenger }

Reply(XR, PDU)-> (null) { challengee }

Process_Request(RR,PDU) -> (pass/fail); { challenger }

11.5 SPDU Types and Formats

SPDU formats are shown in Figure 24, where the number above each field specifies the field width in bits. The symbolic values shown in the picture are mapped onto numeric ranges (0, 1, 2, 3, ...) in the order shown.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

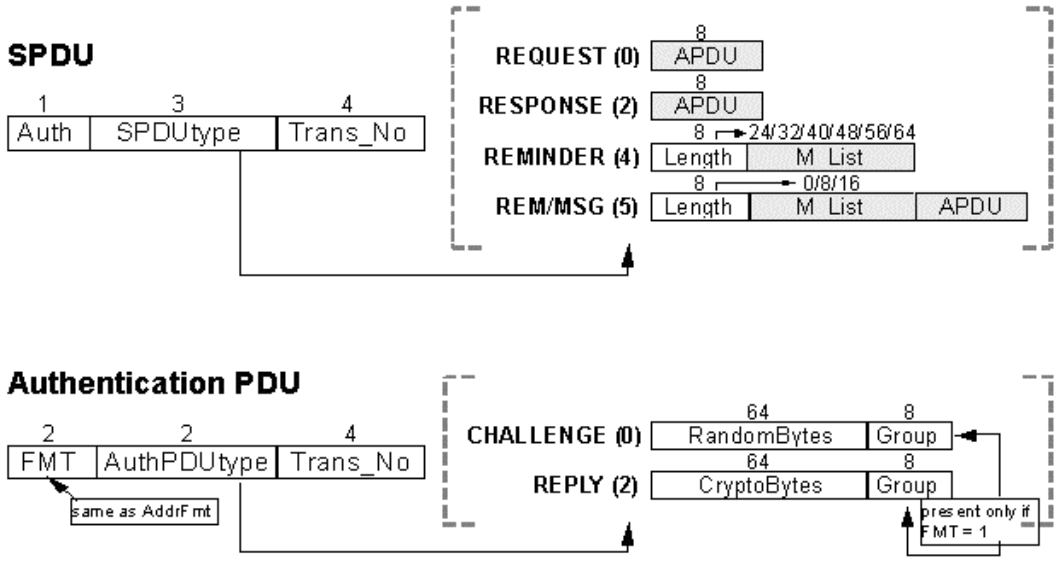

Figure 24 — SPDU Types and Formats

The Request-Response protocol uses three basic PDU types: Request (REQUEST), Response (RESPONSE), and combined Request-Reminder (REM/MSG). The syntax (packet layout) and semantics (packet processing) of these three basic SPDU types correspond closely to that of ACKD, ACK, and REM/MSG TPDUs. Note that although the session layer makes use of the transport layer features internally, no transport layer encoding exists separately in a packet with an SPDU. That is, a packet may have either a TPDU or an SPDU.

The *Request* (REQUEST) SPDU is used with the first transmission of the request. It employs addressing formats #1, #2a, and to a limited extent #3 and #0 (see Figure 34). Address format #0 is used by a special network management command to broadcast a request searching for any nodes that have not been configured. Address format #3 is then used to configure those nodes that respond to this special network management command. The session layer provides a backlog increment to the lower layers equal to the number of responses expected.

The *Request-Reminder* (REM/MSG) SPDU facilitates selective soliciting of responses. REM/MSG *type 5* is used in groups where the highest member number needing to acknowledge is < 16; this SPDU contains both the member list (M_List []), that is an array of bits, and the APDU (i.e., the request itself). The Length field specifies (in bytes) the size of the M_List field. A value of 0 in M_List [X] indicates that member X's response has not been received by the requester, whereas a value of 1indicates that the response has been received. Bit locations are assigned right to left within each byte with bit 0 being the rightmost bit in the byte. The backlog increment encoded with the REM/MSG is equal to the number of remaining responses outstanding.

Type 4 is a plain reminder, without a request (see Figure 24). It is used where the highest member number needing to acknowledge the reminder is ≥ 16; in this case, responses are solicited using the SPDU pair (REMINDER-type 4, REQUEST-type 0) and this pair is logically equivalent to a single type 4 REM/MSG SPDU used in small groups. (A separate solution is provided for large groups because of the need to limit maximum SPDU size.) Finally, when Length = 0 the M_List[] field is absent and the meaning is "all members should acknowledge." The backlog increment for the REMAINDER – type 4 message is always zero, while the REQUEST that immediately follows it contains the backlog increment for the number of responses that are still outstanding.

The *Response* (RESPONSE) SPDU uses addressing format #2a (unicast acknowledgement) or #2b (group acknowledgement). The Trans_No field conveys the transaction being acknowledged. The length of the APDU implicitly defines the type of transaction: if the response can be stored in a single byte, the transaction is treated as non-idempotent. Otherwise the transaction is treated as idempotent. Responses inherit the priority and alt_path attributes of the Request packet.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Authenticated SPDUs (Auth bit set to '1') identify requests that are to be authenticated by the recipient. In all other respects, they are identical to the SPDUs that are not authenticated.

Authentication. The authentication server is a single server contained in Layer 4 and available to the transport and session layer protocols. It provides a one-way authentication service. It is the client's responsibility to initiate authenticated transactions when required. Setting the Auth bit in the SPDU or TPDU does this. When a TPDU or SPDU is received with the Auth bit set, the server shall challenge using the "challenge" AuthPDU. The FMT field in the authentication PDU shall be set to the same value as in the address format field of the NPDU header in the message being challenged. The client then computes a transformation based upon the server's challenge, the original APDU sent by the client, and the client's authentication key. The result of this transformation is sent to the server using the "reply" AuthPDU. The FMT field in the reply shall be set to the same value as the FMT field in the corresponding challenge AuthPDU. When the server receives the reply, its contents are compared to the transformation computed by the server. If they match, the transaction is authenticated. In all cases the SPDU/TPDU is passed to the application layer for processing along with notification as to whether the authentication failed or succeeded. Note that if the application layer on the server node has no requirement for authentication of the particular transaction, it may choose to honour the request even if the authentication failed. If the application layer chooses not to honour the request, it simply discards the APDU without further processing.

11.6 Protocol Timing Diagrams

The protocol timing diagrams in Figure 25 and Figure 26 are intended to provide an intuitive feeling for the session layer protocols and to augment (but not to replace) the protocol specification in 11.7 - 11.13. Note again that for needed acknowledgements from group member numbers < 16, the REM/MSG SPDU is used and is functionally equivalent to the (REMINDER, REQUEST) pair.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

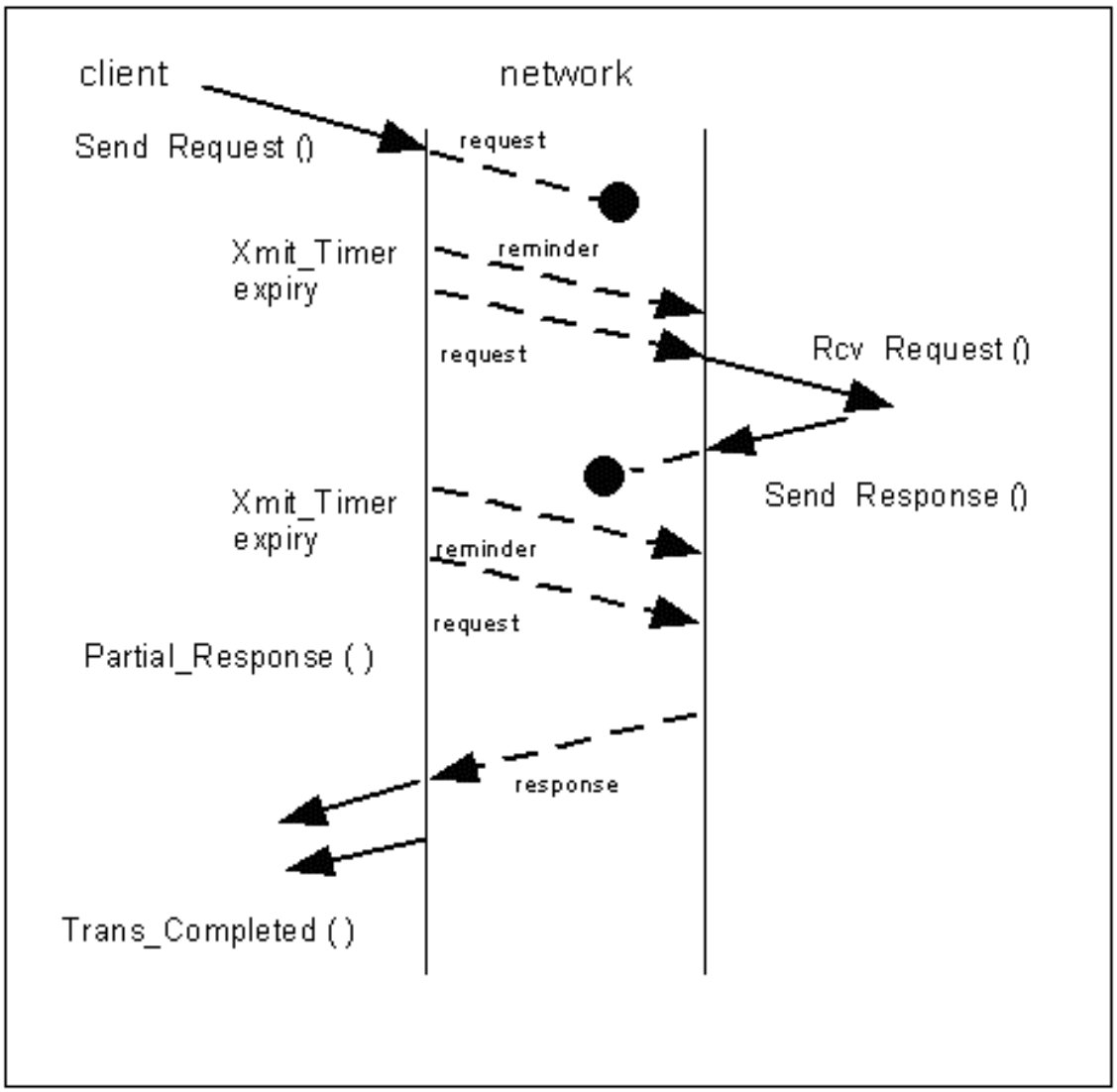

Figure 25 — Non-Idempotent Request with Multiple SPDU Losses

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

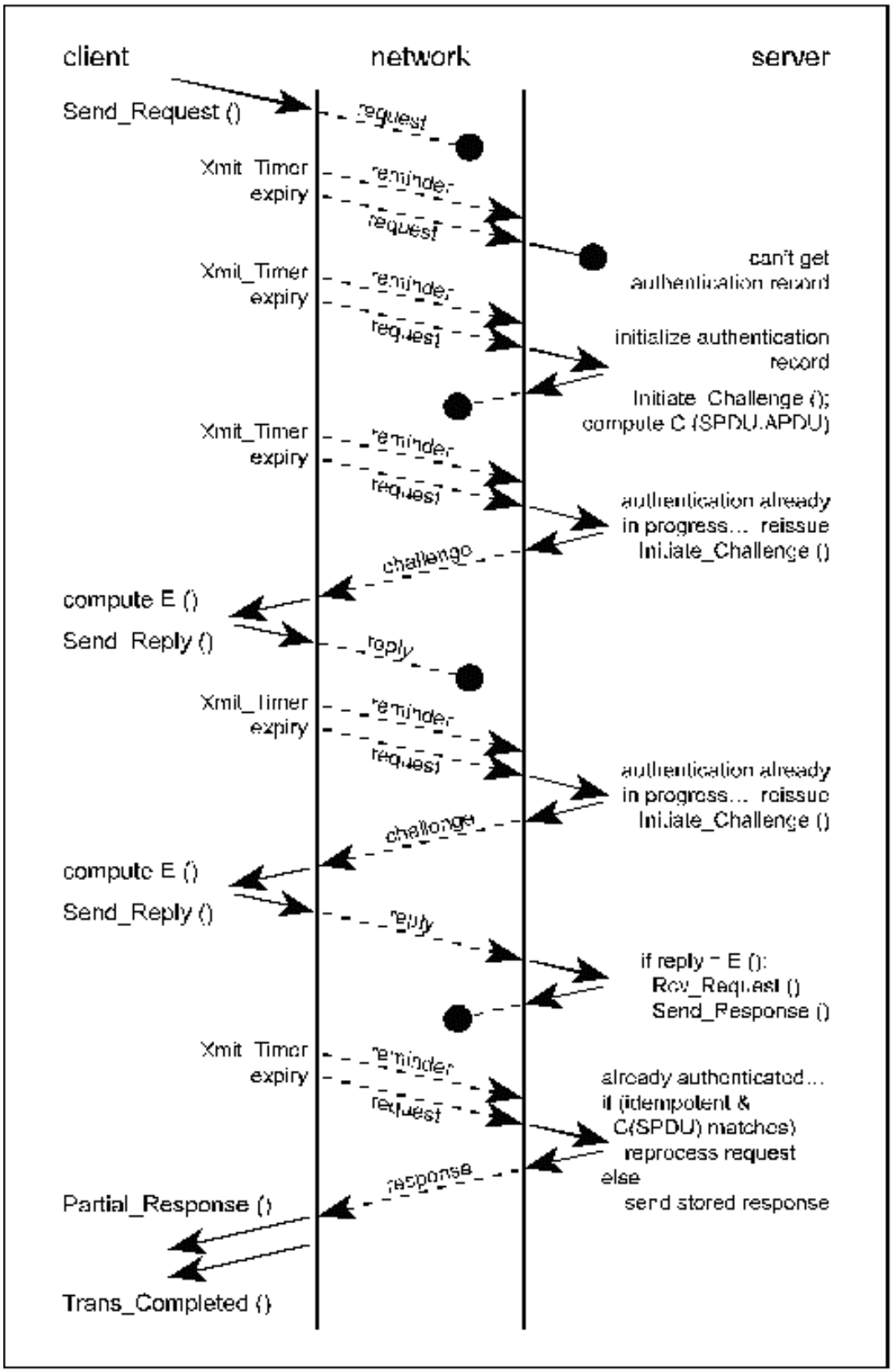

Figure 26 — Secure Idempotent Request with Multiple SPDU Losses

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.7 Request-Response State Variables

Like the Transport protocol, the Request-Response protocol maintains one Transmit record per transaction in progress, and a shared pool of Receive records facilitates message reception. As shown in Annex A.9, these records differ in minor details from those used by the Transport protocol.

11.8 Request-Response Protocol — Client Part

A simplified FSM for the client part of the Request-Response protocol is shown in Figure 27. The detailed specification is in Annex A.9.

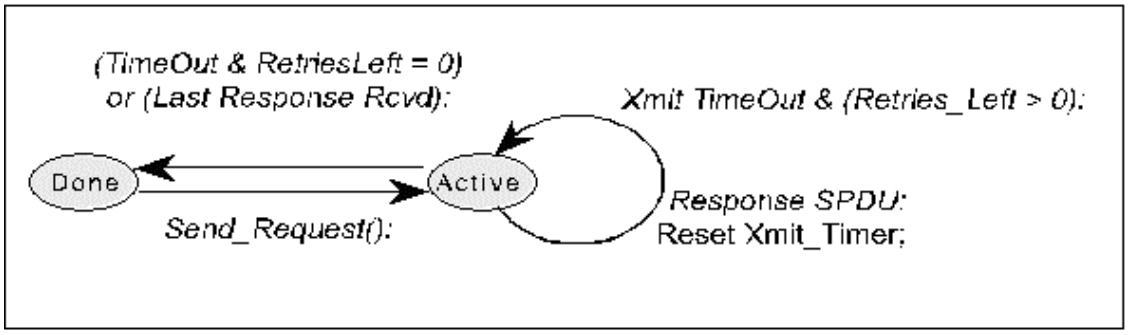

Figure 27 — Request-Response Protocol—Client FSM

11.9 Request-Response Protocol — Server Part

A simplified FSM for the server part of the Request-Response protocol is shown in Figure 28, with the full description in Annex A.9. The protocol treats all transactions with response size > 1 byte as idempotent, implying that it may execute them more than once. Transactions with response size of only 1 byte are never executed more than once.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

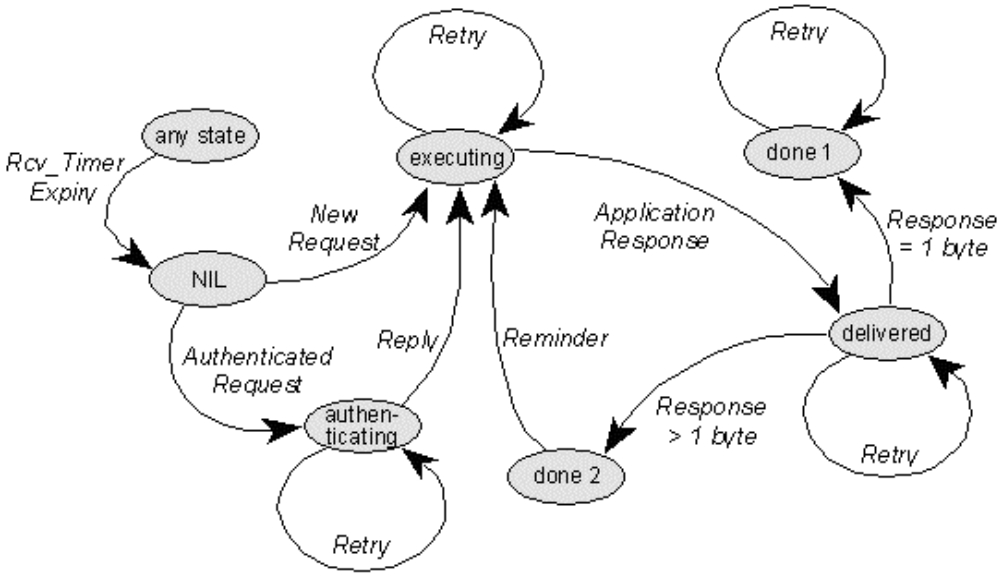

Figure 28 — Request-Response Protocol—Simplified Server FSM

11.10 Request-Response Protocol Timers

The two timers - Xmit_Timer and Rcv_timer - used by the Request-Response protocol follow the function and the form of Transport timers. The recommended values are identical to those for the transport layer as defined in 10.9.3, with the exception that if the request takes a significant amount of processing time on the server (relative to the transaction time), that time should be included in the calculations.

11.11 Authentication Protocol

The Authentication server is implemented in the transport layer. It is accessible by the session layer as well as the transport layer. It is discussed in this subclause to include explanations of the additional interactions needed for the session layer. It relies on the duplicate detection mechanism in the transaction control sublayer, and no other transport layer services. Authentication allows a server to verify the identity of the requester. Use of this service is generally controlled by network management commands that specify the messages/network variable exchanges to be authenticated. Only transactions using acknowledged or request/response services may be authenticated. Messages received on the flex domain are not authenticated. The network management authentication bit is honoured only when the node is in the configured or hard offline states. A checksum error over the node's network configuration memory shall cause the node to go to the unconfigured state.

The Authentication protocol has two asymmetric parts: the challenger and the challengee. The authentication process is initiated by the challenger by generating a random number X; next, the challengee responds with Y = E(X, msg), an encryption of X and the original message using a private key; and finally the challenger compares Y with its own version of E(X, msg), and makes a pass/fail decision based on the outcome of the comparison. The Authentication algorithm described in Annex A.9 defines both the challenger and the challengee functions. All the server calls are synchronous.

11.12 Encryption Algorithm

The encryption algorithm, described in detail in Annex A.9 facilitates one way encoding rather than real encryption. It uses a 48-bit encryption key, K, a variable length APDU, A[len], and a 64-bit input string, R, to produce a 64-bit

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

output string, Y. Desirable properties of the random number R are defined in 11.14. Any 48-bit number is a valid encryption key.

11.13 Retries and the Role of the Checksum Function

The checksum function, defined in Annex A.12, is used for validating APDUs in client retries. The client shall retry if any of the original message, the challenge, the reply, or the acknowledgement/response is lost. Upon receiving a retry, the action taken by the server is a function of the transaction state as follows:

waiting	server is waiting for the authentication record. In this case, server shall attempt to allocate the record again.
authenticating	server has issued a challenge and is waiting for a reply. In case, the server simply reissues the same challenge (with same random number).
authenticated	authentication exchange has completed, with successful verification. If the original message was acknowledged, then the acknowledgement is reissued and the retry is discarded. If the original message was a request of an unknown type, then it is assumed that the application is still composing the response, so the retry is discarded. If the original message was a non-idempotent request, the response is reissued and the retry is discarded. If the original message was an idempotent request, then the retry APDU shall either be saved, or a checksum shall be computed over it as an alternative to save memory. The result is compared with that saved from the original message (actual message contents or checksum). If they do not match, the retry is marked as not authenticated. In any case, the retry is delivered to the application.
not authenticated	authentication exchange has completed without successful verification. The action taken is identical to that for the “authenticated” state, except no encryption/comparison is done for idempotent requests.

Note that messages received on the flexible domain are not authenticated.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.14 Random Number Generation

The random number generator used by the authentication protocol should have the following properties:

i) R, the number generated is mathematically random and unpredictable;

ii) generator does not generate predictable values after events such as power failure or rebooting.

11.15 Using Authentication

The authentication scheme must be correctly used to provide maximum security. One problem that the user should be aware is the transportation of authentication keys in the open using a network management command. This problem can be overcome by using the increment authentication key network management command (see 13.7.6 and Annex A.11) rather than the network management command that provides an absolute value for the key (see 13.7.4 and Annex A.30).

12 Presentation/Application Layer

12.1 Assumptions

The application layer makes no assumptions apart from relying on the transaction control sublayer for correct TPDU/SPDU sequencing and duplicate detection. The provided functions of the presentation layer are specified as a part of the APDU header. In particular, when the APDU header indicates that the APDU is a network variable update, the header has presentation information encoded within it because it tells the node how to interpret the APDU data.

12.2 Service Provided

The presentation/application layer provides six services:

— Network Variable Propagation. This service sends messages that are interpreted by the receiver(s) as network variable updates. A special two-byte header is used to convey the presentation layer information that the APDU is to be interpreted as a network variable. Network variables are propagated using any of the protocol services. Request/Response is used when network variables are polled.

— Network Variable Aliasing. This service allows multiple network variable inputs or outputs to be aliased to a primary network variable. Then when the application updates a primary network variable output, additional packets are also sent for the network variable aliases corresponding to the primary. When a primary input or an aliased input is updated from the network, the application receives an event that the primary variable has changed its value.

— Generic Message Passing. An application may construct an arbitrary message, addressed using any of the addressing modes;

— Network Management Messages. These messages are described in detail in 13.7;

— Network Diagnostic Messages. These messages are typically initiated by a network management tool to test that nodes are fully operational, and to take corrective action around problem areas. These messages are described in detail in 13.8;

— Foreign Frame Transmission. These messages originate external to an environment of this standard, and are destined for nodes also external to the environment. This function is provided as a means to use this protocol as a gateway between two such external nodes, or to tunnel other protocols through this protocol.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12.3 Service Interface

The service interface to the application program has the form shown in Figure 29.

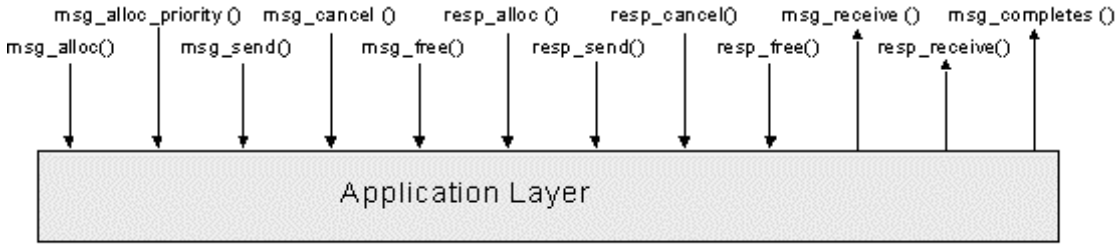

Figure 29 — Application Layer Interface

The service primitives at the application layer interface are described in Table 1 below. Most of them require no parameters. Instead, they operate on the data structures msg_out, msg_in, resp_out and resp_in, described in Annex A.12.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 1 — Application Layer Primitives

Service Interface Primitive	Description
msg_alloc ()	This Boolean primitive allocates a non-priority buffer for an outgoing message. It returns TRUE if a msg_out data structure has been allocated. It returns FALSE if a msg_out data structure cannot be allocated. If it returns FALSE, the application program can continue with other processing if necessary rather than waiting for a free message buffer.
msg_alloc_priority ()	This Boolean primitive allocates a priority buffer for an outgoing message. It returns TRUE if a priority msg_out data structure has been allocated. It returns FALSE if a priority msg_out data structure cannot be allocated. If it returns FALSE, the application program can continue with other processing if necessary rather than waiting for a free priority message buffer.
msg_send(msg_out)	This primitive sends a message in the msg_out data structure.
msg_cancel()	This primitive cancels the message currently being built. It frees the associated buffer, allowing another message to be constructed.
msg_free()	This primitive frees the msg_in data structure for an incoming message.
resp_alloc()	This Boolean primitive allocates a buffer for an outgoing response. It returns TRUE if a resp_out data structure has been allocated. It returns FALSE if a resp_out data structure cannot be allocated.
resp_send(resp_out)	This primitive sends a response in the resp_out data structure.
resp_cancel()	This primitive cancels the response currently being built. It frees the associated resp_out data structure, allowing another response to be constructed.
resp_free()	This primitive frees the resp_in data structure for an incoming response.
msg_receive(msg_in)	This primitive receives a message in the msg_in data structure.
resp_receive(resp_in)	This primitive receives a response in the resp_in data structure.
msg_completes()	This Boolean primitive evaluates to TRUE when an outgoing message completes (that is, either succeeds or fails).

msg_out and msg_in can have any of the formats described in 12.4.

12.4 APDU Types and Formats

The APDU consists of a header followed by the application data. The header is a single byte, that is followed by a second byte only if the header specifies that network variable information is to follow. The data structure for the APDU is given below:

```
struct message
{
   byte destin_type;
   byte data[];
};
```

where data is an open ended array and destin_type is one of the following:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
00xxxxxx   generic application message (64 codes)
1dxxxxxx   a network variable message; "d" indicates direction: 1 for outgoing, 0
           for incoming. The remaining code bits are combined with the first data
           byte to form a 14 bit network variable selector.
011xxxxx   a network management message (32 codes)
0101xxxx   a diagnostic message (16 codes)
0100xxxx   foreign frame (16 codes)
```

The rest of the APDU is defined with the first byte received as leftmost and the last byte received as rightmost. Any 2- or 4-byte quantities stored in the APDU are stored with the most significant byte on the left. The leftmost bit in a byte is the most significant bit. Arrays are stored with the lowest numbered element on the left. Structure fields are also stored left to right.

Every node compliant to this standard shall be able to receive, as a minimum, an APDU of 16 bytes of data, plus the destin_type.

The application protocol data unit (APDU) has the format shown in Figure 30:

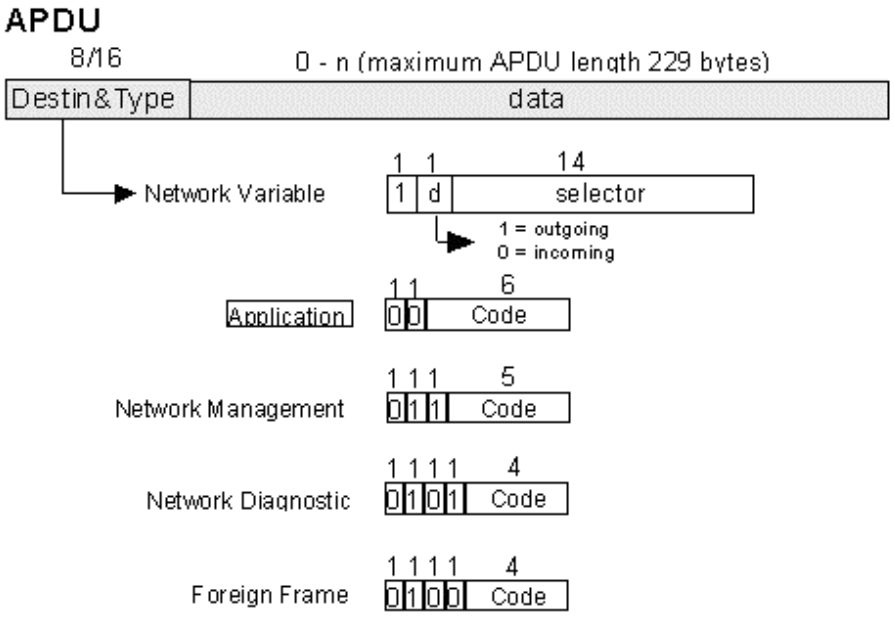

Figure 30 — APDU Format

For description of the network management message codes, see 13.7; for diagnostic message codes, see 13.8.

12.5 Protocol Diagrams

Figure 31 shows a Non-Idempotent multicast transaction with a loss of both the initial APDU and the ACK TPDU.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

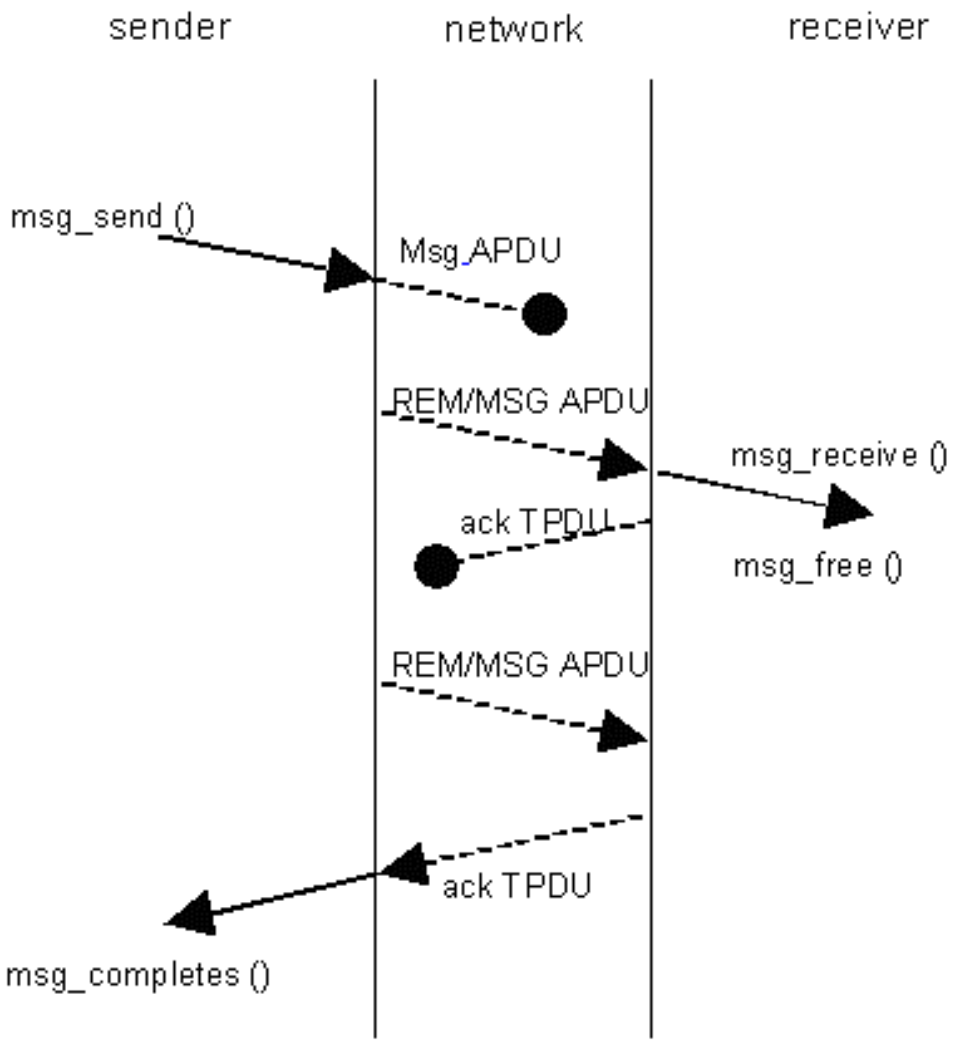

Figure 31 — Application Protocol Diagram for Multicast Acknowledged Transaction

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

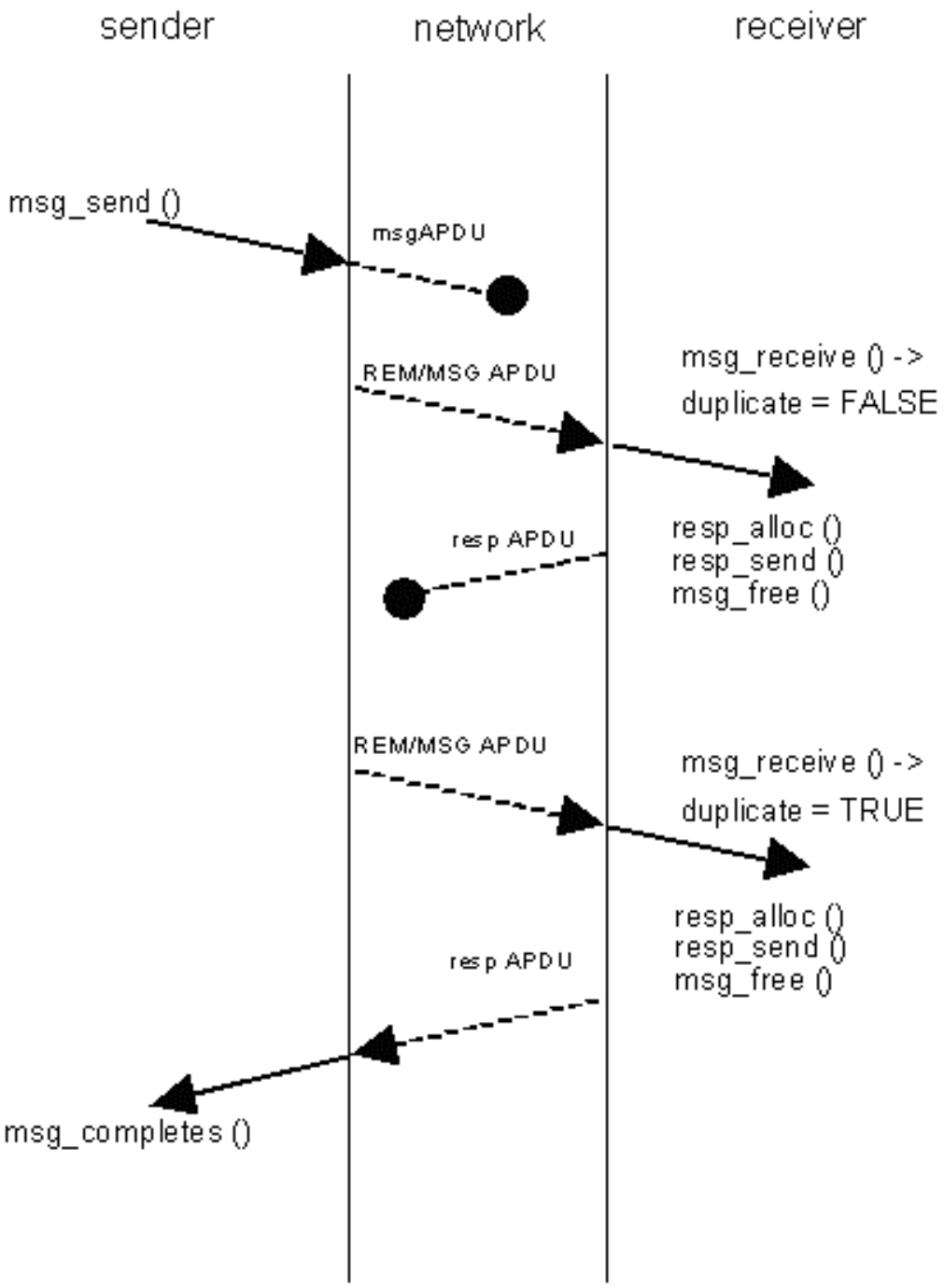

Figure 32 — Application Protocol Diagram for Multicast Request/Response Transaction

Figure 32 shows an Idempotent Multicast Request/Response transaction with a loss of both the Request and Response.

12.6 Application Protocol State Variables

The address format data structures are listed in Annex A.14. The msg_out, msg_in, resp_out, and resp_in structures use these address formats to direct messages to their destinations.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12.7 Request - Response Messaging in Offline State

When using the request/response mechanism either explicitly, or implicitly as with a network variable poll, it is possible to issue a request to a node where the application program is offline, and thus unable to respond. When this condition occurs, what happens depends on whether the response is to a network variable poll or to any other message. If the response is to an application message, the destin_type in the APDU shall be set to 63. When the response is to a foreign frame, the destin_type in the APDU shall be set to 79. When the response is to a network variable poll the response shall contain the network variable selector, but shall not have any associated data. This is also the response received if one attempts to poll a network variable on a node that has no matching selector, or if an authenticated poll fails.

12.8 Network Variables

12.8.1 General

Network variables are logical inputs and outputs on a node. Multiple network variables may be defined on any node, thus network variable messaging permits the propagation of a specific data value from one node to one or more other nodes. Network variables are implemented as a special type of application message where the most significant bit of the APDU is set to 1. They have properties of direction (either input or output), properties of propagation (polled, synchronous), and properties of communication services (priority, acknowledged, unacknowledged, unacknowledged but repeated, authenticated). Network variables may be in connections. Input network variables are updated with a new value only when they receive that new value from an output network variable. Polled network variables are only propagated when the requestor sends a request/response message containing the network variable selector to the node.

A network variable is propagated at the discretion of the application. For example, the network variable might be propagated any time the variable is written to or, alternatively, any time it is changed. In any case, the application must ensure that the propagated data for a multi-byte network variable is consistent within itself. A non-polled output network variable may be ‘synchronous.’ In this case all values written by the application shall be propagated onto the network. Conversely, a non-synchronous network variable does not require that all values assigned to the network variable actually get propagated onto the network. How, an application propagates non-synchronous network variables is implementation dependent. Network variables are configured when they are connected for propagation using any of the lower layer protocol services such as priority, authentication, acknowledgements, etc.

A network variable connection is defined when a network variable on a sender node is configured to have selector value that matches the selector value of a network variable defined on one or more receiver nodes and that network variable is bound via an address table entry to the other node or nodes. A node may have an output network variable defined on it that is connected to one of its own input network variables. This connection is called a turnaround connection even if the output is also sent to another receiver node across the network.

Nodes do not send network variable messages when they are in the offline state. The only exception to this is when a network variable is polled on a node that is offline. In this case, the node responds to the network variable poll message with the proper selector value for the network variable, but no data for the value of the network variable.

Network variables may have aliases. An alias network variable inherits the length, direction and type of the network variable it is aliased to (the primary). Aliased network variables always have a different network variable selector value, and may have a different destination address than the primary network variable. This allows, in effect, multiple selector values to be associated with a single network variable. Multiple selector values are useful to allow network variable connections where the constraint of a single network variable selector would result in selector conflicts(the same selector value in use for multiple variables on a single node).

12.8.2 Network Variable Processing

Whenever an application program updates an output network variable, the application layer first checks to see if there are any readers of the variable on the network. Checking for the network variable having an associated address table entry does this. If there is such an entry, the application layer encodes a packet containing the network variable selector and the network variable value. This packet is then sent to the lower layers for processing. Next the application layer checks to see if the variable has aliases. If aliases exist, then packets are

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

formed with the alias selector values along with the value of the primary network variable. Each of these packets is also sent to the lower layers for processing.

When a network variable update arrives at the receiving application layer, the selector value is compared against the selectors within the receiver node. If there is no match, the update is discarded. If there is a match, a second check is performed against the matching selector to see if it corresponds to a primary network variable or an alias. If the selector corresponds to a primary network variable, then the application layer updates the primary value. If the selector is for a network variable alias, then the application layer updates the primary network variable's value.

12.9 Error Notification to the Application Program

12.9.1 General

Regardless of whether the application program is using network variables or sending messages, it always has available to it the status of the last transaction.

12.9.2 Error Notification for Messages

Messages have three events associated with them: completion, success, or failure. Completion means that the transaction has completed (either successfully or not). For acknowledged transactions, success is defined as all acknowledgements having been received. For request/response transactions, success is defined as a response having been received from all of the intended recipients. In this case, it is up to the application to check the response code to see that the intended node was not offline. For unacknowledged transactions, the transaction succeeds when the transaction completes (when the message is sent). For unacknowledged-repeated transactions the transaction completes and succeeds when the message has been sent the requested number of times.

This European Standard does not define any failure events for unacknowledged transactions. However, specific implementations may post failure events for unacknowledged transactions due to internal errors that may occur during the transmission of a message. Acknowledged transactions provide failure notification to the application when the expected number of acknowledgements is not received. Request-response transactions fail when one or more of the responses are not received.

12.9.3 Error Notification for Network Variables

For network variables, the completion event is posted when the transaction completes. This is identical to messages. Again, as with messages, unacknowledged updates can post failure events due to error conditions detected on the transmitting node. For acknowledged network variable updates, success is defined as all expected acknowledgements having been received. Failure is then defined as one or more expected acknowledgements not being received. Network variables always use the request-response mechanism when they are polled. Polled network variable updates succeed when the target nodes have returned all of the values. A failure event is posted to the application when either:

iii) all of the responding targets did not have valid data (no matching network variable or offline), or

iv) one or more of the expected responses did not arrive. Note that for aliases, rule (ii) applies to the responses for the primary and aliases collectively.

There is a single exception to rule (i): a polled network variable that is connected to another network variable on the same node and is also in a connection with one or more network variables on other nodes will post a failure event to the application if all of the other nodes return offline responses to the poll. This is true even if the node with the turnaround connection responds successfully to its own poll.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13 Network Management & Diagnostics

13.1 Assumptions

Network Management and Network Diagnostic (NM/ND) services are application level procedures that use the session layer. This means that network management and diagnosis is only possible when the session layer (and all the underlying layers) are functioning properly.

With a few exceptions, all NM/ND commands either examine or modify contents of memory locations in one fashion or another. A portion of the various data elements that reside in non-volatile memory, such as address assignment, are supported with their own NM/ND commands for reporting and updating, allowing a more controlled execution of these operations within the protocol processor. Other areas can be read or written using special addressing modes of the read and write memory commands. Thus, users of move and change types of commands need not concern themselves with the physical layout of the non-volatile memory. Those needing to download applications must understand the physical layout of non-volatile memory (although even this information can be wholly contained within a download file).

13.2 Services Provided

The Network Management and Diagnostics services provide the following capabilities:

— Address Assignment: the assignment of all address components (except the Unique_Node_ID);

— Node Query: the querying of node status and essential statistics;

— Router table maintenance.

With a few minor exceptions, network management operations are implemented as remote procedure using the underlying request-response service provided by the session layer. See 11.5.

13.3 Network Management and Diagnostics Application Structure

Network Management is a distributed application with multiple *clients* and multiple *servers*. Server functions must be supported on all nodes, whereas client functions need only be supported on nodes used as network management devices.

13.4 Node States

A node can be in one of four states. These states are maintained in non-volatile memory and have the values listed in parenthesis after the state name. These states are reported in the response to the Query Status network diagnostic command. The numbers in parentheses are enumerated values of node_state (see 13.8.2).

Applicationless: (3) no application yet loaded, application in process of being loaded, or application deemed bad due to application checksum error. No application runs in this state. The Node Status Indicator[1)] (a diagnostic aid optionally available in nodes based on this European Standard) is on continuously.

Unconfigured: (2) application loaded but network configuration memory is either not loaded, being reloaded, or deemed bad due to a network configuration memory checksum error. A specific implementation of this standard may allow a program to make the node go to the unconfigured state. The program determines if an application runs in this state. The Node Status Indicator flashes at a one-second rate. [An application-initiated transition to the unconfigured state shall clear the authentication keys.]

Hard-offline: (6) application loaded but not running. The network configuration memory is considered valid in this state; the network management authentication bit is honoured. The Node Status Indicator is off.

1) (INFORMATIVE) The standard colour for the Node Status Indicator is yellow.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Configured: (4) normal node state. The application is running and the network configuration memory is considered valid. This is the only state in which messages for the application layer are received. In all other states, they are discarded. The Node Status Indicator is off. The configured state has an additional modifier that is the online/offline condition. This condition is not necessarily maintained in non-volatile memory. The states and online/offline condition are controlled via different mechanisms. However, they are reported together in the status command.

Note that there is a subtle distinction between being in the *configured* or *unconfigured* states and a node being referred to as either configured or unconfigured. A node in the *configured* or *unconfigured* state is as described above. However, a node is referred to as configured if it is in either the *hard-offline* state or the *configured* state (having valid network configuration memory contents in either case). A node is referred to as unconfigured if it is either in the *applicationless* or *unconfigured* states (no valid network configuration memory contents in either case). The network management authentication bit is honoured only when the node is in the *configured* state or the *hard-offline* state.

13.5 Using the Network Management Services

13.5.1 General

Most Network Management PDUs (NMPDUs) are conveyed within session layer requests and/or responses. By default, an NMPDU inherits either the request or the response attribute of the enveloping SPDU. However, some requests, are conveyed within TPDUs rather than SPDUs. Commands that use TPDUs are so noted. Some commands, such as Query_ID() may be sent using broadcast addressing and request/response service. Such a command will succeed when at least one response is returned from the network.

When configured to do so, most NM/ND transactions must be authenticated in order to take effect. Authentication is not possible for messages that are addressed using Unique_Node_ID addressing where the server is not in the same domain as the one that the client used to initiate the request. Commands that do not require authentication to be executed are so noted.

If a node does not understand a particular network management request, it shall return a failed response. See 13.7 for encoding of network management responses.

13.5.2 Addressing Considerations

The transmit transaction timer value of the client node must be extended to handle the lengthy delays involved with any command that alters non-volatile memory. When Unique_Node_ID addressing is used, the server node automatically extends the non-group receive transaction timer to about 8 seconds. This allows this timer to be tuned for normal application traffic without concern for lengthy network management transactions.

The recommended addressing mode for initially using these commands on a node is Unique_Node_ID. Once the node has been assigned an adequate non-group receive timer value (for duplicate detection) and a domain, subnet, and node field then subnet/node addressing is recommended.

Unique_Node_ID addressed messages are received regardless of the domain in which they are sent. Unconfigured nodes shall also accept any subnet or domain wide broadcast regardless of the domain. In both of these cases, acknowledgements and responses are returned on the domain in which the message was received with a source subnet/node pair of 0/0. Messages received in a domain in which the node is not a member (either because the node is unconfigured or not in the domain) are termed as being received on the Flexible domain. Some commands are not permitted under these circumstances and are noted in the descriptions of the network management commands starting at 13.7.2.

A significant advantage of using Unique_Node_ID addressing for network management commands is that if a node accidentally becomes unconfigured (e.g., due to a checksum error resulting from a power failure while changing the network configuration memory), the network management tool does not lose its ability to communicate with the node.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13.5.3 Making Network Configuration Changes

The paradigm for making network configuration changes within a node is as follows:

18) Alter the node state or condition (optional);

19) Perform the change or changes;

20) Update the configuration checksum (only necessary if not done in step 2);

21) Return to step 2 if more needs to be done;

22) Restore the node state or condition if changed in step 1;

23) Reset the node if communication parameter changes were made and it is desired that they take effect.

13.5.4 Downloading an Application Program

The paradigm for downloading applications is as follows:

24) Take the node offline;

25) Alter the node state to *applicationless*;

26) Send messages to the node that load the application into the node's memory;

27) Reset the node;

28) Compute the application checksum.

29) Enter the unconfigured state.

At this point, the network configuration memory can be loaded. Note that when loading an application followed by loading of the network configuration memory, a node comes up in the offline condition.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.5.5 Error Handling Conditions (Informative)

There are several classes of errors to consider:

Transaction Failures: If a transaction fails (i.e., the desired acknowledgement or response is not received), it is best to attempt to get to a known state rather than simply retry the transaction. If network management authentication is turned on, returning to a known state should include attempting authenticated transactions using different keys (e.g., the current key, the previous key, etc.) until success is achieved.

Node Resets or Power Cycles: if a node resets while a network management command is in progress, the reset will likely manifest itself as either a communication problem or a transaction failure. When non-volatile memory writes are involved, depending on the implementing technology, the location being modified at the time of the reset may become corrupted.

A reset during a *memory refresh* command could result in the corruption of the network configuration memory or application program. Either could be catastrophic depending on the scope of knowledge in the network management tool. An option here is to put early power down detection on the network management tool and only issue refresh commands (with no retries) when the power appears stable (assuming the client and server share a common power source).

Read/Write Protect Violations: if a node is read/write protected, attempts to write to the application code area are denied. The client can verify that a write memory attempt failed for this reason by reading the read_write_protect field of the read_only_data structure.

Other adverse effects, such as address table and domain changes, need to be carefully handled and understood by the client. A request can produce a response in a new domain, for example.

Every network node maintains two checksums, one over the network configuration memory and one over the application memory. Following the completion of any of the network addressing commands that alter the network configuration memory, a new configuration checksum is calculated and updated. This adds time to the execution of these commands, and the client should take this into account before sending the next message to the target node. This delay should always take into account the non-volatile memory write time multiplied by the number of bytes altered. The delay per byte can be as long as 20 ms. Therefore an *update address* command should have a transmit transaction timeout of at least 20 x 5 + 30, or 130 ms.

Commands that automatically update checksums are noted.

An implementation may choose to validate the parameters to a given network management command. Not all senders of network management commands actually send commands of the exact length. This makes validation based upon the length of the command more difficult. Senders of network management commands shall not exceed the lengths tabulated below.

The response to an invalid NM/ND command should be a failed response. Length validations are based on the chart below. Note that if there is more than one number listed for a given command, then a parameter set that is valid for a smaller size must also be accepted if padded to the larger size. Escape commands are issued by setting a parameter to the command to 0xFF. The escape commands and rules are noted in this clause and in the data structures in Annex A.

Command	Code(hex)	Data Size	
Query Status	51	0	
Proxy Agent	52	12	(Unique Node ID addressing)
Proxy Agent	52	6	(All other address formats)
Proxy Target	52	1	

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Command	Code(hex)	Data Size	
Clear Status	53	0	
Report Transceiver Sts	54	0	
Query ID	61	1	
Query ID (conditional)	61	5+n	(n=number of conditional bytes)
Respond To Query	62	1	
Join Domain	63	16	
Leave Domain	64	1	
Security	65	7	
Modify Address	66	5	(turnaround only)
		6	(non-turnaround only)
Report Address	67	1	
Report NV	68	1	(non-escaped only)
		3	(escaped only)
Update Address	69	5	
Query Domain	6a	1	
Modify NV	6b	4	(regular update)
		5	(alias update)
		5	(null alias update, primary==0xff)
		6	(escaped regular)
		7	(escaped alias)
		9	(double escaped alias)
Node Mode	6c	1	(not for state change)
		2	(for state change)
Read Memory	6d	4	
Write Memory	6e	5+n, 16	(n=number of bytes to write)
Checksum Recalculate	6f	1	
Install	70	0	(basic application wink)
Install	70	1	(For multiple protocol stacks/node)
Install	70	9	(definition of network variables)
Install	70	5	(removal of network variables(nv))
Install	70	7	(query nv self documentation(sd) text)
Install	70	4	(all other query nv commands)
Install	70	5	(query node information)
Install	70	20	(update nv name)
Install	70	7+N	(update nv self documentation text)
Install	70	7	(update network variable rate estimate)
Install	70	5	(update network variable SNVT index)
Install	70	5	(NM_GET_CAPABILITY_INFO)

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Command	Code(hex)	Data Size	
Install	70	4+N	(NM_SET_NV where N is the NV size)
Install	70	4	(NM_NODE::NM_INITIALIZE)
Install	70	7	(NM_DOMAIN::NM_INITIALIZE)
Install	70	20	(NM_DOMAIN:: CREATE / UPDATE)
Install	70	5	(NM_DOMAIN::NM_ENUMERATE reqest)
Install	70	18	(NM_DOMAIN::NM_ENUMERATE resp)
Install	70	12	(NM_DOMAIN::NM_SET_AUTH)
Install	70	7	(NM_ADDRESS::NM_INITIALIZE)
Install	70	12	(NM_ADDRESS::CREATE/ NM_UPDATE)
Install	70	5	(NM_ADDRESS::NM_ENUMERATE)
Install	70	10	(NM_ADDRESS::NM_ENUMERATE resp)
Install	70	7	(NM_NV_CONFIG::NM_INITIALIZE)
Install	70	7	(NM_NV_CONFIG::CREATE/ UPDATE)
Install	70	5	(NM_NV_CONFIG::NM_ENUMERATE)
Install	70	10	(NM_ENUMERATE resp)
Install	70	7	(NM_ALIAS_CONFIG::NM_INITIALIZE)
Install	70	14	(NM_ALIAS_CONFIG::CREATE/UPDATE)
Install	70	5	(NM_ALIAS_CONFIG::NM_ENUMERATE)
Install	70	12	(NM_ALIAS_CONFIG::NM_ENUMERATE resp)
Memory Refresh	71	4	
Query SI Data	72	3	
NV Fetch	73	1	(non-escaped only)
		3	(escaped only)

13.6 Using Router Network Management Commands

The router shall follow the normal protocol processor “states” and must be in the Application/Configured state in order to operate fully as a router.

All of the commands that affect the routing tables affect only a single router half. The NM Node Mode command for OFFLINE, ONLINE, and RESTART shall automatically affect BOTH router halves.

For a router, ONLINE means that the router shall operate normally as described. OFFLINE means that the router performs no forwarding; all packets not addressed to the router that appear in the packet buffer circular lists are dropped. Other than the dropping of these packets, an OFFLINE router continues to perform normally.

A router shall ignore a certain group of Network Management commands. These commands are the broadcast Node Mode commands. This group of commands are ignored to prevent a broadcast RESTART or OFFLINE command from stopping the router and preventing the same broadcast command from reaching destinations on the other side of the router. Routers therefore must be RESTARTed or taken OFFLINE individually when desired. To support this, it must be possible to initiate a unique *Manual Service Request* for the router.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13.7 NMPDU Formats and Types

13.7.1 General

This subclause lists Network Management APDU (NMPDUs), using a notation similar to C structure definitions. The bit and byte ordering rules defined in Annex D apply, with the most significant bit of each byte being transmitted first; the first byte of a record is considered the least significant byte of that record. In the value section of the descriptions, the value corresponds to a command number or a response code for that message.

The first byte of all NMPDUs contains the Destination/Type data that, for NM requests and commands, is always (binary):

011xxxxx

The <xxxxx> field contains the command code.

Responses that have been generated by the execution of these NM commands are directed to the Application, as specified by the first byte of the APDU:

00pxxxxx

The <p> field is set to one if the operation succeeded, or zero if it failed. Failures are usually due to range errors (table boundaries) or non-volatile memory write failures. The <xxxxx> field echoes the original NM command code.

The first byte of all ND message APDUs contains the Destination/Type data of:

0101xxxx

The <xxxx> field contains the command code.

ND responses have the following format, where <p> is the same as in NM responses and <xxxx> mirrors the original command:

00p1xxxx

The implications of this are that all NM/ND requests are delivered to the NM/ND layer, while all NM/ND responses are delivered to the application layer. It is assumed that the responses are to be processed at the application layer.

In this European Standard, only the command field value is described. The <p> field and the destination code are not included but are assumed to be in place.

Single byte responses are provided for NM operations that are considered non-idempotent.

NOTE 13.7 uses “byte” for 8-bit items and “uint16” for 16-bit items.

13.7.2 Query ID

Query_ID() requests a node, or a set of nodes, to report its Unique_Node_ID to the requester. Typically this request is addressed on a single domain as a subnet-wide broadcast, implying that the client has knowledge of at least the domain and may be taking orderly probes at subnet addresses in order to interrogate a set of nodes. The data structures used are described in Annex A.15.

To query unconfigured nodes, the selector value within the Query_ID_Request record is set to ‘0’. In order to query nodes whose “respond to query” bit is set (see following command), the selector value is ‘1’. To query nodes that are unconfigured and whose “respond to query” bit is set, the selector value is “2”. Either the subnet or domain wide broadcast addressing mode is typically used. This command never requires authentication to be executed.

If supplied, the address and data fields are used as additional qualifiers. The address mode and address field are used to form an address (see “read memory” for a description of this process); “count” bytes (1–11) of data starting

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

at that address are then compared with the supplied data. Only if they match does the Query ID proceed (as specified by the “selector”).

For this command only, read protect is assumed to be always on. If the address and count fall in a read protected area (such as where the authentication keys are stored), no response is returned.

13.7.3 Respond to Query

This command sets or clears the “respond to query” bit in the target node(s). When set, the target shall respond to Query_ID()requests that have a selector of ‘1’. It shall continue in this mode until the node is reset or its bit is cleared via command. This command is used for network topology interrogation. The “on” version is usually addressed as subnet broadcast, using the unacknowledged-repeated service. The “off” version is addressed to a specific node once it has been interrogated. This command never requires authentication to be executed. The data structures are defined in Annex A.16.

13.7.4 Update Domain

This command updates one of the domain entries in the server, using the data structures in Annex A.17. Note that the most significant bit of the node field must be set. Execution of this command updates the network configuration memory checksum. If a node can only be in a single domain, attempts to assign domain index ‘1’ shall return an error. If the domain to be updated is the same as the domain in which the modify message was sent, and the node is in the “configured” state, then the response shall come back on the new domain and thus shall not be received by the sender. This command may also be used to leave a domain by setting the domain length to 0xFF, setting the subnet and node addresses to zero, and invalidating the authentication key. Note that the side affects of resetting the node and going to the unconfigured state that automatically occur when using the Leave Domain command will not occur when the domain is left using this command.

Since the encryption key is propagated in the clear, this request should only be used when physical network security can be guaranteed (or security is achieved through other means).

13.7.5 Leave Domain

The node must honour this command even if it still has addresses assigned within this domain. Internally, the node’s domain length is set to 0xFF, and the subnet and node addresses are set to zero. Also, the authentication key is cleared. The network configuration memory checksum is updated during the execution of this command. If the domain to be left is the domain on which the request was received and the node is *configured*, no response is sent. If the domain being left is the last domain in which the node is configured, the node automatically enters the unconfigured state and resets. The data structures used are shown in Annex A.18.

13.7.6 Update Key

This command is used for updating encryption keys, using the data structures in Annex A.19. The domain to be used is specified in the message. The encrypt_key bytes are added to the existing key in a bytewise fashion (no carry). The network configuration memory checksum is updated by this command.

13.7.7 Update Address

This command is executed by index. If the address table entry does not exist, the <p> bit in the response shall be zero. The network configuration memory checksum is updated by this command. No cross-checking for duplicate addresses or groups is performed.

The form of each address entry in the address table is described in Annex A.20

13.7.8 Query Address

This command reports an entry within the node’s address table, given an index. The data structures used are in Annex A.21.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

13.7.9 Query Network Variable Configuration

This command reports the entry in the node's nv_config table, by index number; the entry must exist in the table. The data structures used are in Annex A.22.

This command can also be used to query the node's alias table. The same data structures are used for alias queries as for network variable queries (see Annex A.22). When used for aliases, the index in the command data structure is the alias entry index and the primary index is the index of the primary network variable corresponding to the indexed alias.

13.7.10 Update Group Address

This command is used to update a group entry in an address table; it is typically addressed by group. The group size, timer indices, and retry count are updated. The group member field is left unchanged. The entry is updated based on the domain in which the command was received because group addresses are only unique within a specific domain. Therefore, this command is disallowed for the flexible domain. This command updates the network configuration memory checksum. The data structures used are in Annex A.23.

This operation shall update only the first matched instance of the group address.

13.7.11 Query Domain

This command is used to retrieve the domain information for one of the two domains in a node. If the second domain is requested and room for only one domain exists, the response contains the command failure indication. The data structures used are in Annex A.24.

13.7.12 Update Network Variable Configuration

This command is used to add or modify entries to or from the node's nv_cnfg table, by an index number. There must be free space in the nv_cnfg table for the entry. The address table index may be set to 0–15, where 15 indicates that no address table entry is associated with the network variable. The network configuration memory checksum is updated by this command. For a network variable with index X, a network variable selector with the value 0x3FFF-X implies that the network variable is not in a logical connection (i.e., is not bound). The data structures used are in Annex A.25.

This command can also be used to update the node's alias table. The same data structures are used for alias configuration table updates as for network variable configuration table updates (see Annex A.25). When used to update the alias configuration table, the index in the command data structure is the index of the alias entry index and primary index is the index of the primary network variable corresponding to the indexed alias.

13.7.13 Set Node Mode

This request instructs the application to enter either the offline or online condition, to reset the entire node via an internal reset, or to change the state of a node. When offline, the application program is halted and NM/ND commands continue to be processed. The online request instructs the application scheduler to leave the offline condition and resume operation of the application. One use of the offline condition is for suspending the application during application non-volatile memory downloading.

The service type used for this command varies. For online and offline, no response is ever returned so request-response cannot be used. The receiver will honour a request using the request-response service, however, the state transitions for online and offline prevent the response from being sent. Confirmation of the change in condition is achieved via issuance of a *status request*. For state changes, request-response should be used. Since the state is part of the application, the application checksum is updated. For reset, only the unack'd (or ack'd if authentication is required) service type is used. This message is confirmed with a sequence of network management commands. First the node's status is cleared. Then the node is issued a reset. Finally, the status request command is issued to the node. Note that failure to confirm the reset may indicate that the initial unack'd message was lost, necessitating a retry of the exchange. The data structures used are in Annex A.26.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.7.14 Read Memory

This command is used to read memory. If read/write protect is on, only the read_only_data, config_data, and RAM or non-volatile memory data areas can be read.

The “count” field contains the number of bytes to be read. This number should not exceed 16 unless the target node has buffers sufficiently large to accommodate the additional data. Addresses to read may be specified relative to the read-only structure (see B.1), relative to the configuration structure (B.6), relative to the statistics structure (B.7), or may be absolute addresses. The absolute addressed memory space that can be read is up to 65 535 bytes in length starting from an implementation specific physical offset. A read of absolute memory location 0 with a count of 1 byte returns the original system image version number. This can be used to determine system capabilities when the firmware version number indicates a custom version. The custom version is assumed to be derived from the original system image and thus inherits its capabilities. The data structures used are in Annex A.27.

13.7.15 Write Memory

There are two forms of this command: one form resets the protocol processor after writing, the other does not. Confirmation of the reset form must be performed by reading back memory using *read memory*. The non-reset form produces a response and is conveyed via request-response. The reset form is conveyed using unacknowledged or unacknowledged-repeated service. The network configuration memory checksum and/or application checksum are optionally updated. A node compliant to this European Standard is not required to support write memory commands that span memory technologies. For example, a single write command that writes both flash EPROM memory and EEPROM memory need not be supported.

If read/write protect is on, only the config_data_struct (see Annex B) can be written. The byte count should be limited to 11 bytes unless the target node has buffers that are sufficiently large to handle more. Addresses to write may be specified relative to the configuration structure (see B.6), relative to the statistics structure (see B.7), or may be absolute addresses. Just like in the Read Memory command, absolute addresses can be written in a space up to 65 535 bytes in length starting from an application specific physical offset. The data structures used are in Annex A.28.

13.7.16 Checksum Recalculate

This command forces the protocol processor to compute and store a new network configuration memory or application checksum. It should be used at the end of any Network Management sessions that alter the network configuration non-volatile memory image or application non-volatile memory/RAM image (unless those commands have specifically performed this operation already, as with the address commands). All nodes shall checksum their network configuration non-volatile memory. A checksum of the application image is optional. Nodes that do not compute application checksums always report success when commanded to recalculate their application checksums via this command. The data structures used are in Annex A.29.

13.7.17 Install

This command is used during installation for a variety of purposes. The primary purpose is to force the execution of an application specific wink function. For example, the wink function may blink a light attached to the node for positive node identification. The wink function, if implemented, shall execute even if the node is in an unconfigured state.

A second purpose of this command is to allow multiple protocol processors to exist with a single application. Each protocol processor would have its own unique node ID and associated domain and address tables. This second form of the command allows individual queries to the protocol processors for installation purposes. The scenario of use is that the application is installed by issuing a manual service request to get the unique node ID. Then this command is used to query the application about how many protocol processors are attached to it. With each request, the application responds with the information contained in the manual service response message for the next protocol processor along with a network interface number. Once all the network interfaces have been identified, commands to get the next network interface fail. This command is seldom used, and will likely not be used in the future due to the wide range of protocol processor capabilities enabled by this European Standard.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A third purpose of this command is to support the dynamic (run time/install time versus compile time) creation and deletion of network variables. Support of this feature is optional due to its resource constraints. These commands, that are an extension of the Install command, provide methods to query Self Identification/Self Documentation (SI/SD) data, update SI/SD data, inform the node of a new Network Variable addition, or the removal of an existing Network Variable.

A fourth purpose of this command is to extend the limitations on domain membership and group membership by increasing the upper limit on domain membership to as many as 65 535 (implementations may support less), and increase address table entries to an upper limit of 65 535 (again, an implementation may support less). Support of this feature is optional due to its resource constraints.

The general format of the install commands for the first three purposes is:

<Application Command> <Application Specific Data Fields>

To extend the limitations on domain and group membership, the command structure above is generalised. In this new structure, a node is viewed as having resources (such as network variable configuration table entries) that are managed with commands. Some commands act upon the entire class of resource, while others act upon a specific instance of the resource where that instance is selected by an index parameter (such as an address table index). Resources may have properties that can be set or read. The general format of the install commands for this family of commands, the *extended network management commands*, is:

<Application Command> <Resource> <Resource/Command Specific Data>

The resource field appears with commands 0x20 or greater. Commands 0 to 6 are legacy commands and do not use the extended format. Commands 7 to 0x1F are for non-resource specific commands, and do not have the resource field.

For commands that operate on an instance of a resource, a resource index is also required. For commands that operate on a property of the resource, either a special command or an explicit property field is required.

Resource

A resource is an object, such as an address table entry, that can be configured or queried. A resource may be dynamic or static (both exemplified by network variables), and may have properties that can be acted upon. A resource is specified by an 8-bit resource class id, and a resource index that refers to a specific instance of a resource. The base and size of the index depends on the resource, but is typically zero-based and one or two bytes. When a command acts on the entire resource class, the resource index may be absent. If a command can act on either a resource instance or all instances, then by convention an all 1's index indicates all instances.

The resource class id is partitioned into 125 system resources, 125 device-specific (or application-specific) resources, and 4 values reserved for future expansion. No device-specific resources are currently defined. The following table defines the range and currently defined resource codes:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 2 —Resource Codes

Resource Class ID	Mnemonic	Description
System Resources (MSB=0)		
0x00		(reserved for future use)
0x01	NM_NODE	Node
0x02	NM_DOMAIN	Domain Entry
0x03	NM_ADDRESS	Address Table Entry
0x04	NM_NV_DEF	Network Variables definition (static and dynamic)
0x05	NM_NV_CONFIG	Network Variables Configuration
0x06	NM_ALIAS_CONFIG	Alias Configuration
0x07 – 0x7E		(undefined)
0x7F		(reserved for future use)
Device-Specific Resources (MSB=1)		
0x80		(reserved for future use)
0x81-0xFE		(undefined)
0xFF		(reserved for future use)

Property

A resource generally has properties that may be get or set. The format for specifying a property is to use the commands NM_SET and NM_GET, and define a 1-byte property ID field following the resource. The scope of the property depends on the resource. The space of the property ID is partitioned as follows:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 3 —Space of the Property ID

0x00	(reserved for future use)
0x01 – 0x7E	Common Properties
0x7F	(reserved for future use)
0x80	(reserved for future use)
0x81 – 0xFE	Resource-Specific Properties
0xFF	(reserved for future use)

If a resource has a collection of properties of the same type (multiple instances), then an additional property index shall be defined in the request data structure following the resource index.

All of the resources defined use special command(s) for each property, and the properties are single-instance.

For both versions of the commands, the first byte of the response to the command must be either 0x30 to indicate the command was successful, or 0x10 to indicate the command failed. Additional bytes of the response, if any, depend on the application command type. For application commands with a command code greater than 0x6, failures are reported by having the first byte of the response set to 0x10, followed by a one byte cause value:

```
typedef struct {
      unsigned cause;
} NM_wink_negative_response;
```

The complete list of application commands for the install message are:

APP_WINK (0)	Command the application to perform some visible or audible action.
APP_INSTALL (1)	Find all the network interfaces for an application.
APP_NV_DEFINE (2)	Create a new dynamic NV definition.
APP_NV_REMOVE (3)	Remove an existing dynamic NV definition.
APP_QUERY_NV_INFO (4)	Query SI/SD data for an NV.
APP_QUERY_NODE_INFO (5)	Query SI/SD data for the node.
APP_UPDATE_NV_INFO (6)	Update SI/SD data for an NV.
NM_GET_CAPABILITY_INFO (7)	Query the node capabilites (address table size, etc.).
NM_SET_NV (8)	Write a network variable selected by the index.
NM_INITIALIZE (0x20)	Remove a dynamic resource or reset a static resource.
NM_CREATE (0x21)	Create a dynamic resource at a specified index.
NM_REMOVE (0x22)	Remove a resource.
NM_SET (0X23)	Set the indicated property.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NM_GET (0x24)	Get the indicated property.
NM_UPDATE (0x25)	Modify a resource definition or configuration.
NM_ENUMERATE (0x26)	Return the specified or next instance.
NM_SET_AUTH (0x81)	Update the authentication key and network management authentication for specified or all domains.

Following are the definitions of the application specific data fields by command type.

APP_NV_DEFINE

This command adds a new or modifies an existing NV definition. If an attempt is made to add or modify the definition of an NV that is part of the node's fixed NV interface, or the index exceeds the maximum supported by the node, the node will report command failed.

After a successful define, the application is responsible for setting the network variable configuration to default values (unbound selector, correct direction, etc) such that a subsequent NV config query command shows the correct default attributes for the newly defined NV.

The message data structure contains the following fields:

APP_NV_DEFINE (2)		As defined above.
unsigned	nvIndexHi	The high order byte of the NV index.
unsigned	nvIndexLo	The low order byte of the NV index.
unsigned	arrayLenHi	The high order byte of the number of elements in the NV array. The value is 0 if the NV is not an array.
unsigned	arrayLenLo	The low order byte of the number of elements in the NV array. The value is 0 if the NV is not an array.
unsigned	nvLen	The number of bytes in the NV value (1-31).
unsigned	nv_dflts	Default configurable attributes of the NV. Encoded as a single byte. Includes direction, priority, authentication, and service type.
unsigned	nv_attr	Attributes of the NV. Encoded as a single byte. (See definition below.)

APP_NV_REMOVE

This command removes one or more existing NV definitions. The NV index must be within the area of the interface that is specified to be dynamic. If the NV index specified is within the node's fixed interface, the command will fail. The command will succeed even if the specified set of NVs is not currently defined or exceeds the maximum number of NVs supported by the node. After a successful remove, the application is responsible for setting the network variable configuration to default values (unbound selector, correct direction, etc) such that a subsequent NV config query command shows the correct default attributes for an undefined NV.

The command contains the following fields:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

APP_NV_REMOVE(3)		As defined above.
unsigned	nvIndexHi	The high order byte of the NV index of the first NV definition to be removed.
unsigned	nvIndexLo	The low order byte of the NV index of the first NV definition to be removed.
unsigned	nvCountHi	The high order byte of the number of NVs to remove.
unsigned	nvCountLo	The low order byte of the number of NVs to remove.

APP_QUERY_NV_INFO

This command queries self-documentation data about a specific NV. It contains the following fields:

APP_QUERY_NV_INFO (4)

unsigned	nv_info	The self-documentation data being requested. Current values are:
	NV_INFO_DESC (0)	Basic attributes of the NV, array information, how the NV was defined, and indication of available additional self-documentation.
	NV_INFO_RATE_EST (1)	Average and maximum rate estimates defined for the NV.
	NV_INFO_NAME (2)	The name of the network variable.
	NV_INFO_SD_TEXT (3)	The self-documentation text string associated with the NV.
	NV_INFO_SNVT_INDEX (4)	The index of the SNVT.
unsigned	nvIndexHi	The high order byte of the NV index.
unsigned	nvIndexLo	The low order byte of the NV index.
unsigned	offset_hi	If nv_info is NV_INFO_SD_TEXT, the high order byte of the offset of the requested data.
unsigned	offset_lo	If nv_info is NV_INFO_SD_TEXT, the low order byte of the offset of the requested data.
unsigned	length	If nv_info is NV_INFO_SD_TEXT, the number of bytes requested. The response data structure depends upon the data requested.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If the data requested is NV_INFO_DESC, the following response is sent:

unsigned	length :5	The length of the NV value, in bytes. Encoded as 0 if the NV is currently undefined.
unsigned	origin :3	How the NV was created. One of the following values:
	NV_UNDEFINED (0)	The NV is not currently defined.
	NV_STATIC (1)	The NV was statically defined (cannot be removed).
	NV_DYNAMIC (2)	The NV was dynamically defined (can be removed).
unsigned	nv_dflts	Default configurable attributes of the NV. Encoded as a single byte. Includes direction, priority, authentication, and service type.
unsigned	nv_attr	Basic attributes of the NV.
unsigned	nv_exten	Extension bits. Indicators of additional SI/SD data that are available.
unsigned	nv_array	NV array attributes.
unsigned	nv_name[16]	Optional field. If included, the nm_supplied bit must also be set in the ext field. The name of the NV without array subscripts.

If the requested data is NV_INFO_RATE_EST, the following response is sent:

unsigned	nv_rate_est	The encoded average rate estimate for the NV.
unsigned	nv_rate_est	The encoded maximum rate estimate for the NV.

If the requested data is NV_INFO_NAME, the following response is sent:

unsigned	nv_name[16]	The name of the NV. If NV is part of an array, the name does not include the array subscript. Zero-terminated if length is less than 16 characters.

If the requested data is NV_INFO_SD_TEXT, the following response is sent:

unsigned	length	The number of bytes of SD Text data returned.
unsigned	text[*]	The SD text data segment. The actual text array size is dependent upon the preceding length value.

If the requested data is NV_INFO_SNVT_INDEX, the following response is sent:

unsigned	snvt_type_index	The index of the Standard Network Variable Type associated with the NV. Encoded as zero if the NV is not a SNVT.

APP_QUERY_NODE_INFO

This command queries self-documentation data about the node. It contains the following fields:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

APP_QUERY_NODE_INFO (5)

unsigned	node_info	The self-documentation data being requested. Values are:
	NODE_INFO_SD_TEXT (3)	The self-documentation text string associated with the node.
unsigned	offset_hi	If node_info is NODE_INFO_SD_TEXT, the high order byte of the offset of the requested data.
unsigned	offset_lo	If node_info is NODE_INFO_SD_TEXT, the low order byte of the offset of the requested data.
unsigned	length	If node_info is NODE_INFO_SD_TEXT, the number of bytes requested.

The response data structure depends upon the data requested. If the requested data is NODE_INFO_SD_TEXT, the following response is sent:

unsigned	length	The number of bytes of SD Text data returned.
unsigned	text[*]	The SD text data segment. The actual text array size is dependent upon the preceding length value.

APP_UPDATE_NV_INFO

This command updates self-documentation data about a specific NV. In general, update messages cannot be relied upon to be validated by the target application. The initiator must perform all validation. This message contains the following fields:

APP_UPDATE_NV_INFO (6)

unsigned	nv_info	The self-documentation data being updated. Current values are:
	NV_INFO_RATE_EST (1)	Average and maximum rate estimates defined for the NV.
	NV_INFO_NAME (2)	The name of the network variable.
	NV_INFO_SD_TEXT (3)	The self-documentation text string associated with the NV.
	NV_INFO_SNVT_INDEX (4)	The index of the SNVT.
unsigned	nvIndexHi	The high order byte of the NV index.
unsigned	nvIndexLo	The low order byte of the NV index.

The remaining data fields are dependent upon the nv_info value specified. If the specified nv_info is NV_INFO_RATE_EST, the following data is provided:

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

unsigned	clear_mre: 1	If set, clear the NV's maximum rate estimate value, and indicate that this data is not available.
unsigned	clear_re: 1	If set, clear the NV's rate estimate value, and indicate that this data is not available.
unsigned	update_mre: 1	If set, update the NV's maximum rate estimate value. Also, indicate that the maximum rate estimate data is available.
unsigned	update_re: 1	If set, update the NV's rate estimate value. Also, indicate that the rate estimate data is available.
unsigned	nv_rate_est	The encoded average rate estimate for the NV.
unsigned	nv_rate_est	The encoded maximum rate estimate for the NV.

If the specified nv_info is NV_INFO_NAME, the following data is provided:

unsigned	nv_name[16]	The null (zero) terminated name of the NV. The name does not include the array subscripts.

If the specified nv_info is NV_INFO_SD_TEXT, the following data is provided:

unsigned	length	The number of bytes of SD Text data being updated.
unsigned	offset_hi	The high order byte of the offset of the data to be updated.
unsigned	offset_lo	The low order byte of the offset of the data to be updated.
unsigned	text[*]	The SD text data segment. The actual text array size is dependent upon the preceding length value.

If the specified nv_info is NV_INFO_SNVT_INDEX, the following data is provided:

unsigned	snvt_type_index	The index of the Standard Network Variable Type associated with the NV. Encoded as zero if the NV is not a SNVT.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The following data structures are referenced in the preceding message definitions:

nv_attr		This data structure describes the basic attributes of a specific network variable. It contains the following fields:
	unsigned nv_sync: 1	If set, all values assigned to the NV are propagated, in their original order. Mutually exclusive with nv_polled.
	unsigned nv_polled: 1	If set, the output NV's value is sent only in response to a poll. Updates are not generated when the application updates the NV value.
	unsigned nv_offline: 1	If set, the node should be taken offline before updating the value of the network variable.
	unsigned nv_service_type_config: 1	If set, the NV's service type attribute is configurable.
	unsigned nv_priority_config: 1	If set, the NV's priority attribute is configurable.
	unsigned nv_auth_config: 1	If set, the NV's authentication attribute is configurable.
	unsigned nv_config_class: 1	If set, the NV is a configuration NV.
	unsigned snvt_type_index	The index of the Standard Network Variable Type associated with this NV. Zero if the NV is not a SNVT.

nv_exten		This data structure indicates what additional self-documenting data is available for a specific network variable. It contains the following fields:
	unsigned mre: 1	If set, the NV's maximum rate estimate is available.
	unsigned re: 1	If set, the NV's average rate estimate is available.
	unsigned nm: 1	If set, the NV's name is available.
	unsigned sd: 1	If set, the NV has a self-documenting text string.
	unsigned nm_supplied: 1	Applies only to APP_QUERY_NV_INFO response for NV_INFO_DESC. If this bit is set, then the nv_name is also provided in the response, following the existing data fields.

nv_array		This data structure provides the array attributes for a specific network variable. It contains the following fields:
	unsigned count_hi	Total number of NVs in the array. An unsigned 16 bit number.
	unsigned count_lo	
	unsigned element_hi	The index of the NV within the array. Encoded as a zero
	unsigned element_lo	if the current NV is not a member of an array.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

nv_dflts		This data structure provides the default values for the configurable attributes for a specific network variable. It contains the following fields:
	unsigned nv_direction: 1	The default direction of the NV.
	unsigned nv_auth: 1	The default authentication setting of the NV.
	unsigned nv_priority: 1	The default priority setting of the NV.
	unsigned nv_service: 2	The default service type of the NV.

Extended Network Management Commands

NM_GET_CAPABILITY_INFO

This command returns the capabilities of nodes that support no Self Identification (SI) data, or SI data versions 0 and 1. It can be viewed as returning the SI data relative to the Alias Record (alias_field). The purpose of the command is to allow a network manager to obtain the node's capability information near the end of the SI data versions 0 and 1 without having to scan the preceding and variable NV records, or to obtain the information when the node does not have SI data.

The response data has the following subclauses as described in SI Versions 0 and 1. Note that the Alias Record (alias_field) can be one or three bytes; the network manager must compute the offset of the Compatibility Record accordingly.

Alias Record (1 or 3 bytes)	Compatibility Record	Capability Info Record

NM_SET_NV

This command sets a network variable to the value specified in the command. The length of the nv must match the defined length on the receiving node otherwise a negative response is returned. The nv is specified in the command by its nv index. This command is honoured even if the receiving node is offline. If the receiving node is offline, no update event is posted to the application.

Resource Commands

The following set of extended network management commands are the commands to manage node resources.

Node Commands (NM_NODE)

The node resource supports the following command:

NM_NODE::NM_INITIALIZE

This command clears the configuration data of the node, in the following sequence:

30) Initialises all NV and alias configuration table entries of each class (see NM_INITIALIZE/NM_NV_CONFIG).

31) Initialises all address table entries (see NM_INITIALIZE/NM_ADDRESS).

32) Initialises all domain table entries (see NM_INITIALIZE/NM_DOMAIN).

33) Clears the network management authentication bit in the config data structure.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

34) Sets the node's priority to zero.

35) Removes all dynamic NVs.

36) Optionally go to a specified state.

The field state is used only if non-zero. When specified, it indicates the state the node is to go to upon completion of the initialisation.

Domain Commands (NM_DOMAIN)

NM_DOMAIN::NM_INITIALIZE

This command initialises a range of domain table entries to the empty value. This is semantically equivalent to the Leave Domain command described in 13.7.5 except that it never causes a node reset and it supports many more domain indices. The response is the standard response code format defined above.

To initialise one entry, specify both index and index_end to the index of the entry. To initialise all entries, specify index to be 0 and index_end to be all 1's. To initialise all entries at or greater than an index, specify index_end to be all 1's. Invalid indices are ignored and fail silently.

If the command leaves no other domain entries to be configured, the node goes to the unconfigured state.

NM_DOMAIN::NM_CREATE

This command uses the same data structure and has the same effect as NM_DOMAIN::NM_UPDATE. The response is the standard response code format defined above.

NM_DOMAIN::NM_UPDATE

This command updates a domain table entry. It is equivalent to the Update Domain command in 13.7.4, but supports a 16-bit domain index. The response is the standard response code format defined above.

NM_DOMAIN::NM_ENUMERATE

This command queries a domain table entry, or the next entry if the specified entry is empty. See Annex B.2 for this structure.

Enumeration Rules

At each successive enumeration, the requestor should specify the request's index to be 1 greater than the index returned from the previous response.

Only entries with a valid domain length (0, 1, 3, 6) are returned.

NM_DOMAIN::NM_SET_AUTH

This command is used to set the authentication key. The response is the standard response code format defined above.

Address Table Commands (NM_ADDRESS)

Expanded Address Table Structure

To accommodate the larger domain index, the address table entry will be appended with a 2-byte domain index. The address structures for group, subnet/node, and broadcast are affected.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The structure for the group address contains the new field “restriction”, allowing a node to send or receive messages to a group for which it is not a member. The response is the standard response code format defined above.

```
typedef enum
{
  GRP_NORMAL     = 0,
  GRP_OUTPUT_ONLY   = 1,
  GRP_INPUT_NO_ACK  = 2,
} group_restriction;
```

GRP_NORMAL – This is the normal node and is the typical default. All legacy implementations of this European Standard would use this if they did not want to modify the protocol code to support more address table entries as described in these extended network management commands.

GRP_OUTPUT_ONLY – The group is used for output only; the node must not use the entry to match the destination address of incoming addresses. This mode allows a node to send an implicitly addressed message to a group without being its member. The member id is not used and may be set to zero.

GRP_INPUT_NO_ACK – The node will receive incoming messages on that group but would not return ACKS, responses, or issue challenges; the member id may be set to zero. This mode allows a node to monitor group messages without being a member of the group. The member id is not used and may be set to zero.

NM_ADDRESS::NM_INITIALIZE

This command initialises a range of address table entries to the “empty” value. The entire entry is zeroed. The response is the standard response code format defined above.

To initialise all entries, specify index to be 0 and index_end to be all 1’s. To initialise all entries at or greater than an index, specify index_end to be all 1’s. When some indices are valid and some are invalid in a range of indices, invalid indices are ignored and fail silently.

NM_ADDRESS::NM_CREATE

This command uses the same data structure and has the same effect as NM_ADDRESS::NM_UPDATE. The response is the standard response code format defined above.

NM_ADDRESS::NM_UPDATE

This command is used to update an address table entry. Address table entries with an index greater than 14 must use this command. The response is the standard response code format defined above.

NM_ADDRESS::NM_ENUMERATE

This command queries an address table entry, or the next entry if the specified entry is empty.

Enumeration Rules

At each successive enumeration, the requestor should specify the request’s index to be 1 greater than the index returned from the previous response.

Only in-use entries are returned. Unused entries are ones whose first two bytes are zero.

NV Configuration Commands (NM_NV_CONFIG)

The network variable configuration table is used for static, and dynamic NVs, but not aliases. The table is indexed by a flat nv index corresponding to that used for the NM_NV_DEF command.

NV Configuration Table Entry Structure

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

The network variable configuration entry nv_struct is expanded to accommodate the extended network management command features. The new version is nv_struct_ext.

The index of the target nv; used when the nv configuration entry is used to target a single NV. This field is valid if its value is not 0xFFFF, and either bits nv_read_by_index or nv_write_by_index is set. If nv_read_by_index is set, the node may use NV_FETCH to fetch the target NV at the indicated index. If nv_write_by_index is set, the node may use NM_SET_NV to write to the target NV at the indicated index. The target node's address is indicated in the address table entry whose index is indicated by nv_addr_index.

NM_NV_CONFIG::NM_INITIALIZE

This command initialises a range of nv configuration entries to the "unbound" value. Each NV array element is treated as a separate entry.

To initialize all entries, specify index to be 0 and index_end to be all 1's. To initialize all entries at or greater than an index, specify index_end to be all 1's. Invalid indices are ignored and fail silently. The response is the standard response code format defined above.

The fields are initialised as follows:

Field	Initial Value
nv_priority	unchanged
nv_direction	unchanged
nv_selector_hi/lo	unchanged
nv_turnaround	0
nv_service	unchanged
nv_auth	unchanged
nv_write_by_index	0
nv_read_by_index	0
nv_remote_nm_auth	0
nv_service	unchanged
nv_addr_index	0xFFFF
Unused fields	0
nv_target_index	0xFFFF

NM_NV_CONFIG::NM_CREATE

This command uses the same data structure and has the same effect as NM_NV_CONFIG::NM_UPDATE. The response is the standard response code format defined above.

NM_NV_CONFIG::NM_UPDATE

This command is used to modify an nv or alias configuration entry. Each NV array element is treated as a separate entry. The response is the standard response code format defined above.

NM_NV_CONFIG::NM_ENUMERATE

This command queries an nv config table entry, or the next allocated entry if the specified entry is not allocated. Allocation means the corresponding NV is allocated. Each NV array element is enumerated as a separate entry.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Enumeration Rules

At each successive enumeration, the requestor should specify the request's index to be 1 greater than the index returned from the previous response.

The enumeration returns only bound NVs (one with either a bound selector, or whose address table index is not 0xffff).

Alias Configuration Commands (NM_ALIAS_CONFIG)

The alias configuration table is used for aliases of primary network variables.

Alias Configuration Table Structures

The alias entry contains the expanded nv_struct_ext and a 16-bit index of the primary NV. Unlike the structure defined in Annex A.22 and Annex A.25 that has a "short" version with an 8-bit primary index and a "long" version with a 3 bytes primary index, the alias_struct_ext below will be uniformly used in all platforms that support the extended command set.

NM_ALIAS_CONFIG::NM_INITIALIZE

This command initialises a range of alias entries to the "unbound" value.

To initialise all entries, specify index to be 0 and index_end to be all 1's. To initialise all entries at or greater than an index, specify index_end to be all 1's. Invalid indices are ignored and fail silently. The index fields in this structure are alias indices. The alias index is relative to the beginning of the alias table.

The fields are initialised as follows:

Field	Initial Value
alias_nv.nv_priority	0
alias_nv.nv_direction	0
alias_nv.nv_selector_hi/lo	0 0xFFFF
alias_nv.nv_turnaround	0
alias_nv.nv_service	0
alias_nv.nv_auth	0
alias_nv.nv_write_by_index	0
alias_nv.nv_remote_nm_auth	0
alias_nv.nv_selection	0
alias_nv.nv_addr_index	0
alias_nv.nv_read_by_index	0
Unused fields	0
alias_primary	0xFFFF

NM_ALIAS_CONFIG::NM_CREATE

This command uses the same data structure and has the same effect as NM_ALIAS_CONFIG::NM_UPDATE. The response is the standard response code format defined above.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

NM_ALIAS_CONFIG::NM_UPDATE

This command is used to update an alias configuration entry. The response is the standard response code format defined above.

NM_ALIAS_CONFIG::NM_ENUMERATE

This command queries an alias configuration entry, or the next allocated entry if the specified entry is not allocated.

Enumeration Rules

At each successive enumeration, the requestor should specify the request's index to be 1 greater than the index returned from the previous response.

The data structures used for all the install command messages are in Annex A.30.

Backward Compatibility

A node that supports the extended command set, ECS, must continue to support the legacy Network Management commands as defined in this European Standard. However, once a node has been written by any extended commands, it is considered to operate in the extended mode, and shall return a negative response to the following legacy commands. The hexadecimal command codes from this standard are included for reference.

The ECS command set contains spare bits for some of the fields. These are reserved for future use. In order to provide for backward compatibility in the future, all unused bits must be set to zero.

ECS implementations built on top of pre-ECS protocol stacks may be able to respond negatively only to network variable commands. Such nodes would have the EXTCAP_INCOMING_GROUP_RESTRICTED capability flag set. This is acceptable as long as one of the following is true:

37) 1. The node has no more than 15 bindable message tags.

38) 2. The node has at least one static network variable.

Commands Requiring a negative response if operating in extended mode:

- NM_query_nv_config (0x68)
- NM_update_nv_config (0x6B)
- NM_update_group_addr command (0x69)
- NM_query_address (0x67)
- NM_update_address (0x66)
- NM_update_domain (0x63)
- NM_query_domain (0x6A)
- NM_leave_domain (0x64)

These restrictions, responding negatively to the legacy commands if a node can, or otherwise living within the limits of no more than 15 bindable message tags or having at least one static network variable allows a legacy network manager to perform database recovery on the network without destroying the operation of the nodes that support the extended network management command set. Otherwise, the recovery of the network database from the nodes will be inconsistent and could result in a catastrophic network outage with no backup network database (since database recovery was invoked to begin with).

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Message Handling

Message Size

The node will return failure if a command data is not the exact size as defined. In the future, extensions to command data or addition of new commands will be signaled by a node via a higher version number in the field snvt_capability_info.ver_nm_max. In the future, the response to the NM_GET_CAPABILITY_INFO command may be lengthened in a backward compatible way without affecting the version number to accommodate new capabilities. In this case, if the requestor receives a response to the NM_GET_CAPABILITY_INFO command with more information than expected, the requestor can use the response data that it understands and ignore the rest of the returned response.

13.7.18 Memory Refresh

This command causes the node to rewrite the existing contents of non-volatile memory at a specified address for a specified number of bytes. This can be used to periodically rewrite the contents of non-volatile memory to extend the retention time of the memory contents. An error is returned if a refresh of memory is requested and none exists at the specified address. Also, if “offset” falls beyond the end of the non-volatile memory area, an error is returned. In this way, the sender of these commands can simply increment the offset until an error is returned. The count should be limited to 8 if the target is online and could be as high as 38 if the target is offline. The data structures used are in Annex A.31. If the non-volatile memory can be written by sector only, the count and offset values refer to entire sectors instead of individual bytes.

13.7.19 Query SI

This command is used to retrieve self-identification (SI) data for a node and for its network variables if this information is located outside the addressable memory space of the protocol processor (e.g., in the memory space of a co-processor). The field ‘si’ in the structure described in B.1.1 is a pointer to the address where a node’s self-identification data is located. A null (0) address indicates that that node provides no self-identification data. A non-null address that does not consist of all 1’s points to the address where this data is located. In this case, the data is read directly with a read memory command. An address consisting of all 1’s indicates that the self-identification information is located outside the addressable memory space of the protocol processor. The Query SI command must be implemented on the protocol processor in that case to read the self-identification data.

The device issuing this command need know nothing about physical addresses on the co-processor; instead the requester only deals with offsets. The requester starts with an offset of 0 and then bumps the offset for each subsequent request. The byte count shall be limited to 16 unless the target node has sufficiently large buffers to handle larger counts. The data structures used are in Annex A.32.

13.7.20 Network Variable Value Fetch

This message is used to poll network variables. It has two advantages over the network variable poll message: it uses the network variable index and is thus independent of network variable selector assignments, and it also obtains the value regardless of the node's online/offline condition. The data structures used are in Annex A.33.

13.7.21 Manual Service Request Message

This message is unlike the other network management messages in that it is an unsolicited message. It is sent over the network from a node when a Manual Service Request for that node is initiated. The message is sent as a domain-wide broadcast on domain length 0 with a source subnet 0 and node address of 0. The data structures used are in Annex A.34.

13.7.22 Network Management Escape Code

One of the network management command codes is reserved as an escape. The value of the escape code is 0x7D. Sending the escape code as the network management command causes the first two bytes of the APDU to be interpreted as additional command codes. This capability allows the network management protocol to be extended in product specific ways.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

If a node responds to network management messages that use the network management escape code, then that node shall always respond to the Product Query command. All other commands are product specific and are documented with the products. The response to this command has two forms. The short form contains only a single byte to specify the product. The complete form contains:

— response code,

— product byte (as in the short form),

— two byte field for the model number,

— single byte for the firmware version,

— byte for the device configuration or mode, and

— byte for the transceiver type.

The single byte product specification codes are reserved and must be allocated to ensure no conflict among implementations. In the complete response form, the value of the device configuration options/modes byte returned is zero unless the device can be put into several modes or device configurations. In this case, the byte contains the current mode or configuration of the device.

Success or failure is reported on the escape code rather than on the subcode or command. The data structures used are in Annex A.35.

13.7.23 Router Mode

This request instructs the router to perform one of several router-related tasks. The “resume” command returns the router from the “all flood” state. The “init router tables” command copies all routing tables from non-volatile memory into the RAM tables (if a configured router) or sets all RAM tables to flood (if a learning router); this is the same action that occurs after node reset. The “mode all flood” command causes the router to forward all packets in the domain. The Router Mode command affects both router halves, and is conveyed via the Request-Response protocol. Note that the normal Network Management Node Mode request may be used to take the entire router offline and online. The data structures used are in Annex A.36.

13.7.24 Router Clear Group or Subnet Table

This request is used to clear all entries in either the group or subnet routing table for a single domain for a *single router half*. The command is segmented to cover 8-byte sections in order to prevent lengthy non-volatile memory write operations. This command is conveyed via the Request-Response protocol. The network configuration memory checksum in non-volatile memory is updated. The data structures used are in Annex A.37.

13.7.25 Router Group or Subnet Table Download

This request is used to configure the entire group or subnet table in non-volatile memory for the specified domain for a single router half. The download function is broken into 8-byte sections. This command is conveyed via the Request-Response protocol. The network configuration memory checksum in non-volatile memory is updated. The data structures used are described in Annex A.38.

13.7.26 Router Group Forward

This request sets the forwarding flag in the routing table for a given group in the specified domain. This command is conveyed via the Request-Response protocol. The network configuration memory checksum in non-volatile memory is updated if changed. The data structures used are described in Annex A.39.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.7.27 Router Subnet Forward

This request sets the forwarding flag in the routing table for a given subnet in the specified domain. This command is conveyed via the Request-Response protocol. The network configuration memory checksum in non-volatile memory is updated if changed. The data structures used are described in Annex A.40.

13.7.28 Router Do Not Forward Group

This request clears the forwarding flag in the routing table for a given group in the specified domain. This command is conveyed via the Request-Response protocol. The network configuration memory checksum in non-volatile memory is updated if changed. The data structures used are described in Annex A.41.

13.7.29 Router Do Not Forward Subnet

This request clears the forwarding flag in the routing table for a given subnet in the specified domain. This command is conveyed via the Request-Response protocol. The network configuration memory checksum in non-volatile memory is updated if changed. The data structures used are described in Annex A.42.

13.7.30 Router Group or Subnet Table Report

This request is used to report the current settings of either group or subnet tables in non-volatile memory or RAM for the specified domain for a single router half. The report function is broken into 8-byte sections. This command is conveyed via the Request-Response protocol. The data structures used are described in Annex A.43.

13.7.31 Router Status

This request is used to report the router network configuration memory and flood/normal modes. It is conveyed via the Request-Response protocol. The data structures used are described in Annex A.44.

13.7.32 Router Half Escape Code

Although this is not in itself a network management command, it is included in this subclause for completeness. When this code is placed at the start of the APDU and is followed by any Network Management or Network Diagnostic command, that command shall be passed over to the *other* router half for processing. Any responses shall be returned in the normal manner.

```
byte command;    /* Destination: NM, code: 30 */
```

13.8 DPDU Types and Formats

13.8.1 General

Most Diagnostic PDUs (DPDUs) are conveyed within session layer Requests and/or Responses. By default, a DPDU then inherits either the request or the response attribute of the enveloping SPDU.

This subclause lists DPDUs. The bit and byte ordering rules defined in Annex A apply, with the most significant bit of each byte being transmitted first; the first byte of a record is considered the least significant byte of that record. In the value section of the descriptions, the value corresponds to a command number or a response code for that message. In addition, the string “+ pass/fail” means that a single bit flag is set in the high order bit of the response to indicate that the command was either successful or that it failed.

13.8.2 Query Status

This command gives a snapshot of a node’s health. It conveys error statistics, reset information, the node state, the error log, the system image version, and the protocol processor model number. The error statistics, reset cause, and error log can all be cleared via the “clear status” command. Note that the statistics are also cleared whenever

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

the node resets. This command never requires authentication to be executed. The data structures used are described in Annex A.45.

The fields are defined as follows:

- transmission_errors: This is a count of the number of transmission errors that have occurred on the network. A transmission error is detected via a CRC error during packet reception or as a packet that is less than 8 bytes long. This could result from a collision, a noisy medium, signal attenuation, etc.;

- transaction_timeouts: This is a count of the number of timeouts that have occurred in attempting to carry out acknowledged or request/response transactions. A timeout occurs when a node fails to receive all the expected acknowledgments or responses after retrying the configured number of times at the configured interval;

- receive_transaction_full: This counter reflects the number of times an incoming unackd_rpt, ackd or request message was lost because there were no receive transaction records available;

- lost_messages: This is the number of messages that were addressed to the node that were thrown away because there was no application buffer available for the message;

- missed_messages: This is the number of messages that were on the network but could not be received because there was no network buffer (packet buffer) available for the message or the network buffer was too small to receive the message;

- reset_cause: This byte contains the reset cause information. This identifies the source of the most recent reset. The values for this byte are as follows (X => don't care):

- Power-up reset `0bXXXXXXX1`

- External reset `0bXXXXXX10`

- WDT reset `0bXXXX1100`

- Software-initiated reset `0bXXX10100`

- node_state: This contains both the node state and node condition (as defined in the "Node State" section in the network management section).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The node_state structure is shown below:

```
struct node_state

{

 unsigned        :4; // application specific

 unsignedoffline :1; // 1=>Offline,0=>Online

 unsignedstate   :3; // 2=>Unconfigured

                     // 3=>Applicationless

                     // 4=>Configured

                     // 6=>Hard offline

};
```

Generally, a node will only have the offline field set if it is in the configured state. However, if a node is told to transition from the configured state to another state while it is offline, the node_state may show both offline as set and a state other than configured.

Examples of node_state values when viewed as a byte field are shown below:

```
Unconfigured              0x02
Unconfig/App-less         0x03
Configured/online         0x04
Configured/hard-offline   0x06   /* Permanent offline      */
Configured/offline        0x0C   /* Non-reset retained offline */
                                 /* (note this is actually an  */
                                 /* encoding of the node state */
                                 /* of "configured" and the    */
                                 /* offline condition)     */
Configured                0x8C   /* Non-reset retained bypass- */
                                 /* mode offline Like config-  */
                                 /* ured/offline except that   */
                                 /* the application went off-  */
                                 /* line in bypass mode        */
```

— version: The version number reflects the version of the protocol implementation residing in the protocol processor and may be used by a network management tool for computing addresses to non-volatile memory data fields not supported by the standard NM address assignment/reporting commands. The version number is 1 to 127 for implementations containing no additional, application specific customization, and 128 to 255 for implementations with additional, application specific features. 255 is a special escape version that means more version number information is available via other commands;

— error_log: The error log contains the most recent error logged by the system. A value of 0 indicates no error. An error in the range 1..127 is an application error and unique to the application. An error in the range 128..255 is a system error. While the errors a given instance of the protocol can detect and report is implementation specific, the following are the currently allocated error numbers allocated for system errors.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

0x81	Bad Event	The application processor's scheduler has an event table. It contains encoded representations of the various when clauses in the program. If the scheduler encounters a illegal event code, this error is logged.
0x82	NV Length	If an incoming NV update has a length that differs from the length of the matching input NV in the application, this error is logged and the NV update is discarded.
0x83	NV Short	If an incoming NV update is missing the second byte of the selector, this error is logged.
0x84	EE Write Fail	After each write to EEPROM, the result is read back. If there is a mismatch, this error is logged.
0x85	Bad Address Type	If the type field in the address table is illegal, this error is logged.
0x88	Sync NV	When NVs are written to, a bit array is updated to indicate an update needs to be transmitted. In "post_events", this bit array is examined and a message is generated for each NV that is so marked and its bit is cleared. However, if no buffer is available, the application goes into preemption mode. If while in preemption mode, a sync NV whose bit is set is written to by the application, then this error is logged.
0x8A	Invalid Domain	An attempt is made to send a message using a domain table entry that has a length byte that does not yield 0 when AND'ed with 0xF8. The message is discarded.
0x8D	Invalid Address Index	An attempt was made to access the address table using an out of range index. This could occur when using any of the address table related NM commands, or when sending explicit messages or NVs using the address table.
0x93	Invalid NV Index	An attempt was made to access the NV config or NV alias table using an out of range index. This could occur when using any of the NV table related NM commands, or when formulating an NV update message.
0x9A	Illegal Transceiver Register Address	An attempt was made to access a special purpose mode transceiver register using an out of range index. This could occur when accessing a transceiver register using the "retrieve_xcvr_status" function.
0x9B	Transceiver Register Operation Failure	A timeout occurred waiting for the MAC layer to get a chance to query the special purpose mode transceiver.
0xA1	Self-Installation Semaphore	
0xA2	Read/Write Semaphore	
0xC0-DF	State Byte Semaphore	The error log byte is used in certain cases for saving state information across unexpected.

— model_number: The model number is the protocol processor model.

The model and version numbers together can be used to determine the exact firmware image in use by a node.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.8.3 Proxy Status

This command can be used to deliver a command to one or more target nodes via an agent node. The proxy command is sent to an agent along with a target address in the APDU. The agent node relays the command to the target and then relays the response back to the original requester. The agent repeats the message from the requestor verbatim except for the target address being removed. The proxy command can only be used to relay a status request or a query id (*unconfigured*) request. Although this command never itself requires authentication to be executed, if the original request is marked to be authenticated, the relayed request to the target shall also be so marked.

The original requester specifies the target address via data in the form of an address table entry in the APDU. The agent node uses this to determine the destination address and the retry/timeout values used during the transaction. There is one exception—the domain bit is ignored (the message is always relayed in the same domain in which it was received). Note that the retry/timeout values supplied in the target address should result in a transaction duration that is shorter than those used by the original requester. In general, this command works best if the agent and target are on the same channel. The data structures used are described in Annex A.46.

The response data received by the original requester shall be identical to that which would have been received as a result of a direct request to the target. Note that in the case of a query id (*unconfigured*) broadcast, a direct message could result in multiple responses, whereas a proxy command sent to a single agent with a broadcast target would result in only a single response to the original requester.

This command is disallowed if the agent receives the request on a flexible domain.

Finally, if the agent node is in the process of sending outgoing transactions, it may not be able to deliver the relayed request immediately. Depending on how long this is delayed, the response may not be received by the original requester in time, even after several retries. Therefore, a transaction failure for a proxy command is more likely than for other transactions; this should be taken into account when drawing conclusions from same. Also, in order for the proxy command to work, the agent node must have at least two application input buffers.

13.8.4 Clear Status

This command clears a subset of the information in the status response. The statistics information, the extended statistics information, the reset cause register and the error log are cleared by this command. A node that does a status request on a periodic basis may choose to use a clear command following each successful status response. The data structures used are described in Annex A.47.

13.8.5 Query Transceiver Status

This command retrieves the status register information from a transceiver. It fails if there is no transceiver on the node, or if communication with the transceiver fails. It returns seven registers' worth of data regardless of the number of registers that the transceiver actually supports. It is up to the controller to know how many registers are valid. The data structures used are described in Annex A.48.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex A
Reference Implementation
(Normative)

A.1 General

This annex contains pseudocode algorithms that define functions within the protocol and descriptions of the related data structures. These algorithms are extracted from a working reference implementation. To make a working implementation to verify the algorithms, it was necessary to define a boundary and an interface between the protocol processor and the physical layer as well as a boundary between the protocol application layer and the actual application. These interfaces are informative. Also, there are places in the reference implementation where a particular function could be accomplished in several ways. These are also noted as informative exceptions. Finally, the reference implementation was done on a Motorola MC68360 processor, so there are some machine dependent implementation details called out in the code.

A.2 Predictive CSMA Algorithm

```
/*********************************************************************
          File:         spm.c

       Version:         1.7

     Reference:         None

       Purpose:         MAC sublayer, comm_type = 2 using SPI port on 360.
                        This does not support continuous frame exchange.
                        This files also has functions to support
                        io buttons for testing purpose such as
                        manual service request, reset, io pin, and LEDs.

          Note:         None.


         To Do:         None.
*********************************************************************/

/* START INFORMATIVE - Direct Mode */
/* This implementation is for transceivers that work with the MAC
 * sublayer configured for comm_type = 2 only.
 * If the implementation needs to support transceivers with the MAC
 * sublayer configured for comm_type = 1,
 * this example serves only as a starting point for such an
 * implemenation. */
/* END INFORMATIVE - Direct Mode */

/*********************************************************************
Section: Includes
*********************************************************************/
#include <string.h>
#include <stdlib.h>

#include <cnp_1.h>
#include <node.h>
#include <link.h>
#include <physical.h>
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/*****************************************************************
Section: Constant Definitions
*****************************************************************/
/* Used to indicate how often to check status of i/o buttons */
#define PHYIO_CHECK_INTERVAL 0.1 /* In Seconds */

/* Macro sets bit B of 16 bit word X */
#define SET_BIT(B,X) (((0x0001U << (B)) | (X))

/* Macro Clears bit B of 16 bit word X */
#define CLEAR_BIT(B,X) (( ~(0x0001U << (B)) ) & (X))

/* Dual Port Ram Base on Arnewsh Board */
#define DPRB            0x01000000UL

/* SPI Parameter Ram Base */
#define SPIB            (DPRB + 0xD80UL)

/* SCC2 Parameter Ram Base */
#define SCC2B           (DPRB + 0xD00UL)

/* Register Base = DPRB + 4 k  */
#define REGB            (DPRB + 0x1000UL)

/* 16bit CPM Command Register */
#define CR              (REGB + 0x5C0UL)

/* 16 bit Serial DMA Config Register */
#define SDCR            (REGB + 0x51EUL)

/* 24(32)bit CPM Interrupt Config Reg.*/
#define CICR            (REGB + 0x540UL)

/* 32 bit CPM Interrupt Pending Reg. */
#define CIPR            (REGB + 0x544UL)

/* 32 bit CPM Interrupt Mask Reg. */
#define CIMR            (REGB + 0x548UL)

/* Clear CISR bit by writing a 1*/
/* 32 bit CPM Interrupt Service Register*/
#define CISR            (REGB + 0x54CUL)

/* 32 bit SI Clock Route */
#define SICR            (REGB + 0x6ECUL)

/* 16 bit SPI Mode Register */
#define SPMODE          (REGB + 0x6A0UL)

/* Clear SPIE bit by writing a 1*/
/* 8 bit SPI Event Register */
#define SPIE            (REGB + 0x6A6UL)

/* 8 bit SPI Mask Register */
#define SPIM            (REGB + 0x6AAUL)

/* 8 bit SPI Command Register */
#define SPCOM           (REGB + 0x6ADUL)

/* 32 bit SCC2 general mode register low */
#define GSMRL2          (REGB + 0x620UL)

/* 32 bit SCC2 general mode register high */
#define GSMRH2          (REGB + 0x624UL)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* 16 bit SCC2 protocol specific mode reg */
#define PSMR2           (REGB + 0x628UL)

/* 16 bit SCC2 transmit on demand */
#define TODR2           (REGB + 0x62CUL)

/* 16 bit SCC2 Data sync register */
#define DSR2            (REGB + 0x62EUL)

/* Clear SCCE bit by writing a 1 */
/* 16 bit SCC2 event register */
#define SCCE2           (REGB + 0x630UL)

/* 16 bit SCC2 mask register */
#define SCCM2           (REGB + 0x634UL)

/* 8 bit SCC2 status register */
#define SCCS2           (REGB + 0x637UL)

/* 32 bit Baud rate gen config register */
#define BRGC2           (REGB + 0x5F4UL)

/* 16 bit port A data direction reg */
#define PADIR           (REGB + 0x550UL)

/* 16 bit port A pin assignment reg */
#define PAPAR           (REGB + 0x552UL)

/* 16 bit port A open drain register */
#define PAODR           (REGB + 0x554UL)

/* 16 bit port A data register */
#define PADAT           (REGB + 0x558UL)

/* 32 bit port B direction register */
#define PBDIR           (REGB + 0x6B8UL)

/* 32 bit port B pin assignment register */
#define PBPAR           (REGB + 0x6BCUL)

/* 16 bit port B open drain register */
#define PBODR           (REGB + 0x6C2UL)

/* 32 bit port B data register */
#define PBDAT           (REGB + 0x6C4UL)

/* 16 bit port C direction register */
#define PCDIR           (REGB + 0x560UL)

/* 16 bit port C pin assignment reg */
#define PCPAR           (REGB + 0x562UL)

/* 16 bit port C special options */
#define PCSO            (REGB + 0x564UL)

/* 16 bit port C data register */
#define PCDAT           (REGB + 0x566UL)

/* 16 bit port C interrupt register */
#define PCINT           (REGB + 0x568UL)

/* Timer Related Constants */
/* 16 bit timer general config register */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
#define TGCR      (REGB + 0x580UL)

/* 16 bit timer reference  reg 1 */
/* or 32 bit cascaded 1 & 2      */
#define TRR1      (REGB + 0x594UL)

/* 16 bit timer counter reg 1 */
/* or 32 bit cascaded 1 & 2   */
#define TCN1      (REGB + 0x59CUL)


/* 16 bit timer mode register 4 */
#define TMR2      (REGB + 0x592UL)

/* 16 bit timer reference register 4 */
#define TRR2      (REGB + 0x596UL)

/* 16 bit timer capture register 4 */
#define TCR2      (REGB + 0x59AUL)

/* 16 bit timer counter register 4 */
#define TCN2      (REGB + 0x59EUL)

/* 16 bit timer event register 4 */
#define TER2      (REGB + 0x5B2UL)


/* ISR defines */
/* Interrupt vector base on Arnewsh Board*/
#define VBASE           0x00000000UL

/* User Interrupt vector num 3 MSbits */
#define UIVN_MSB        0x4U

/* SPI Int.vector num 5 LSBits from CPIC table*/
#define SPIVN_LSB       0x5U

/*****************************************************************
8 bit vector number = 3 msb set by user, 5 lsb from CPIC table shift
UIVN_MSB left 5 to bits 7 6 5 then OR with SPIVN_LSB to get vector
number vector address is 4 times vector number + vector base
*****************************************************************/
/* 32 bit SPI Int. vector address */
#define SPIV      ((((UIVN_MSB << 5) | SPIVN_LSB) * 4UL) + VBASE)


/* need an ISR for int error since int. error cannot be masked */
/* CPM int. error int. vector. num 5 LSB  */
#define CERRVN_LSB      0x0UL

/* cpm error iv */
#define CERRV  ((((UIVN_MSB << 5) | CERRVN_LSB) * 4UL) + VBASE)

/* spurious interrupt exception  vector */
#define SPURINTV 0x60U + VBASE

/* bus error  exception  vector */
#define BUSERRV 0x8U + VBASE

/* SPI Channel Number for CPM Command Register Commands */
#define SPI_CH_NUM      0x5U

/* SCC2 channel number for CPM Command Register Command */
#define SCC2_CH_NUM     0x4U
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* Opcode for CPM Com. Reg. to Init Tx Rx Param Ram */
#define INIT_TRP_OPCODE 0x0U

/* number of receive (transmit) BDs for SPI */
#define NUM_BD 0x1U

/* RBD_BASE and TBD_BASE must be divisible by 8 */
/* receive buffer descriptor base */
#define RBD_BASE (DPRB + 0x500UL)

/* put Transmit buffers right after Receive,  8bytes per BD */
/* transmit buffer descriptor base */
#define TBD_BASE (RBD_BASE + (NUM_BD * 0x8U))

/*****************************************************************
   FCR_INIT
        bits 7-5 unused,
        bit 4, Mot big endian = 1
        bits 3-0, function code dma = 8
*****************************************************************/
/* SPI Param Function Code Register Init */
#define FCR_INIT        0x18U

/* max length of receive buffers 2 bytes per buffer*/
#define MAX_BUF_LEN 0x2U

/* CP Interrupt Request Level */
#define CPIR_LEVEL 0x4U

#define NUM_BD_SCC2     0x1U

/* scc2 receive buffer descriptor. base */
#define RBD_BASE_SCC2 (DPRB + 0x440UL)

/* scc2 transmit bd base */
#define TBD_BASE_SCC2 (RBD_BASE_SCC2 + (NUM_BD_SCC2 * 8U))

/* max buffer length scc2 */
#define MAX_BL_SCC2     16U

/* length of history arrays */
#define NUM_HIST 512

/* time limit before xcvr resets */
#define RESET_COUNT_LIMIT 0xFFFF

/* number of ticks to delay during restart processing */
#define RESTART_DELAY_TICKS 165000UL

/*  Constants related to Channel Access Algorithm */
/* maximum size of backlog for access algorithm */
#define MAX_BACKLOG 63U

/* # of additional slots per backlog increment */
#define W_BASE 16U

/* used in conversion from this specification's
   reference time to clock ticks on the 360 */
#define NICS_TICKS_BASE 480UL

/* used in computation of SPI-SPM bit clock */
#define BIT_CLOCK_BASE 312500UL
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* used in computation of bit clock */
#define RATIO_BASE 3UL

/*****************************************************************
Implementation specific timing adjustments to channel access algorithm
This is where one would add or subtract from the respective times to
account for differences between the implementation specific delays and
latencies and those in this specification.
*****************************************************************/
/* adjustment to the beta1 slot time */
#define BETA1_ADJUST_TICKS 0

/* adjustment to the beta2 slot time */
#define BETA2_ADJUST_TICKS 0

/* adjustment to the cycle timer time */
#define CYCLE_ADJUST_TICKS 0

/* adjustment to the begin of packet tx */
#define WAIT_TX_ADJUST_TICKS 0

/*****************************************************************
Section: Type Definitions
*****************************************************************/
/* 68360 communications processor register definitions */

typedef struct          /* CP Command Register 16 bit */
{
   unsigned reset  : 1; /* reset CPM */
   unsigned        : 3; /* unused */
   unsigned opcode : 4; /* opcode for command */
   unsigned chnum  : 4; /* channel number selects which port
                           SPI, SCC etc */
   unsigned        : 3; /* unused */
   unsigned flag   : 1; /* flag, command executes when set,
                           cp clears when done */
} CPCommandReg;

typedef struct          /* Serial DMA Config Register 16 bit */
{
   unsigned      : 1;   /* unused */
   unsigned frz  : 2;   /* freeze */
   unsigned      : 2;   /* unused */
   unsigned sism : 3;   /* SDMA interrupt service mask */
   unsigned      : 1;   /* unused */
   unsigned said : 3;   /* SDMA arbitration ID */
   unsigned      : 2;   /* unused */
   unsigned inte : 1;   /* interrupt error */
   unsigned intb : 1;   /* interrupt break point */
} SDMAConfigReg;

typedef struct  /* CPM interrupt configuration register 32 bit */
{
   unsigned             : 8; /* unused */
   unsigned        scdp : 2; /* scc port for priority slot d */
   unsigned        sccp : 2; /* scc port for priority slot c */
   unsigned        scbp : 2; /* scc port for priority slot b */
   unsigned        scap : 2; /* scc port for priority slot a */
   unsigned        irl  : 3; /* interrupt request level */
   unsigned        hpi  : 5; /* highest priority interrupt */
   unsigned        vba  : 3; /* vector base address number offset */
   unsigned             : 4; /* reserved */
   unsigned        sps  : 1; /* spread priority scheme */
} CIConfigReg;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* CPM interrupt source 32 bit, same type for CIPR CIMR CISR */
typedef struct
{ /* names correspond to sources */
   unsigned pc0    : 1;           /* bit 31 */
   unsigned scc1   : 1;
   unsigned scc2   : 1;
   unsigned scc3   : 1;
   unsigned scc4   : 1;
   unsigned pc1    : 1;
   unsigned timer1 : 1;
   unsigned pc2    : 1;           /* bit 24  */
   unsigned pc3    : 1;           /* bit 23 */
   unsigned sdma   : 1;
   unsigned idma1  : 1;
   unsigned idma2  : 1;
   unsigned        : 1;           /* unused */
   unsigned timer2 : 1;
   unsigned rtt    : 1;
   unsigned        : 1;           /* bit 16 unused */
   unsigned pc4    : 1;           /* bit 15 */
   unsigned pc5    : 1;
   unsigned        : 1;           /* unused */
   unsigned timer3 : 1;
   unsigned pc6    : 1;
   unsigned pc7    : 1;
   unsigned pc8    : 1;
   unsigned        : 1;           /* bit 8 unused */
   unsigned timer4 : 1;           /* bit 7 */
   unsigned pc9    : 1;
   unsigned spi    : 1;           /* bit 5 */
   unsigned smc1   : 1;
   unsigned smc2   : 1;           /* also pip */
   unsigned pc10   : 1;
   unsigned pc11   : 1;
   unsigned        : 1;           /* bit 0 unused */
} CISourceReg;

/* serial interface clock route register 32 bit*/
typedef struct
{
   unsigned gr4   : 1;    /* grant support for scc4*/
   unsigned sc4   : 1;    /* connection for scc4*/
   unsigned r4cs  : 3;    /* receive clock source for scc4*/
   unsigned t4cs  : 3;    /* transmit clock source for scc4*/
   unsigned gr3   : 1;    /* grant support for scc3 */
   unsigned sc3   : 1;    /* connection for scc3 */
   unsigned r3cs  : 3;    /* receive clock source for scc3 */
   unsigned t3cs  : 3;    /* transmit clock source for scc3 */
   unsigned gr2   : 1;    /* grant support for scc2 */
   unsigned sc2   : 1;    /* connection for scc2 */
   unsigned r2cs  : 3;    /* receive clock source for scc2 */
   unsigned t2cs  : 3;    /* transmit clock source for scc2 */
   unsigned gr1   : 1;    /* grant support for scc1 */
   unsigned sc1   : 1;    /* connection for scc1 */
   unsigned r1cs  : 3;    /* receive clock source for scc1 */
   unsigned t1cs  : 3;    /* transmit clock source for scc1 */
} SIClockRouteReg;

typedef struct            /* SPI command register 8 bit*/
{
   unsigned str : 1;      /* start transmit and receive */
   unsigned res : 7;      /* reserved write with zeros */
} SPICommandReg;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
typedef struct            /* SPI mode register 16 bit */
{
   unsigned       : 1;    /* unused */
   unsigned loop  : 1;    /* local loop back */
   unsigned ci    : 1;    /* clock invert */
   unsigned cp    : 1;    /* clock phase */
   unsigned div16 : 1;    /* divide brgclk by 16 for spiclk */
   unsigned rev   : 1;    /* reverse data from big endian to
                             little endian */
   unsigned ms    : 1;    /* master slave */
   unsigned en    : 1;    /* enable SPI */
   unsigned len   : 4;    /* number of bits per char = len + 1 */
   unsigned pms   : 4;    /* prescale modulus select, divide
                             down clock */
} SPIModeReg;

typedef struct  /* SPI event and mask registers 8 bit */
{
   unsigned     : 2;   /* unused */
   unsigned mme : 1;   /* multi-master error */
   unsigned txe : 1;   /* transmit error */
   unsigned     : 1;   /* unused */
   unsigned bsy : 1;   /* busy error */
   unsigned txb : 1;   /* transmit buffer complete */
   unsigned rxb : 1;   /* receive buffer complete */
} SPIEventMaskReg;

typedef struct          /* SPI Parameter RAM  40 bytes*/
{
   uint16 rbase;  /* base address of receive BD's, set by user */
   uint16 tbase;  /* base address of transmit BD's, set by user */
   uint8  rfcr;   /* receive function code, set by user  */
   uint8  tfcr;   /* transmit function code, set by user  */
   uint16 mrblr;  /* maximum receive buffer length register,
                     set by user  */
   uint32 rstate; /* rx internal state    */
   uint32 ridp;   /* rx internal data ptr */
   uint16 rbptr;  /* rx BD ptr */
   uint16 ribc;   /* rx internal byte count */
   uint32 rtemp;  /* rx temp */
   uint32 tstate; /* tx internal state    */
   uint32 tidp;   /* tx internal data ptr */
   uint16 tbptr;  /* tx BD ptr */
   uint16 tibc;   /* tx internal byte count */
   uint32 ttemp;  /* tx temp */
} SPIParamRam;


typedef struct           /* Receive Buffer Descriptor 8 bytes */
{
   unsigned e  : 1;      /* empty buffer ready to receive*/
   unsigned    : 1;      /* unused */
   unsigned w  : 1;      /* wrap final bd so wrap to top */
   unsigned i  : 1;      /* interrupt, set rxb bit in spie register
                            when filled */
   unsigned l  : 1;      /* last, if slave then buffer has last bit */
   unsigned    : 1;      /* unused */
   unsigned cm : 1;      /* continuous mode buffer can be overwritten
                            e always 1 */
   unsigned    : 7;      /* unused */
   unsigned ov : 1;      /* receiver overrun only when slave */
   unsigned me : 1;      /* multiple master error */
   uint16   dataLen;     /* data length = number of bytes written
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                           into buffer */
   SPMRxFrame * dataPtr; /* ptr to buffer of data */
} RBufferDesc;

typedef struct          /* Transmit Buffer Descriptor 8 bytes */
{
   unsigned r  : 1;   /* ready to transmit buffer */
   unsigned    : 1;   /* unused */
   unsigned w  : 1;   /* wrap final bd so wrap to top */
   unsigned i  : 1;   /* interrupt, set txb bit in spie register
                         when sent */
   unsigned l  : 1;   /* last, if slave then buffer has last bit */
   unsigned    : 1;   /* unused */
   unsigned cm : 1;   /* continuous mode buffer can be re-sent
                         r always 1 */
   unsigned    : 7;   /* unused */
   unsigned un : 1;   /* transmit underrun only when slave */
   unsigned me : 1;   /* multiple master error */
   uint16   dataLen;  /* data length = number of bytes to send
                        from this buffer */
   SPMTxFrame * dataPtr;   /* ptr to buffer of data */
} TBufferDesc;

/* scc structs */

typedef struct             /* general scc mode register  high 32 bit */
{
   unsigned      : 15;     /* unused */
   unsigned gde  : 1;      /* glitch detect enable */
   unsigned tcrc : 2;      /* transparent crc */
   unsigned revd : 1;      /* reverse data */
   unsigned trx  : 1;      /* transparent receiver */
   unsigned ttx  : 1;      /* transparent transmitter */
   unsigned cdp  : 1;      /* cd pulse */
   unsigned ctsp : 1;      /* cts pulse */
   unsigned cds  : 1;      /* cd sampling */
   unsigned ctss : 1;      /* cts sampling */
   unsigned tfl  : 1;      /* transmit fifo length */
   unsigned rfw  : 1;      /* receive fifl width */
   unsigned txsy : 1;      /* transmitter synchronized to receiver */
   unsigned synl : 2;      /* sync length */
   unsigned rtsm : 1;      /* rts mode */
   unsigned rsyn : 1;      /* receive sync timing */
} GSMRegHigh;


typedef struct             /* general scc mode register  low 32 bit */
{
   unsigned      : 1;      /* unused */
   unsigned edge : 2;      /* DPLL clock edge */
   unsigned tci  : 1;      /* transmit clock invert */
   unsigned tsnc : 2;      /* transmit sense bits */
   unsigned rinv : 1;      /* dpll receive invert data*/
   unsigned tinv : 1;      /* dpll transmit invert data */
   unsigned tpl  : 3;      /* transmit preamble length */
   unsigned tpp  : 2;      /* transmit preamble pattern */
   unsigned tend : 1;      /* transmit frame ending */
   unsigned tdcr : 2;      /* transmit divide clock rate */
   unsigned rdcr : 2;      /* receive dpll clock rate */
   unsigned renc : 3;      /* receiver decoding method */
   unsigned tenc : 3;      /* transmitter encoding method */
   unsigned diag : 2;      /* diagnostic mode */
   unsigned enr  : 1;      /* enable receive */
   unsigned ent  : 1;      /* enable transmit */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   unsigned mode : 4;      /* channel protocol mode */
} GSMRegLow;

typedef struct           /* SCC event/mask register 16 bit */
{
   unsigned     : 3;     /* unused */
   unsigned glr : 1;     /* glitch on receive */
   unsigned glt : 1;     /* glitch on tranmit */
   unsigned dcc : 1;     /* dpll carrier sense changed */
   unsigned     : 2;     /* unused */
   unsigned gra : 1;     /* graceful stop complete */
   unsigned     : 2;     /* unused */
   unsigned txe : 1;     /* transmit error */
   unsigned rch : 1;     /* receive character or long word */
   unsigned bsy : 1;     /* busy condition */
   unsigned tx  : 1;     /* buffer transmitted */
   unsigned rx  : 1;     /* buffer received */
} SCCEventMaskReg;

typedef struct          /* scc status register 8 bit */
{
   unsigned    : 6;  /* unused */
   unsigned cs : 1;  /* carrier sense */
   unsigned    : 1;  /* unused */
} SCCStatusReg;

typedef struct    /* SCC Parameter RAM Transparent mode 56 bytes */
{
   uint16 rbase;  /* base address of receive BD's, set by user */
   uint16 tbase;  /* base address of transmit BD's, set by user */
   uint8  rfcr;   /* receive function code, set by user  */
   uint8  tfcr;   /* transmit function code, set by user  */
   uint16 mrblr;  /* maximum receive buffer length register,
                     set by user  */
   uint32 rstate; /* rx internal state    */
   uint32 ridp;   /* rx internal data ptr */
   uint16 rbptr;  /* rx BD ptr */
   uint16 ribc;   /* rx internal byte count */
   uint32 rtemp;  /* rx temp */
   uint32 tstate; /* tx internal state    */
   uint32 tidp;   /* tx internal data ptr */
   uint16 tbptr;  /* tx BD ptr */
   uint16 tibc;   /* tx internal byte count */
   uint32 ttemp;  /* tx temp */
   uint32 rcrc;   /* temp receive crc */
   uint32 tcrc;   /* temp transmit crc */
   uint32 crcp;   /* crc preset for transparent mode */
   uint32 crcc;   /* crc constant for transparent mode */
} SCCParamRam;

typedef struct    /* SCC Receive Buffer Descriptor 8 bytes */
{
   unsigned e  : 1; /* empty buffer ready to receive*/
   unsigned    : 1; /* unused */
   unsigned w  : 1; /* wrap final bd so wrap to top */
   unsigned i  : 1; /* interrupt, set rxb bit in scce register
                       when filled */
   unsigned l  : 1; /* this buffer last in frame */
   unsigned f  : 1; /* first in frame */
   unsigned cm : 1; /* continuous mode buffer can be overwritten
                       e always 1 */
   unsigned    : 1; /* unused */
   unsigned de : 1; /* dpll error */
   unsigned    : 2; /* unused */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   unsigned no : 1; /* non octet error */
   unsigned    : 1; /* unused */
   unsigned cr : 1; /* crc error */
   unsigned ov : 1; /* receiver overrun  */
   unsigned cd : 1; /* carrier detect lost */
   uint16   dataLen; /* data length = number of bytes written
                        into buffer */
   Byte   * dataPtr; /* ptr to buffer of data */
} SCCReceiveBD;

typedef struct   /* SCC Transmit Buffer Descriptor 8 bytes */
{
   unsigned r  : 1; /* ready to transmit buffer */
   unsigned    : 1; /* unused */
   unsigned w  : 1; /* wrap final bd so wrap to top */
   unsigned i  : 1; /* interrupt, set txb bit in scce register
                       when sent */
   unsigned l  : 1; /* last byte in frame in this buffer */
   unsigned tc : 1; /* transmit crc */
   unsigned cm : 1; /* continuous mode buffer can be re-sent r always 1 */
   unsigned    : 7; /* unused */
   unsigned un : 1; /* transmit underrun */
   unsigned ct : 1; /* cts lost during frame transmission */
   uint16   dataLen; /* data length = number of bytes to send
                        from this buffer */
   Byte   * dataPtr; /* ptr to buffer of data */
} SCCTransmitBD;

typedef struct  /* BRGC baud rate generator config register 32 bit */
{
   unsigned       : 14;    /* unused */
   unsigned rst   : 1;     /* reset brg */
   unsigned en    : 1;     /* enable brg count */
   unsigned extc  : 2;     /* enable external clock source */
   unsigned atb   : 1;     /* autobaud */
   unsigned cd    : 12;    /* clock divider */
   unsigned div16 : 1;     /* div 16 clock */
} BRGConfigReg;

typedef struct    /* port a direction register 16 bit */
{
   unsigned dr15 : 1;      /* pin direction 0 = input 1 = output */
   unsigned dr14 : 1;
   unsigned dr13 : 1;
   unsigned dr12 : 1;
   unsigned dr11 : 1;
   unsigned dr10 : 1;
   unsigned dr9  : 1;
   unsigned dr8  : 1;
   unsigned dr7  : 1;
   unsigned dr6  : 1;
   unsigned dr5  : 1;
   unsigned dr4  : 1;
   unsigned dr3  : 1;
   unsigned dr2  : 1;
   unsigned dr1  : 1;
   unsigned dr0  : 1;
} PADirectionReg;

typedef struct          /* port a pin assignment register 16 bit */
{
   unsigned dd15 : 1;      /* 0 = general purpose io */
   unsigned dd14 : 1;
   unsigned dd13 : 1;
```

```
   unsigned dd12 : 1;
   unsigned dd11 : 1;
   unsigned dd10 : 1;
   unsigned dd9  : 1;
   unsigned dd8  : 1;
   unsigned dd7  : 1;
   unsigned dd6  : 1;
   unsigned dd5  : 1;
   unsigned dd4  : 1;
   unsigned dd3  : 1;
   unsigned dd2  : 1;
   unsigned dd1  : 1;
   unsigned dd0  : 1;
} PAPinAssignmentReg;

typedef struct          /* port a open drain register 16 bit */
{
   unsigned      : 8;   /* unused */
   unsigned od7  : 1;   /* 0 = active drive 1 = open drain */
   unsigned od6  : 1;
   unsigned od5  : 1;
   unsigned od4  : 1;
   unsigned od3  : 1;
   unsigned      : 1;    /* unused */
   unsigned od1  : 1;
   unsigned      : 1;    /* unused */
} PAOpenDrainReg;

typedef struct           /* port a data register 16 bit */
{
   unsigned d15 : 1;       /* value of pin */
   unsigned d14 : 1;
   unsigned d13 : 1;
   unsigned d12 : 1;
   unsigned d11 : 1;
   unsigned d10 : 1;
   unsigned d9  : 1;
   unsigned d8  : 1;
   unsigned d7  : 1;
   unsigned d6  : 1;
   unsigned d5  : 1;
   unsigned d4  : 1;
   unsigned d3  : 1;
   unsigned d2  : 1;
   unsigned d1  : 1;
   unsigned d0  : 1;
} PADataReg;

typedef struct          /* PBDIR port b direction register 32 bit */
{
   unsigned      : 14;            /* unused*/
   unsigned dr17 : 1;             /* pin with given # */
   unsigned dr16 : 1;             /* pin with same # */
   unsigned dr15 : 1;             /* pin with same # */
   unsigned dr14 : 1;             /* pin with same # */
   unsigned dr13 : 1;             /* pin with same # */
   unsigned dr12 : 1;             /* pin with same # */
   unsigned dr11 : 1;             /* pin with same # */
   unsigned dr10 : 1;             /* pin with same # */
   unsigned dr9  : 1;             /* pin with same # */
   unsigned dr8  : 1;             /* pin with same # */
   unsigned dr7  : 1;             /* pin with same # */
   unsigned dr6  : 1;             /* pin with same # */
   unsigned dr5  : 1;             /* pin with same # */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   unsigned dr4  : 1;              /* pin with same # */
   unsigned dr3  : 1;              /* pin with same # */
   unsigned dr2  : 1;              /* pin with same # */
   unsigned dr1  : 1;              /* pin with same # */
   unsigned dr0  : 1;              /* pin with same # */
} PBDirectionReg;

typedef struct  /* PBPAR port b pin assignment register 32 bit */
{
   unsigned      : 14;             /* unused*/
   unsigned dd17 : 1;              /* pin with given # */
   unsigned dd16 : 1;              /* pin with same # */
   unsigned dd15 : 1;              /* pin with same # */
   unsigned dd14 : 1;              /* pin with same # */
   unsigned dd13 : 1;              /* pin with same # */
   unsigned dd12 : 1;              /* pin with same # */
   unsigned dd11 : 1;              /* pin with same # */
   unsigned dd10 : 1;              /* pin with same # */
   unsigned dd9  : 1;              /* pin with same # */
   unsigned dd8  : 1;              /* pin with same # */
   unsigned dd7  : 1;              /* pin with same # */
   unsigned dd6  : 1;              /* pin with same # */
   unsigned dd5  : 1;              /* pin with same # */
   unsigned dd4  : 1;              /* pin with same # */
   unsigned dd3  : 1;              /* pin with same # */
   unsigned dd2  : 1;              /* pin with same # */
   unsigned dd1  : 1;              /* pin with same # */
   unsigned dd0  : 1;              /* pin with same # */
} PBPinAssignmentReg;

typedef struct          /* PBODR port b open drain register 16 bit */
{
   unsigned od15 : 1;              /* pin with same # */
   unsigned od14 : 1;              /* pin with same # */
   unsigned od13 : 1;              /* pin with same # */
   unsigned od12 : 1;              /* pin with same # */
   unsigned od11 : 1;              /* pin with same # */
   unsigned od10 : 1;              /* pin with same # */
   unsigned od9  : 1;              /* pin with same # */
   unsigned od8  : 1;              /* pin with same # */
   unsigned od7  : 1;              /* pin with same # */
   unsigned od6  : 1;              /* pin with same # */
   unsigned od5  : 1;              /* pin with same # */
   unsigned od4  : 1;              /* pin with same # */
   unsigned od3  : 1;              /* pin with same # */
   unsigned od2  : 1;              /* pin with same # */
   unsigned od1  : 1;              /* pin with same # */
   unsigned od0  : 1;              /* pin with same # */
} PBOpenDrainReg;

typedef struct      /* PBDAT port b data register 32 bit */
{
   unsigned      : 14;             /* unused*/
   unsigned d17 : 1;               /* pin with given # */
   unsigned d16 : 1;               /* pin with same # */
   unsigned d15 : 1;               /* pin with same # */
   unsigned d14 : 1;               /* pin with same # */
   unsigned d13 : 1;               /* pin with same # */
   unsigned d12 : 1;               /* pin with same # */
   unsigned d11 : 1;               /* pin with same # */
   unsigned d10 : 1;               /* pin with same # */
   unsigned d9  : 1;               /* pin with same # */
   unsigned d8  : 1;               /* pin with same # */
   unsigned d7  : 1;               /* pin with same # */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   unsigned d6  : 1;              /* pin with same # */
   unsigned d5  : 1;              /* pin with same # */
   unsigned d4  : 1;              /* pin with same # */
   unsigned d3  : 1;              /* pin with same # */
   unsigned d2  : 1;              /* pin with same # */
   unsigned d1  : 1;              /* pin with same # */
   unsigned d0  : 1;              /* pin with same # */
} PBDataReg;

typedef struct          /* port c direction register 16 bit */
{
   unsigned      : 4;   /* unused */
   unsigned dr11 : 1;   /* pin direction 0 = input 1 = output */
   unsigned dr10 : 1;
   unsigned dr9  : 1;
   unsigned dr8  : 1;
   unsigned dr7  : 1;
   unsigned dr6  : 1;
   unsigned dr5  : 1;
   unsigned dr4  : 1;
   unsigned dr3  : 1;
   unsigned dr2  : 1;
   unsigned dr1  : 1;
   unsigned dr0  : 1;
} PCDirectionReg;

typedef struct          /* port c pin assignment register 16 bit */
{
   unsigned      : 4;   /* unused */
   unsigned dd11 : 1;   /* 0 = general purpose io */
   unsigned dd10 : 1;
   unsigned dd9  : 1;
   unsigned dd8  : 1;
   unsigned dd7  : 1;
   unsigned dd6  : 1;
   unsigned dd5  : 1;
   unsigned dd4  : 1;
   unsigned dd3  : 1;
   unsigned dd2  : 1;
   unsigned dd1  : 1;
   unsigned dd0  : 1;
} PCPinAssignmentReg;

typedef struct          /* port c data register 16 bit */
{
   unsigned     : 4;   /* unused */
   unsigned d11 : 1;   /* value of pin */
   unsigned d10 : 1;
   unsigned d9  : 1;
   unsigned d8  : 1;
   unsigned d7  : 1;
   unsigned d6  : 1;
   unsigned d5  : 1;
   unsigned d4  : 1;
   unsigned d3  : 1;
   unsigned d2  : 1;
   unsigned d1  : 1;
   unsigned d0  : 1;
} PCDataReg;

typedef struct           /* port c interrupt control register 16 bit */
{
   unsigned       : 4;   /* unused */
   unsigned edm11 : 1;   /* edge detect mode for line*/
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   unsigned edm10 : 1;
   unsigned edm9  : 1;
   unsigned edm8  : 1;
   unsigned edm7  : 1;
   unsigned edm6  : 1;
   unsigned edm5  : 1;
   unsigned edm4  : 1;
   unsigned edm3  : 1;
   unsigned edm2  : 1;
   unsigned edm1  : 1;
   unsigned edm0  : 1;
} PCInterruptReg;

typedef struct          /* port c special options register 16 bit */
{
   unsigned      : 4;              /* unused */
   unsigned cd4  : 1;              /* Carrier detect */
   unsigned cts4 : 1;              /* clear to send */
   unsigned cd3  : 1;              /* Carrier detect */
   unsigned cts3 : 1;              /* clear to send */
   unsigned cd2  : 1;              /* Carrier detect */
   unsigned cts2 : 1;              /* clear to send */
   unsigned cd1  : 1;              /* Carrier detect */
   unsigned cts1 : 1;              /* clear to send */
   unsigned      : 4;              /* unused */
} PCSpecialOptionsReg;

/* Timer related TypeDefs */

typedef struct    /* timer general config register  16 bit */
{
   unsigned cas4 : 1;   /* cascade timer 4 */
   unsigned frz4 : 1;   /* freeze timer 4 */
   unsigned stp4 : 1;   /* stop timer 4 */
   unsigned rst4 : 1;   /* reset timer 4 */
   unsigned gm2  : 1;   /* gate mode pin 2 */
   unsigned frz3 : 1;   /* freeze timer 3 */
   unsigned stp3 : 1;   /* stop timer 3 */
   unsigned rst3 : 1;   /* reset timer 3 */
   unsigned cas2 : 1;   /* cascade timer 2 */
   unsigned frz2 : 1;   /* freeze timer 2 */
   unsigned stp2 : 1;   /* stop timer  2 */
   unsigned rst2 : 1;   /* reset timer 2 */
   unsigned gm1  : 1;   /* gate mode pin 1 */
   unsigned frz1 : 1;   /* freeze timer 1 */
   unsigned stp1 : 1;   /* stop timer 1 */
   unsigned rst1 : 1;   /* reset timer 1 */
} TimerGenConfigReg;

typedef struct    /*  timer mode register 16 bit */
{
   unsigned ps   : 8;    /* prescaler */
   unsigned ce   : 2;    /* capture edge */
   unsigned om   : 1;    /* output mode */
   unsigned ori  : 1;    /* output ref interrupt enable */
   unsigned frr  : 1;    /* free run/restart */
   unsigned iclk : 2;    /* input clock source */
   unsigned ge   : 1;    /* gate enable */
} TimerModeReg;

typedef struct    /* timer event register 16 bit */
{
   unsigned     : 14;   /* unused */
   unsigned ref : 1;    /* output reference event */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   unsigned cap : 1;    /* input capture event */
} TimerEventReg;


/* structure used for byte by byte crc checking on receive */

typedef struct
{
   uint16 poly;
   uint16 crc;
   uint8 crcBit;
   uint8 dataBit;
   uint8 dataByte;
} CRCParam;

/* structures used for debugging purposes */

typedef struct  /* contains record of SPM frame */
{
   SPMState state;   /* state when received status byte */
   SPMRxFrame rf;    /* RX frame */
   SPMTxFrame tf;    /* TX frames */
   uint32 duration;  /* time to run isr */
   uint32 start;     /* start time of isr */
   uint16 rb;        /* index to receive buffer */
   uint16 tb;        /* index to transmit bufer */
} Record;


typedef struct /* contains history of frame records */
{
   int index; /* current record wraps around when full */
   Record records[NUM_HIST];  /* array of records */
} History;

/*****************************************************************
Section: Local Globals
*****************************************************************/
/* pointers to SPI registers  all are constant pointers to volatiles
   volatile is needed to insure mem access occurs at each reference */

static volatile CPCommandReg  * const crPtrGbl =
           (volatile CPCommandReg *) CR;
static volatile SDMAConfigReg  * const sdcrPtrGbl =
           (volatile SDMAConfigReg *) SDCR;

static volatile CIConfigReg  * const cicrPtrGbl =
           (volatile CIConfigReg *) CICR;
static volatile CISourceReg  * const ciprPtrGbl =
           (volatile CISourceReg *) CIPR;
static volatile CISourceReg  * const cimrPtrGbl =
           (volatile CISourceReg *) CIMR;
static volatile CISourceReg  * const cisrPtrGbl =
           (volatile CISourceReg *) CISR;

static volatile SIClockRouteReg  * const sicrPtrGbl =
           (volatile SIClockRouteReg *) SICR;

static volatile SPIModeReg * const spmodePtrGbl =
           (volatile SPIModeReg *) SPMODE;
static volatile SPIEventMaskReg  * const spiePtrGbl =
           (volatile SPIEventMaskReg *) SPIE;

static volatile SPIEventMaskReg  * const spimPtrGbl =
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
          (volatile SPIEventMaskReg *) SPIM;
static volatile SPICommandReg  * const spcomPtrGbl =
          (volatile SPICommandReg *) SPCOM;

static volatile SPIParamRam  * const spiParamPtrGbl =
          (volatile SPIParamRam *) SPIB;
static volatile RBufferDesc  * const rbdPtrGbl =
          (volatile RBufferDesc *) RBD_BASE;
static volatile TBufferDesc  * const tbdPtrGbl =
          (volatile TBufferDesc *) TBD_BASE;


/* SCC2 registers */
static volatile GSMRegLow  * const gsmrl2PtrGbl =
          (volatile GSMRegLow *) GSMRL2;
static volatile GSMRegHigh  * const gsmrh2PtrGbl =
          (volatile GSMRegHigh *) GSMRH2;
static volatile uint16 * const dsr2PtrGbl =
          (volatile uint16 *) DSR2;
static volatile uint16 * const todr2PtrGbl =
          (volatile uint16 *) TODR2;
static volatile SCCEventMaskReg  * const scce2PtrGbl =
          (volatile SCCEventMaskReg *) SCCE2;
static volatile SCCEventMaskReg  * const sccm2PtrGbl =
          (volatile SCCEventMaskReg *) SCCM2;
static volatile SCCStatusReg  * const sccs2PtrGbl =
          (volatile SCCStatusReg *) SCCS2;
static volatile BRGConfigReg  * const brgc2PtrGbl =
          (volatile BRGConfigReg *) BRGC2;

static volatile SCCParamRam  * const scc2ParamPtrGbl =
          (volatile SCCParamRam *) SCC2B;
static volatile SCCReceiveBD  * const scc2rbdPtrGbl =
          (volatile SCCReceiveBD *) RBD_BASE_SCC2;
static volatile SCCTransmitBD  * const scc2tbdPtrGbl =
          (volatile SCCTransmitBD *) TBD_BASE_SCC2;

/* allocate storage for scc transmit and receive buffers,
   init to zeros */
static volatile Byte scc2rBufGbl[NUM_BD_SCC2][MAX_BL_SCC2] = {0};
static volatile Byte scc2tBufGbl[NUM_BD_SCC2][MAX_BL_SCC2] = {0};

/* Port A registers */
static volatile PADirectionReg  * const padirPtrGbl =
          (volatile PADirectionReg *) PADIR;
static volatile PAPinAssignmentReg  * const paparPtrGbl =
          (volatile PAPinAssignmentReg *) PAPAR;
static volatile PAOpenDrainReg  * const paodrPtrGbl =
          (volatile PAOpenDrainReg *) PAODR;
static volatile PADataReg  * const padatPtrGbl =
          (volatile PADataReg *) PADAT;

/* Port B registers */
static volatile PBDirectionReg  * const pbdirPtrGbl =
          (volatile PBDirectionReg *) PBDIR;
static volatile PBPinAssignmentReg  * const pbparPtrGbl =
          (volatile PBPinAssignmentReg *) PBPAR;
static volatile PBOpenDrainReg  * const pbodrPtrGbl =
          (volatile PBOpenDrainReg *) PBODR;
static volatile PBDataReg  * const pbdatPtrGbl =
          (volatile PBDataReg *) PBDAT;
/* Port C registers */
static volatile PCDirectionReg  * const pcdirPtrGbl =
          (volatile PCDirectionReg *) PCDIR;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
static volatile PCPinAssignmentReg  * const pcparPtrGbl =
          (volatile PCPinAssignmentReg *) PCPAR;
static volatile PCDataReg  * const pcdatPtrGbl =
          (volatile PCDataReg *) PCDAT;
static volatile PCSpecialOptionsReg * const pcsoPtrGbl =
          (volatile PCSpecialOptionsReg *) PCSO;
static volatile PCInterruptReg * const pcintPtrGbl =
          (volatile PCInterruptReg *) PCINT;

/* const ptr to ptr to function returning void with void params */
static volatile void  (** const spivPtrGbl)(void) =
          (volatile void  (**)(void)) SPIV; /* ptr to vector */
static volatile void  (** const cerrvPtrGbl)(void) =
          (volatile void  (**)(void)) CERRV; /* ptr to vector */
static volatile void  (** const spurintvPtrGbl)(void) =
          (volatile void  (**)(void)) SPURINTV; /* ptr to vector */
static volatile void  (** const buserrvPtrGbl)(void) =
          (volatile void  (**)(void)) BUSERRV; /* ptr to vector */

/* Timer 1-2 stuff */
static volatile TimerGenConfigReg  * const tgcrPtrGbl =
          (volatile TimerGenConfigReg *) TGCR;

static volatile uint32  * const trr12PtrGbl =
          (volatile uint32 *) TRR1; /* cascaded 32 bit ref */
static volatile uint32  * const tcn12PtrGbl =
          (volatile uint32 *) TCN1; /* cascaded 32 bit timer */

static volatile TimerModeReg  * const tmr2PtrGbl =
          (volatile TimerModeReg *) TMR2;
static volatile uint16  * const trr2PtrGbl =
          (volatile uint16 *) TRR2;
static volatile uint16  * const tcr2PtrGbl =
          (volatile uint16 *) TCR2;
static volatile uint16  * const tcn2PtrGbl =
          (volatile uint16 *) TCN2;
static volatile TimerEventReg  * const ter2PtrGbl =
          (volatile TimerEventReg *) TER2;

/*********************************************************************
Section: Globals
*********************************************************************/
/* parameters for SPMIsr execution history for debugging purposes */
volatile MACParam macGbl;   /* extern in physical.h */
volatile SPMParam spmGbl;   /* extern in physical.h */
volatile CRCParam crcGbl;

volatile History hBufGbl = {0}; /* Initialize to zero */

/* exception count for debugging puposes. Incremented in exception
   service routines for bus error and spurious interrupt */
volatile uint32 exceptions = 0;

/* Timer for PHYIO */
MsTimer phyIOTimer;

/*-------------------------------------------------------------------
Section: Local Function Prototypes
-------------------------------------------------------------------*/
/* Special Purpose Mode Init */
static int16   SPMInit(void);

/* Special Purpose Mode Int. Service Routine */
static void    SPMIsr(void);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* CPM int error Int. Service Routive */
static void    CErrIsr(void);

/* Spurious interrupt error ISR */
static void    SpurIntIsr(void);

/* bus error ISR */
static void    BusErrIsr(void);

/* Initialize MAC hardware timer */
static uint32 * MACTimerInit(void);

/* delay function waits for delay ticks */
static Boolean DelayTicks(uint32 delay);

/* uses MACTimer hardware timer */
static void    UpdateElapsedTimer(TimerData32 * t);
static void    StartElapsedTimer(TimerData32 * t);
/* increments backlog */
static void    IncrementBacklog(uint8 deltaBacklog);
/* decrement backlog */
static void    DecrementBacklog(uint8 deltaBacklog);

/*-------------------------------------------------------------------
Section: Function Definitions
-------------------------------------------------------------------*/
/* #define SPM_TEST */    /* To include a main to test SPM only */
/* #define SPM_HISTORY */ /* To debug without break points */


/*****************************************************************
Function:  main
Returns:
Reference:
Purpose:   To test mac layer only
Comments:
*****************************************************************/

#ifdef SPM_TEST
void main()
{
   long count = 0;

   PHYInitSPM();


   macGbl.tl = 8;  /* 5 byte packet */
   macGbl.tc = 0;
   macGbl.tPkt[0] = 1;
   macGbl.tPkt[1] = 2;
   macGbl.tPkt[2] = 3;
   macGbl.tPkt[3] = 4;
   macGbl.tPkt[4] = 5;
   macGbl.tPkt[5] = 6;
   macGbl.tPkt[6] = 7;
   macGbl.tPkt[7] = 8;
   macGbl.tpr = TRUE;

   count = 0;
   while (macGbl.tpr != FALSE )
   {
      count++;
   }
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   count = 0;
   while (spmGbl.mode != STOP )
   {
      count++;
   }
   return;
}

#endif

/*****************************************************************
Function:  PHYInitSPM
Returns:
Reference:
Purpose:   Set up special purpose mode
           SPMInit() initializes all the 68360 registers
           MACTimerInit() starts up a 32 bit hardware timer (25Mhz)
           PHYEnableSPMIsr() configures the Interrupt Service Routine
Comments:
*****************************************************************/
void PHYInitSPM(void)
{
   int16 initOK = 0;

   exceptions = 0; /* for debugging */

   initOK = SPMInit();

   /* initialize and start MAC hardware timer */
   spmGbl.clock = MACTimerInit();

   /* reconfigure SPI SPMODE */
   /* spmodePtrGbl->loop  = 1; */   /* local loop back for test */

   hBufGbl.index = 0;  /* start history at 0 */


   /* updates communications parameters
      brings out of reset and starts frame exchange
      and loads config regs
   */
   PHYEnableSPMIsr();

   return;
}

/*****************************************************************
Function:  SPMInit
Returns:   1 if ok, 0 if error.
Reference:
Purpose:    set up special purpose mode
            Basic approach: Use SPI to tx and rx frames from xcvr.
            The spi bit clock is wired externally to clk3 input
            This is used to clock the scc2 port that generates
            the frame clock.
Comments:
*****************************************************************/
static int16 SPMInit()
{
   int16 i;                      /* for loop counter */
   SDMAConfigReg sdcrTemp;       /* scratch to set up sdcr */
   CPCommandReg crTemp;          /* scratch to set up cr */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    SCCReceiveBD srbdTemp;         /* scratch */
    SCCTransmitBD stbdTemp;        /* scratch */
    SCCEventMaskReg scceTemp;      /* scratch */
    CIConfigReg cicrTemp;          /* scratch to set up cicr */
    SPIEventMaskReg spiemTemp;     /* scratch to set up spie or spim*/
    SPIModeReg spmodeTemp;         /* scratch to set up spmode */

    /* initialize registers and vectors*/

    /* initialize SDMA Config Register */
    sdcrTemp.frz = 0;       /* ignore freeze */
    sdcrTemp.sism = 7;      /* level 7 interrupt mask */
    sdcrTemp.said = 4;      /* level 4 bus arbitration priority */
    sdcrTemp.inte = 0;      /* no error interrupts */
    sdcrTemp.intb = 0;      /* no break interrupts */
    *sdcrPtrGbl = sdcrTemp; /* write out sdcr */


    /* set sicr */
    sicrPtrGbl->sc2 = 0;    /* scc2 in nmsi mode */
    sicrPtrGbl->r2cs = 6;   /* receive clock from clk3 */
    sicrPtrGbl->t2cs = 6;   /* transmit clock from clk3 */

    /*************************************************/
    /* initialize SCC2 Port for frame clock*/
    /*************************************************/
    /* initialize RBASE and TBASE in SCC2 Param RAM */
    scc2ParamPtrGbl->rbase = (uint16)  RBD_BASE_SCC2;
    scc2ParamPtrGbl->tbase = (uint16)  TBD_BASE_SCC2;

    /* initialize other parts of SCC2 parameter ram using CP command */
    crTemp.reset =0;                /* no cpm reset */
    crTemp.chnum = SCC2_CH_NUM;     /* SCC2 Channel */
    crTemp.opcode = INIT_TRP_OPCODE; /* init param ram */
    crTemp.flag = 1;                /* start command */
    *crPtrGbl = crTemp;             /* write to command register */

#ifndef SIMULATION
    /* wait for command to finish */
    while ( crPtrGbl->flag == 1)
    {
       ; /* Do nothing */
    }
#endif

    /* initialize function codes for param ram  and buffer length
       for scc2 */
    /* endian and function code */
    scc2ParamPtrGbl->rfcr = (uint8) FCR_INIT;
    /* endian and function code */
    scc2ParamPtrGbl->tfcr = (uint8) FCR_INIT;
    /*  max receive buffer length */
    scc2ParamPtrGbl->mrblr = (uint16) 2;

    /* initialize scc receive buffer descriptors */
    srbdTemp.e = 1;          /* 1 = buffer ready to receive */
    srbdTemp.w = 0;          /* 0 = don't wrap this bd */
    srbdTemp.i = 0;          /* 0 = disable interrupts */
    srbdTemp.l = 0;          /* 0 = not last */
    srbdTemp.cm = 1;         /* 1 = put in continuous mode */
    srbdTemp.de = 0;         /* clear out dpll error*/
    srbdTemp.no = 0;         /* clear out non octet error*/
    srbdTemp.cr = 0;         /* clear out crc error*/
    srbdTemp.ov = 0;         /* clear out receiver overrun error */
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
srbdTemp.cd = 0;         /* clear out carrier detect lost error */
srbdTemp.dataLen = 0;    /* superfluous  set by cp */
/* set to first buffer but change below */
srbdTemp.dataPtr = &scc2rBufGbl[0][0];

for (i = 0; i < NUM_BD_SCC2; i++) /* loop thru buf descriptors */
{
   scc2rbdPtrGbl[i] = srbdTemp; /* assign defaults from temp */
   /* assign ptr to buffer */
   scc2rbdPtrGbl[i].dataPtr = &scc2rBufGbl[i][0];
}
scc2rbdPtrGbl[NUM_BD_SCC2 - 1].w = 1; /* wrap last rbd */

/* initialize scc transmit buffer descriptors */
stbdTemp.r = 1;          /* 1 = buffer ready to transmit */
stbdTemp.w = 0;          /* 0 = don't wrap this bd */
stbdTemp.i = 0;          /* 0 = disable interrupts  */
stbdTemp.l = 0;          /* 0 = not last */
stbdTemp.tc = 0;         /* 0 = don't transmit crc */
stbdTemp.cm = 1;         /* 1 = put in continuous mode */
stbdTemp.un = 0;         /* clear out transmit underrunn error */
stbdTemp.ct = 0;         /* clear out cts lost error */
stbdTemp.dataLen = 2;    /* set to buffer length should be
                            2 bytes */
/* set to first buffer but change below */
stbdTemp.dataPtr = &scc2tBufGbl[0][0];

for (i = 0; i < NUM_BD_SCC2; i++) /* loop thru buf descriptors */
{
   scc2tbdPtrGbl[i] = stbdTemp; /* assign defaults from temp */
   /* assign ptr to buffer */
   scc2tbdPtrGbl[i].dataPtr = &scc2tBufGbl[i][0];
}
scc2tbdPtrGbl[NUM_BD_SCC2 - 1].w = 1; /* wrap last tbd */

/* initialize port A and C for SCC2 Transparent mode operation */
padirPtrGbl->dr2 = 0;   /* RXD for scc2 */
paparPtrGbl->dd2 = 1;   /* rxd connect internal */

paodrPtrGbl->od3 = 0;   /* not open drain for txd */
padirPtrGbl->dr3 = 0;   /* TXD for scc2 */
paparPtrGbl->dd3 = 1;   /* txd  connect internal */

padirPtrGbl->dr10 = 0;  /* clk3 sc2 */
paparPtrGbl->dd10 = 1;  /*  connect internal */
/* enable rts for scc2 */
pcdirPtrGbl->dr1 = 0;
pcparPtrGbl->dd1 = 1;
/* general purpose io so CD always asserted */
pcdirPtrGbl->dr7 = 1;
pcparPtrGbl->dd7 = 0;
pcsoPtrGbl->cd2 = 0;
/* use cts2 to initiate transmission. configure as gen purpose
   io so always asserted low */
pcdirPtrGbl->dr6 = 1;
pcparPtrGbl->dd6 = 0;
pcsoPtrGbl->cts2 = 0; /* configure as low always */

/* set gsmr */

gsmrh2PtrGbl->tcrc = 0; /* 16 bit ccitt crc */
gsmrh2PtrGbl->revd = 1; /* 1= send msbit of each byte out first */
gsmrh2PtrGbl->trx = 1;  /* receiver in transparent mode */
gsmrh2PtrGbl->ttx = 1;  /* transmitter in transparent mode */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    gsmrh2PtrGbl->cdp = 1;  /* cd pulse mode */
    gsmrh2PtrGbl->ctsp = 1; /* cts pulse mode */
    gsmrh2PtrGbl->cds = 1;  /* cd synchronous  mode */
    gsmrh2PtrGbl->ctss = 1; /* cts synchronous mode */
    gsmrh2PtrGbl->synl = 0; /* receive sync on cd not on
                               sync pattern */
    gsmrh2PtrGbl->tfl = 1;  /* 1 byte fifo for lower transmit
                               latency  */

    gsmrl2PtrGbl->diag = 1;  /* local loop back for test*/

    /* initialize transparent mode crc type */
    scc2ParamPtrGbl->crcp = 0x0000FFFFUL;  /* 16 bit CCITT CRC */
    scc2ParamPtrGbl->crcc = 0x0000F0B8UL;   /* 16 bit CCITT CRC */

    /* clear any previous scc2 interrupt events, write 1 to clear */
    scceTemp.glr = 1;
    scceTemp.glt = 1;
    scceTemp.dcc = 1;
    scceTemp.gra = 1;
    scceTemp.txe = 1;
    scceTemp.rch = 1;
    scceTemp.bsy = 1;
    scceTemp.tx  = 1;
    scceTemp.rx  = 1;
    *scce2PtrGbl = scceTemp;  /* write it out to SCCE2 */

    /* enable/disable interrupts write 1 to enable */
    scceTemp.glr = 0;
    scceTemp.glt = 0;
    scceTemp.dcc = 0;
    scceTemp.gra = 0;
    scceTemp.txe = 0;
    scceTemp.rch = 0;
    scceTemp.bsy = 0;
    scceTemp.tx  = 0;
    scceTemp.rx  = 0;
    *sccm2PtrGbl = scceTemp;  /* write it out to sccm2 */

    /* clear any old scc2 interrupts from CISR */
    /* clear by writing 1, spi interrupt in service */
    cisrPtrGbl->scc2 = 1;

    /* enable/disable system interrupt for spi in CIMR*/
    cimrPtrGbl->scc2 = 0;           /* enable by writing 1 */

    /*****************************************/
    /*   SPI Port Initialization */
    /*****************************************/

    /* initialize RBASE and TBASE in SPI Param RAM */
    /* start of Receive BDs */
    spiParamPtrGbl->rbase = (uint16) RBD_BASE;
    /* start of transmit BDs */
    spiParamPtrGbl->tbase = (uint16) TBD_BASE;

    /* initialize other parts of SPI parameter ram using CP command */
    crTemp.reset =0; /* no cpm reset */
    crTemp.chnum = SPI_CH_NUM; /* SPI Channel */
    crTemp.opcode = INIT_TRP_OPCODE; /* init param ram */
    crTemp.flag = 1; /* start command */
    *crPtrGbl = crTemp; /* write to command register */

#ifndef  SIMULATION
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* wait for command to finish */
   while ( crPtrGbl->flag == 1)
   {
      ;
   }
#endif

   /* initialize function codes for param ram  and buffer length*/
   /* endian and function code */
   spiParamPtrGbl->rfcr = (uint8) FCR_INIT;
   /* endian and function code */
   spiParamPtrGbl->tfcr = (uint8) FCR_INIT;
   /* maximum buffer length */
   spiParamPtrGbl->mrblr = (uint16) MAX_BUF_LEN;

   /* initialize receive buffer descriptors */
   rbdPtrGbl->e = 1;           /* 1 = buffer ready to receive */
   rbdPtrGbl->w = 1;           /* 1 = wrap this bd */
   rbdPtrGbl->i = 1;           /* 1 = enable interrupts of rxb */
   rbdPtrGbl->cm = 1;          /* 1 = continuous mode */
   rbdPtrGbl->l = 0;           /* clear out */
   rbdPtrGbl->ov = 0;          /* clear out */
   rbdPtrGbl->me = 0;          /* clear out */
   rbdPtrGbl->dataLen = 0;     /* superfluous set by cp*/
   rbdPtrGbl->dataPtr = &spmGbl.rf;        /* assign ptr to buffer */

   /* initialize transmit buffer descriptors */
   tbdPtrGbl->r = 1;  /* 1 = buffer ready to transmit */
   tbdPtrGbl->w = 1;  /* 1 = wrap this bd */
   tbdPtrGbl->i = 0;  /* 0 = don't enable interrupts of txb */
   tbdPtrGbl->cm = 1; /* 1 = continuous mode */
   tbdPtrGbl->l = 1;  /* 1 = last  */
   tbdPtrGbl->un = 0; /* clear out */
   tbdPtrGbl->me = 0; /* clear out */
   tbdPtrGbl->dataLen = 2; /* set to buffer length should be 2bytes */
   tbdPtrGbl->dataPtr = &spmGbl.tf; /* assign ptr to buffer */

/*********************************************************************
  Initialize Globals Structure for Channel Access Algorithm
  and Special Purpose Mode
*********************************************************************/

   /* initialize receive frame buffers just to prevent garbage
      in case out of sync */
   spmGbl.rf.setTxFlag = 0;    /* frame status not transmitting */
   spmGbl.rf.clrTxReqFlag = 0; /* don't clear */
   spmGbl.rf.rxDataValid = 0;  /* no valid data this frame */
   spmGbl.rf.txDataCTS = 0;    /* no clear to send */
   spmGbl.rf.setCollDet = 0;   /* no collision detected */
   spmGbl.rf.rxFlag = 0;       /* not receiving */
   spmGbl.rf.rwAck = 0;        /* not acknowledged */
   spmGbl.rf.txOn = 0;         /* not receiving */
   spmGbl.rf.data = 0;         /*  */

   /* initialize receive frame buffers just to prevent garbage in case */
   /* out of sync*/
   spmGbl.tf.txFlag = 0;      /* frame status not transmitting */
   spmGbl.tf.txReqFlag = 0;   /* don't req */
   spmGbl.tf.txDataValid = 0; /* no valid data this frame */
   spmGbl.tf.blank = 0;       /* unused */
   spmGbl.tf.txAddrRW = 0;    /* write */
   spmGbl.tf.txAddr = 0;      /* default address */
   spmGbl.tf.data = 0;        /*  */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* Initialize spm global parameters */

spmGbl.mode = STOP;            /* no errors yet */
spmGbl.state = IDLE;           /* state machine start state */
macGbl.tpr = FALSE;            /* packet not ready to transmit */
macGbl.tc = 0;                 /* transmit byte count 0 */
macGbl.tl = 0;                 /* last byte is 0 */
macGbl.rpr = FALSE;            /* packet not yet received */
macGbl.rc = 0;                 /* receive byte count 0 */
macGbl.rl = 0;                 /* last byte is 0 */
spmGbl.crw = FALSE;            /* don't write config register */
spmGbl.cra = 0;                /* address zero nothing */
spmGbl.crData = 0;             /* empty data */
spmGbl.srr = FALSE;            /* don't read from status register */
spmGbl.sra = 0;                /* empty address */
spmGbl.srData = 0;             /* put nothing in */
spmGbl.resetCount = 0;         /* zero to start */
spmGbl.collisionsThisPkt = 0 ; /* # of collisions this packet */
macGbl.priorityPkt = FALSE;    /* FALSE = not a priority packet */

/* FALSE = channel access algo not complete */
spmGbl.accessApproved = FALSE;
/* TRUE means cycle timer to be reset and started */
spmGbl.cycleTimerRestart = TRUE;

spmGbl.backlog = 0;  /* current channel backlog */
/* delta backlog on current transmit packet */
macGbl.deltaBLTx = 0;
macGbl.deltaBLRx = 0;  /* delta backlog on last receive packet */
macGbl.altPathBit = 0; /* alternate path bit */
/* signal to write alternate path bit */
spmGbl.writeAltPathBit = FALSE;
/* alt path bit written state for this pkt */
spmGbl.altPathBitWritten = FALSE;
spmGbl.nodePriority = 0; /* node's priority slot number */
/* comm parameters for this node */
for ( i = 0; i < NUM_COMM_PARAMS; i++)
{
   spmGbl.configData[i] = 0;
}
spmGbl.phase = RANDOM_IDLE;   /* Idle for a long time */

spmGbl.kind = POST_RX;         /* beta1 time slot type */
spmGbl.nicsToTicks = 15;       /* conversion factor this specification's
                                  time base to 68360 ticks */
spmGbl.bitClockRate = 156250; /* in units of Hz */
spmGbl.beta2Ticks = 0;         /* length of beta2 in 68360 ticks
                                  40ns each */
spmGbl.beta1Ticks = 0;         /* length of beta1 in 68360 ticks
                                  40ns each */
spmGbl.beta1PostTxTicks = 0;  /* length of beta1 in 68360 ticks
                                   40ns each */
spmGbl.beta1PostRxTicks = 0;  /* length of beta1 in 68360 ticks
                                   40ns each */
spmGbl.baseTicks = 0;          /* duration of wbase in 68360 ticks
                                  40ns each */
spmGbl.cycleTicks = 0;         /* duration of avg packet cycle in
                                  68360 ticks 40ns each */
spmGbl.priorityChPostTxTicks = 0; /* duration of channel priority
                                     slots */
spmGbl.priorityChPostRxTicks = 0; /* duration of channel priority
                                     slots */
spmGbl.priorityIdleTicks = 0;  /* duration of priority idle wait */
/* duration until node's priority slot */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
spmGbl.priorityNodeTicks  = 0;
spmGbl.randomTicks        = 0;
/* timers for channel access algorithm */
spmGbl.idleTimerStart     = 0;
spmGbl.baseTimerStart     = 0;
spmGbl.cycleTimerStart    = 0;
spmGbl.transmitTimerStart = 0;
spmGbl.elapsed            = 0;
spmGbl.stopped            = 0;
spmGbl.lastTime           = 0;

/* initialize crcGbl  for good form */
crcGbl.poly = 0;
crcGbl.crc = 0;
crcGbl.crcBit = 0;
crcGbl.dataBit = 0;
crcGbl.dataByte = 0;

/* initialize port B for SPI Master operation */
/* pin0 = spisel or chip select if single master */
/* pin1 = spiclk   clock*/
/* pin2 = spimosi master out slave in */
/* pin3 = spimiso master in slave out */
/* pin5 = ~`reset */

pbodrPtrGbl->od0 = 0;  /*0=active driven 1=open drain(3 state), if output */
pbodrPtrGbl->od1 = 0;
pbodrPtrGbl->od2 = 0;
pbodrPtrGbl->od3 = 0;
pbodrPtrGbl->od5 = 0;  /* not open drain for reset? */

pbparPtrGbl->dd0 = 0;   /* general purpose pin chip select*/
pbparPtrGbl->dd1 = 1;   /* internal connect spiclk */
pbparPtrGbl->dd2 = 1;   /* internal connect spimosi */
pbparPtrGbl->dd3 = 1;   /* internal connect spimiso */
pbparPtrGbl->dd5 = 0;   /* general purpose i/o */

pbdirPtrGbl->dr0 = 1;   /* output */
pbdirPtrGbl->dr1 = 1;   /* output */
pbdirPtrGbl->dr2 = 1;   /* output */
/* although input must config as out else brgo4 */
pbdirPtrGbl->dr3 = 1;
pbdirPtrGbl->dr5 = 1;   /* output */


/* initialize port B for IO pins : reset switch, manual service request,
   ioswitch, and LEDs reset-service and IOLED
   pb8 = led reset service
   pb9 = general purpose out led
   pb12 = reset switch
   pb13 = manual service request
   pb14 = general purpose in switch
*/

pbodrPtrGbl->od8 = 0;   /* 0 = active driven not open drain  */
pbodrPtrGbl->od9 = 0;
/* since input NA, however open drain for pull down */
pbodrPtrGbl->od12 = 1;
/* since input NA, however open drain for pull down */
pbodrPtrGbl->od13 = 1;
/* since input NA, however no pull down */
pbodrPtrGbl->od14 = 0;

pbparPtrGbl->dd8 = 0;        /* general purpose pin chip select*/
```


--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
pbparPtrGbl->dd9 = 0;        /* general purpose pin chip select*/
pbparPtrGbl->dd12 = 0;       /* general purpose pin chip select*/
pbparPtrGbl->dd13 = 0;       /* general purpose pin chip select*/
pbparPtrGbl->dd14 = 0;       /* general purpose pin chip select*/

pbdirPtrGbl->dr8 = 1;        /* output */
pbdirPtrGbl->dr9 = 1;        /* output */
pbdirPtrGbl->dr12 = 0;       /* input */
pbdirPtrGbl->dr13 = 0;       /* input */
pbdirPtrGbl->dr14 = 0;       /* input */

/* set initial values */
pbdatPtrGbl->d8 = 0;
pbdatPtrGbl->d9 = 0;
pbdatPtrGbl->d12 = 1; /* input so NA */
pbdatPtrGbl->d13 = 1; /* input so NA */
pbdatPtrGbl->d14 = 0; /* input so NA */

/* reset xcvr */
/* driven low to reset hold in reset until change. */
pbdatPtrGbl->d5 = 0;

/* clear any previous spi interrupt events write 1 to clear */
spiemTemp.mme = 1;
spiemTemp.txe = 1;
spiemTemp.bsy = 1;
spiemTemp.txb = 1;
spiemTemp.rxb = 1;
*spiePtrGbl = spiemTemp;  /* write it out to SPIE */

/* assign address of isr functions to vector table entry */
*spivPtrGbl = &SPMIsr;         /* spi isr */
*cerrvPtrGbl = &CErrIsr;       /* comm error isr */
*spurintvPtrGbl = &SpurIntIsr; /* spurious interrupt isr */
/* *buserrvPtrGbl = &BusErrIsr; */ /* spurious interrupt isr */

/* enable interrupts write 1 to enable */
spiemTemp.mme = 0;
spiemTemp.txe = 0;
spiemTemp.bsy = 0;
spiemTemp.txb = 0;  /* isr will run after transmit */
spiemTemp.rxb = 1;  /* isr will run after receive buffer filled */
*spimPtrGbl = spiemTemp;  /* write it out to SPIM */

/* init CICR register */
cicrTemp.scdp = 3; /* scc4 has priority d */
cicrTemp.sccp = 2; /* scc3 has priority c */
cicrTemp.scbp = 1; /* scc2 has priority b */
cicrTemp.scap = 0; /* scc1 has priority a */
cicrTemp.irl  = CPIR_LEVEL; /* level 4 interrupt request level */
cicrTemp.hpi  = 31; /* default highest priority = 31 = 1F hex*/
cicrTemp.vba  = UIVN_MSB; /* user interrupt vector number offset */
cicrTemp.sps  = 0; /* use grouped priority scheme */
*cicrPtrGbl = cicrTemp; /* write to cicr */

/* clear any old SPI interrupts from CISR */
cisrPtrGbl->spi = 1; /* clear by writing 1,
                        spi interrupt in service */
/* enable system interrupt for spi in CIMR*/
cimrPtrGbl->spi = 1; /* enable by writing 1 */

/* init SPMODE register */
spmodeTemp.loop  = 0;       /* 1 = local loop back
                               0 = no local loop back */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   spmodeTemp.ci    = 0;       /* don't invert clock */
   spmodeTemp.cp    = 1;       /* toggle clock at begin of data transfer */
   spmodeTemp.div16 = 0;       /* don't divide brgclk by 16 */
   spmodeTemp.rev   = 1;       /* use bigendian MSB first */
   spmodeTemp.ms    = 1;       /* master mode */
   spmodeTemp.len   = 7;       /* 8 bits per character = len + 1 */
   spmodeTemp.pms   = 4;       /* prescale modulus select,
                                  4 gives 1.25 Mbps */
   spmodeTemp.en    = 1;       /* enable spi  */
   *spmodePtrGbl = spmodeTemp; /* write to spmode register */

   /* initialize SPCOM (probably don't need to do */
   spcomPtrGbl->res = 0;  /* should never have to change */
   spcomPtrGbl->str = 0;  /* superfluous auto cleared one
                             clock cycle after set */

   /*************************************************************
    Enable interrupts by setting interrupt priority mask level
    in Status Register to 0, must use in line assembly code
    assumes CPU is in supervisor mode, Bits 10 9 8 are
    interrupt priority mask level
   *************************************************************/

   asm(" ANDI #~0x0700,SR"); /* sets priority mask to level 000 */


   /* Set up frame clock using scc2*/
   /* put data into transmit buffer */
   scc2tBufGbl[0][0] = 0x01U;    /* first byte 7 bit delay
                                    to sync frame to spi */
   scc2tBufGbl[0][1] = 0x00U;    /* second byte */
   scc2tbdPtrGbl[0].dataLen = 2; /* send just two bytes */


   /* enable frame clock */
   gsmrl2PtrGbl->enr = 1;  /* receive */
   gsmrl2PtrGbl->ent = 1;  /* transmit */


   return(1);
}


/*************************************************************
Function:  MACTimerInit
Returns:   pointer to hardware timer counter register
Reference:
Purpose:   initialize MAC  hardware timer
Comments:
*************************************************************/

uint32 * MACTimerInit(void)
{
   tgcrPtrGbl->cas2 = 1; /* cascade 1 and 2 */
   tgcrPtrGbl->frz2 = 0; /* ignore freeze pin */
   tgcrPtrGbl->stp2 = 0; /* normal unstopped operation */
   tgcrPtrGbl->rst2 = 0; /* reset timer */

   tmr2PtrGbl->ps   = 0; /* prescaler */
   tmr2PtrGbl->ce   = 0; /* input capture disabled */
   tmr2PtrGbl->om   = 0; /* output mode pulse */
   tmr2PtrGbl->ori  = 0; /* diable output reference interrupt */
   tmr2PtrGbl->frr  = 0; /* 0= free running counter */
   tmr2PtrGbl->iclk = 1; /* 1 = internal general system clk source */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   tmr2PtrGbl->ge   = 0; /* tgate is ignored */

   /* set reference to max all ones */
   *trr12PtrGbl = (uint32) 0xFFFFFFFFU;

   /* writing a 1 resets the reference event bit */
   ter2PtrGbl->ref = 1;
   /* writing a 1 resets the capture event bit */
   ter2PtrGbl->cap = 1;

   *tcn12PtrGbl = 0; /* start at zero */

   tgcrPtrGbl->rst2 = 1; /* start timer running */

   return(tcn12PtrGbl);
}


/*****************************************************************
Function:  DelayTicks
Returns:   Boolean state of mac timer running = true stopped = false
Reference:
Purpose:   Delay for a "delay" number of ticks
           Used during various startup routines
           to wait for xcvr hardware to respond
Comments:
*****************************************************************/

Boolean DelayTicks(uint32 delay)
{
   uint32 start;

   if (tgcrPtrGbl->rst2 == 0)
   {
      /* timer not running so exit immediately with false */
      return (FALSE);
   }
   else
   {
      start = *tcn12PtrGbl;

      while ((*tcn12PtrGbl - start) < delay)
      {
         ; /* keep waiting */
      }

      return (TRUE);
   }
}


/*****************************************************************
Function:  PHYSoftResetSPMXCVR
Returns:
Reference:
Purpose:  Software reset Special Purpose Mode XCVR by reading
          from register zero
Comments:
*****************************************************************/

void PHYSoftResetSPMXCVR(void)
{
   uint16 count;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   spmGbl.sra = 0;
   spmGbl.srr = TRUE;

   count = 0;
   while ((spmGbl.srr != FALSE) && (count < 0xFFFF))
   {
      count++;
   }
   spmGbl.srData = 0;

   return;
}

/*****************************************************************
Function:  PHYHardResetSPMXCVR
Returns:
Reference:
Purpose:   hardware reset Special Purpose Mode XCVR by
           asserting reset pin. then configures xcvr
           this is used when xcvr hangs in tx_on mode
           e.g. spmGbl.resetCount times out
Comments:
*****************************************************************/

void PHYHardResetSPMXCVR(void)
{
   unsigned long count;
   int i;
   Boolean good;


   pbdatPtrGbl->d5 = 0; /* drive xcvr into reset */

   /* wait for a while for everything to settle down */
   good = DelayTicks(RESTART_DELAY_TICKS);


   pbdatPtrGbl->d5   = 1; /* bring xcvr out of reset */

   /* initialize config registers on PLT-20 transceiver */

   /* load in reverse order */
   i = NUM_COMM_PARAMS - 1; /* should be 6 */
   while ( i >= 0)
   {
      spmGbl.crData = spmGbl.configData[i];
      /* config registers start at 7 down to 1 */
      spmGbl.cra = (Byte) i + 1;
      spmGbl.crw = TRUE;
      count = 0;
      /* wait for ack but time out so not infinite loop */
      while ((spmGbl.crw != FALSE) && (count < 0xFFFFF))
      {
         count++;
      }
      if (spmGbl.crw == TRUE) /* xcvr never acked try again */
      {
         pbdatPtrGbl->d5   = 0; /* reset xcvr */

         /* wait for a while for everything to settle down */
         good = DelayTicks(RESTART_DELAY_TICKS);


         pbdatPtrGbl->d5   = 1; /* bring xcvr out of reset */
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            i = NUM_COMM_PARAMS - 1; /* start over again */
         } /* NOTE!!! if the xcvr never acks this code will hang */
         else
         {
            i--;
         }
      }

      spmGbl.crData = 0;
      spmGbl.cra = 0;

      return;
   }

   /*****************************************************************
   Function:  PHYDisableSPMIsr
   Returns:
   Reference:
   Purpose:   To diable the Isr temporarily on node reset.
              Must use PHYEnableIsr to reenable correctly.
   Comments:
   *****************************************************************/

   void PHYDisableSPMIsr(void)
   {
      /* once the first line below is executed, the ISR will stop
         running. the next time it runs */
      spmGbl.mode  = STOP;

      pbdatPtrGbl->d5 = 0; /* drive xcvr into reset */

      spmGbl.state = DEBUG;


      macGbl.tpr   = FALSE;
      macGbl.tc    = 0;
      macGbl.tl    = 0;
      macGbl.rpr   = FALSE;
      macGbl.rc    = 0;
      macGbl.rl    = 0;
      spmGbl.crw   = FALSE;
      spmGbl.srr   = FALSE;
      spmGbl.sra   = 0;

      return;
   }

   /*****************************************************************
   Function:  PHYEnableSPMIsr
   Returns:
   Reference:
   Purpose:   To Enable the Isr after Being disabled on node reset
              Also to enable on powerup etc.
              Computes clock rates etc based on node comm parameters
   Comments:
   *****************************************************************/

   void PHYEnableSPMIsr(void)
   {
      long  i;
      uint32 inputClock;
      uint32 commClock;
      uint32 rxPad;
      uint32 txPad;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
uint32 beta2;
Boolean good;

/* compute conversion factor nics_To_Tick  */
/* converts from this specification's time base to 683660 ticks */
inputClock = (uint32) eep->configData.inputClock;
commClock = (uint32) eep->configData.commClock;
spmGbl.nicsToTicks = NICS_TICKS_BASE >> inputClock;

/* compute channel access algorithm parameters */

beta2 = (eep->configData.reserved[2] * 20) + 40;

/* compute beta1 = postpacket + networkidlewait +
                   interpacketpad + prepacket */

if (eep->configData.reserved[4] < 128)
{
   rxPad = eep->configData.reserved[4] * 41;
}
else
{
   rxPad = (eep->configData.reserved[4] - 128) * 145;
}

if (eep->configData.reserved[3] < 128)
{
   txPad = eep->configData.reserved[3] * 41;
}
else
{
   txPad = (eep->configData.reserved[3] - 128) * 145;
}

/* set beta1 time as per formula with implementation
   specific adjustment BETA1_ADJUST_TICKS */
spmGbl.beta1PostRxTicks =
   (285 + beta2 + rxPad + 317) * spmGbl.nicsToTicks +
   BETA1_ADJUST_TICKS;
spmGbl.beta1PostTxTicks =
   (307 + beta2 + txPad + 317) * spmGbl.nicsToTicks +
   BETA1_ADJUST_TICKS;

/* set beta2 time as per formula with implementation
   specific adjustment BETA2_ADJUST_TICKS */
spmGbl.beta2Ticks =
   beta2 * spmGbl.nicsToTicks + BETA2_ADJUST_TICKS;
spmGbl.baseTicks = spmGbl.beta2Ticks * W_BASE;
/* set cycle time as per formula with implementation
   specific adjustment CYCLE_ADJUST_TICKS */
spmGbl.cycleTicks =
   eep->configData.reserved[1] * 1794 * spmGbl.nicsToTicks;

spmGbl.beta1Ticks = 0; /* this is set by isr to either post rx or
                          post tx beta1 as appropriate */

spmGbl.nodePriority =
   eep->configData.nodePriority; /* set node's priority slot # */

/* Add W_Base slots to wait time for post tx access only when there
   are non zero number of channel priority slots */
if (eep->configData.channelPriorities > 0)
{
    /* duration of channel priority slots */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   spmGbl.priorityChPostTxTicks = (eep->configData.channelPriorities +
                                   W_BASE ) * spmGbl.beta2Ticks;
    /* duration of channel priority slots */
   spmGbl.priorityChPostRxTicks = eep->configData.channelPriorities
                               * spmGbl.beta2Ticks;
}
else
{
   /* duration of channel priority slots */
   spmGbl.priorityChPostTxTicks = 0;
   spmGbl.priorityChPostRxTicks = 0;
}
/* duration of priority idle wait */
spmGbl.priorityIdleTicks = spmGbl.priorityChPostTxTicks;

/* set time of nodes priority slot. If eep->configData.nodePriority is 0,
   then this field will be negative, but it is not used in this case. */
spmGbl.priorityNodeTicks =
   (eep->configData.nodePriority - 1) * spmGbl.beta2Ticks;
spmGbl.randomTicks = 0;
spmGbl.bitClockRate = (BIT_CLOCK_BASE << inputClock) >>
   (RATIO_BASE + commClock);


/* make local copy of xcvr parameters so can reset xcvr free of changes to eep
   without following node reset. Needed in case xcvr locks up, See
   PHYHardResetSPMXCVR(void)
   */
for (i = 0; i < NUM_COMM_PARAMS; i++)
{
   spmGbl.configData[i] = eep->configData.param.xcvrParams[i];
}

/* reinitialize remainder of spmGbl */
spmGbl.mode = RUN;                /* get ready to go */
spmGbl.state = IDLE;              /* state machine start state */
macGbl.tpr = FALSE;               /* packet not ready to transmit */
macGbl.tc = 0;                    /* transmit byte count 0 */
macGbl.tl = 0;                    /* last byte is 0 */
macGbl.rpr = FALSE;               /* packet not yet received */
macGbl.rc = 0;                    /* receive byte count 0 */
macGbl.rl = 0;                    /* last byte is 0 */
spmGbl.crw = FALSE;               /* don't write config register */
spmGbl.cra = 0;                   /* address zero nothing */
spmGbl.crData = 0;                /* empty data */
spmGbl.srr = FALSE;               /* don't read from status register */
spmGbl.sra = 0;                   /* empty address */
spmGbl.srData = 0;                /* put nothing in */
spmGbl.resetCount = 0;            /* zero to start */
spmGbl.collisionsThisPkt = 0 ;    /* # of collisions this packet */
macGbl.altPathBit = 0;            /* alternate path bit */
spmGbl.writeAltPathBit = FALSE;   /* signal to write
                                     alternate path bit */
spmGbl.altPathBitWritten = FALSE; /* alt path bit written
                                     state for this pkt */
macGbl.priorityPkt = FALSE;        /* FALSE = not a priority packet */

spmGbl.accessApproved = FALSE;     /* FALSE = channel access algo
                                      not complete */
spmGbl.cycleTimerRestart = TRUE;   /* indicates whether cycle
                                      timer can be updated */

spmGbl.backlog = 0;                /* current channel backlog */
macGbl.deltaBLTx = 0;              /* delta backlog on curr transmit packet */
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   macGbl.deltaBLRx = 0;              /* delta backlog on last receive packet */
   spmGbl.phase = RANDOM_IDLE;        /* Idle for a long time */

   spmGbl.kind = POST_RX;             /* beta1 time slot type */
   spmGbl.idleTimerStart = 0;
   spmGbl.baseTimerStart = 0;
   spmGbl.transmitTimerStart = 0;
   spmGbl.elapsed = 0;
   /* spmGbl.clock is initialized in PHYInitSPM */
   spmGbl.stopped = *(spmGbl.clock);
   spmGbl.cycleTimerStart = spmGbl.stopped; /* start up 1st time */
   spmGbl.lastTime = spmGbl.stopped;


   /* configure bit clock */
   /* div16 = 1 and pms = 4 gives 78.125 khz */
   /* div16 = 1 and pms = 3 gives 97.656 khz */
   /* div16 = 1 and pms = 2 gives 130.208 khz */
   /* div16 = 1 and pms = 1 gives 195.312 khz */
   /* div16 = 1 and pms = 0 gives 390.625 khz */
   /* div16 = 0 and pms = 9 gives 625.000 khz */

/*****************************************************************
      Since this implementation runs in noncontinuous mode the
      effective frame period is the actual frame period plus the
      processing time for the ISR at the end of the frame. Therefore
      we need to run at a higher bit clock than specified by the
      network management so that the effective frame rate is the same
      or better than the network manager specifies. For a bit rate of
      156.25 Khz the frame period is 16 * 6.4 usec = 102.4
      usec. Suppose the worst case time for ISR is 58 micro seconds
      then the time left over for the actual frame is 102.4 - 58 =
      44.4 usec. This corresponds to a bit rate of 16 * 1/(44.4 usec)
      = 360.36 khz. The closest bit rate greater than this that the
      360 supports is 390.625 khz. The effective frame period becomes
      58 usec + (16 * (1/390.625 khz)) = 98.96 usec. This is faster so
      OK. Effective bit rate is 16 * 1/(98.96 usec) = 161.68 kHz.
      Unless we can more than half the time for the ISR to run we
      can't run fast enough to make the next step that is the 312.5
      Khz bit rate.
*****************************************************************/

   switch (spmGbl.bitClockRate)
   {
      case 78125:
         /* run at 130 khz to get same or faster effective rate */
         spmodePtrGbl->div16 = 1; /* divide by 16  */
         spmodePtrGbl->pms   = 2; /* prescale modulus select */
         break;
      case 156250:
         /* run at 390 khz to get same or faster effective rate */
         spmodePtrGbl->div16 = 1; /* divide by 16  */
         spmodePtrGbl->pms   = 0; /* prescale modulus select */
         break;
      default:
         /* 156250 */
         /* run at 390 khz to get same or faster eff rate */
         spmodePtrGbl->div16 = 1; /* divide by 16  */
         spmodePtrGbl->pms   = 0; /* prescale modulus select */
         break;
   }

   /* reinitialize receive frame buffers */
   spmGbl.rf.setTxFlag    = 0; /* frame status not transmitting */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    spmGbl.rf.clrTxReqFlag = 0; /* don't clear */
    spmGbl.rf.rxDataValid  = 0; /* no valid data this frame */
    spmGbl.rf.txDataCTS    = 0; /* no clear to send */
    spmGbl.rf.setCollDet   = 0; /* no collision detected */
    spmGbl.rf.rxFlag       = 0; /* not receiving */
    spmGbl.rf.rwAck        = 0; /* not acknowledged */
    spmGbl.rf.txOn         = 0; /* not receiving */
    spmGbl.rf.data         = 0; /*  */

    /* reinitialize receive frame buffers */
    spmGbl.tf.txFlag      = 0;  /* frame status not transmitting */
    spmGbl.tf.txReqFlag   = 0;  /* don't req */
    spmGbl.tf.txDataValid = 0;  /* no valid data this frame */
    spmGbl.tf.blank       = 0;  /* unused */
    spmGbl.tf.txAddrRW    = 0;  /* write */
    spmGbl.tf.txAddr      = 0;  /* default address */
    spmGbl.tf.data        = 0;  /*  */

    /* initialize crcGbl  for good form */
    crcGbl.poly = 0;
    crcGbl.crc = 0;
    crcGbl.crcBit = 0;
    crcGbl.dataBit = 0;
    crcGbl.dataByte = 0;


    /* Send First Frame  ISR keeps it going */
    tbdPtrGbl->r     = 1; /* buffer ready to transmit */
    spcomPtrGbl->str = 1;

    /* wait for a while for everything to settle down */
    good = DelayTicks(RESTART_DELAY_TICKS);


    PHYHardResetSPMXCVR();  /* this also writes config registers */

    return;

}

/*****************************************************************
Function:  PHYIOInit
Returns:   None
Reference: None
Purpose:   To Init variables related to PHYIO function.
Comments:  None
*****************************************************************/
void PHYIOInit(void)
{
    SetMsTimer(&phyIOTimer, (uint16)(PHYIO_CHECK_INTERVAL * 1000));
    gp->resetPinPrevState    = 1; /* Allow Reset Button Push */
    gp->manualServiceRequestPrevState  = 1; /* Allow Manual Service Request Push */
    gp->ioInputPin0PrevState = 1;
}

/*****************************************************************
Function:  PHYIO
Returns:   None
Reference: None
Purpose:   To handle Input switches (reset service and IO) and
           to set value of ouput leds
           Manual Service Request and Reset pin share same LED to indicate
           switch pushed
Comments:  None
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
*****************************************************************/
void PHYIO(void)
{

   /* Perform Pin related checks only on timer expiry
      to avoid bounce problem */
   if (MsTimerExpired(&phyIOTimer))
   {
      /* check reset switch */
      /* reset pulled low &&
         reset not enabled so need to enable &&
         have not yet set this pin push */
      if (( pbdatPtrGbl->d12 == 0) &&
          !gp->resetNode &&
          (gp->resetPinPrevState != 0))
      {
         gp->resetNode = TRUE; /* signal scheduler there is reset */
         nmp->resetCause = EXTERNAL_RESET;
         /* NodeReset function will clear when reset is finished */
         /* prevents sending more than 1 reset until pin is toggled */
         gp->resetPinPrevState = 0;
      }

      /* reset pin high */
      if (pbdatPtrGbl->d12 == 1)
      {
         gp->resetPinPrevState = 1;
      }

      if ((pbdatPtrGbl->d13 == 0) && /* manual service request pulled low */
          !gp->manualServiceRequest &&      /* manualServiceRequest not enabled */
          (gp->manualServiceRequestPrevState != 0)) /* have not yet set
                                                       this pin push */
      {
         gp->manualServiceRequestPrevState = 0;
         gp->manualServiceRequest = TRUE; /* notify stack that
                                             manual service request enabled */
      }

      if (pbdatPtrGbl->d13 == 1) /* manual service request high */
      {
         gp->manualServiceRequestPrevState = 1; /* manual service request toggled
                                                   so next press will allow push */
      }

      if(pbdatPtrGbl->d14 == 1 &&
         (gp->ioInputPin0PrevState != 0))  /* debouncer, check
                                              gen purpose io input */
      {
         gp->ioInputPin0 = 1;
      }
      else
      {
         gp->ioInputPin0 = 0;
      }

      if (pbdatPtrGbl->d14 == 1) /* io pin high */
      {
         gp->ioInputPin0PrevState = 1; /* io pin toggled so next
                                          press will allow push */
      }

      SetMsTimer(&phyIOTimer,
                 (uint16)(PHYIO_CHECK_INTERVAL * 1000));
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }


   if (gp->resetNode || gp->manualServiceRequest || gp->ioOutputPin0)
   {
      pbdatPtrGbl->d8 = 1; /* turn on reset-service light */
   }
   else
   {
      pbdatPtrGbl->d8 = 0; /* turn off reset-service light */
   }


   if(gp->ioOutputPin1 == 0) /* set io output pin */
   {
      pbdatPtrGbl->d9 = 0;
   }
   else
   {
      pbdatPtrGbl->d9 = 1;
   }

   return;
}


/*****************************************************************
Function:  IncrementBacklog
Returns:
Reference:
Purpose:   increments backlog and handles backlog overflow
Comments:
*****************************************************************/

static void    IncrementBacklog(uint8 deltaBacklog)
{
   if (deltaBacklog > MAX_BACKLOG)
   {
      return; /* Erroneous deltaBacklog value */
   }
   /* spmGbl.backlog size must allow 2 * MAX_BACKLOG without
      wrap around */
   spmGbl.backlog += deltaBacklog;
   if (spmGbl.backlog > MAX_BACKLOG)
   {
      spmGbl.backlog = MAX_BACKLOG;
      INCR_STATS(nmp->stats.backlogOverflow);
   }
}


/*****************************************************************
Function:  DecrementBacklog
Returns:
Reference:
Purpose:   decrements backlog and handles backlog underflow
Comments:
*****************************************************************/

static void    DecrementBacklog(uint8 deltaBacklog)
{
   if (deltaBacklog > MAX_BACKLOG)
   {
      return; /* Erroneous deltaBacklog value */
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }
   if (spmGbl.backlog <= deltaBacklog)
   {
      spmGbl.backlog = 0;
   }
   else
   {
      spmGbl.backlog -= deltaBacklog;
   }
}

/*****************************************************************
Function:  SPMIsr
Returns:   none, no arguments allowed either
Reference:
Purpose:
    Interrupt Service Routine

    This ISR serves two primary functions. the first is to execute the
    channel access algorithm. the second is to performs the special
    purpose mode transfers with the XCVR. For the channel access
    algorithm a single hardware timer is used as reference for a set
    of global values that are the expire times for the various
    channel access algorithm timers. The ISR polls the hardware timer
    each time the ISR runs and checks for expiration of the various
    timers.  The ISR has two coupled state machines. the first
    indicated by spmGbl.phase controls the channel access algorithm
    and the second indicated by spmGbl.state controls the SPM
    transfers.

    The ISR runs after each 16 SPI transfers or clock transitions.
    An Interrupt is generated by an rxb event when a receive buffer
    is full.

Comments:  must use interrupt pragma
*****************************************************************/
/*  interrupt pragma informs compiler to make next function
    an interrupt handler with RTE and saved state of registers */

#pragma interrupt()

static void SPMIsr()
{
   SPIEventMaskReg spieTemp;  /* local copy of event register */
   int j;  /* for loop index */

   spieTemp = *spiePtrGbl;  /* read SPIE into local copy */

   /* clear spie ASAP so as not to miss any events while in isr */
   /* write all ones to clear */
   *((uint8 * ) spiePtrGbl) = (uint8) 0xFFU;

   /* each event handled by this isr needs a separate if statement */
   if ( spieTemp.rxb == 1)
   {
      /* handle interrupt for receive buffer filled */
      /* transfer can be stopped by physical layer or for error */
      if (spmGbl.mode == STOP) /* stop spm exchanges  */
      {
         spmGbl.state = DEBUG;
         macGbl.tc = 0;
         macGbl.tl = 0;
         macGbl.tpr = FALSE;
         macGbl.rc = 0;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

```
         macGbl.rl = 0;
         macGbl.rpr = FALSE;
         spmGbl.crw = FALSE;
         spmGbl.srr = FALSE;
      }
      else    /* State Machines for Transmit and Receive */
      {

         /* debug */
         /* last received frame is in spmGbl.rf  */
         /* update history for debugging purposes*/
#ifdef SPM_HISTORY
         hBufGbl.records[hBufGbl.index].rf = spmGbl.rf;
         hBufGbl.records[hBufGbl.index].state = spmGbl.state;
         hBufGbl.records[hBufGbl.index].rb = spiParamPtrGbl->rbptr;
         hBufGbl.records[hBufGbl.index].tb = spiParamPtrGbl->tbptr;

         hBufGbl.records[hBufGbl.index].start = *(spmGbl.clock);
#endif
         /*******************************************/
         /* state machine for channel access algorithm */
         /*******************************************/
         switch (spmGbl.phase)
         {
            case BUSY: /* some node is transmitting */
               /* waits here until channel is idle */

               if ((spmGbl.rf.rxFlag == 0) &&
                   (spmGbl.rf.txOn == 0))
               {
                  /* channel has become idle neither receive or transmit */
                  /* start idle timer */
                  spmGbl.idleTimerStart = *(spmGbl.clock);

                  /* set beta1 duration based on last packet,
                     either rx or tx */
                  if (spmGbl.kind == POST_RX)
                  {
                     spmGbl.beta1Ticks = spmGbl.beta1PostRxTicks;
                     spmGbl.priorityIdleTicks = spmGbl.priorityChPostRxTicks;
                  }
                  else
                  {
                     spmGbl.beta1Ticks = spmGbl.beta1PostTxTicks;
                     spmGbl.priorityIdleTicks = spmGbl.priorityChPostTxTicks;
                  }

                  /* Previous Packet has ended so reset altPathBit flags.
                     Once tpr goes true then we will know to write
                     altPathBit once before tx */
                  spmGbl.writeAltPathBit = FALSE;
                  spmGbl.altPathBitWritten = FALSE;

                  /* start waiting out beta1 time */
                  spmGbl.phase = BETA1_IDLE;
               }
               else /* channel still busy */
               {
                  if (spmGbl.rf.rxFlag == 1)
                  {
                     /* busy receiving packet */
                     spmGbl.kind = POST_RX;
                  }
                  else
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      {
         /* busy transmitting packet */
         spmGbl.kind = POST_TX;
      }


      spmGbl.phase = BUSY;
   }
   break;
case BETA1_IDLE:  /* waits out beta1 time */

   if ((spmGbl.rf.rxFlag == 1) ||
       (spmGbl.rf.txOn == 1))
   {
      /* channel is now busy */
      /* other node must have used channel or
         xcvr is in error with bad txOn*/

      spmGbl.phase = BUSY;
   }
   else if ((macGbl.tpr == TRUE) &&
            (spmGbl.writeAltPathBit == FALSE) &&
            (spmGbl.state == IDLE)) /*in case doing status register
                                      read*/
   {
      /* tx packet is ready */
      /* always do unconfirmed write
         of altPathBit before tx */
      spmGbl.writeAltPathBit = TRUE;

      /* wait until next frame before begin channel
         access */
      spmGbl.phase = BETA1_IDLE;
   }
   else
   {
      /* check idle timer to see if beta1 time expired */
      spmGbl.elapsed =
         *(spmGbl.clock) - spmGbl.idleTimerStart;

      if ( spmGbl.elapsed >= spmGbl.beta1Ticks )
      {
         /* idle timer expired */
         /* check if transmit packet is still ready and
            set priority access type */
         /* conditions for priority access
            Packet marked as priority
            Collisions less than 2
            Node has a priority slot
            Only after  received packet
            only after valid crc on last received pkt
         */
         if ( (macGbl.tpr == TRUE) &&
              (macGbl.priorityPkt == TRUE) &&
              (spmGbl.kind == POST_RX) && /* only after  rx */
              (spmGbl.collisionsThisPkt <= 1) &&
              (spmGbl.nodePriority > 0) &&
              (spmGbl.state == IDLE)) /* in case doing status */
                                      /* register read       */
         {

            /* attempt priority slot access */
            spmGbl.transmitTimerStart = spmGbl.idleTimerStart +
                                        spmGbl.beta1Ticks;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            spmGbl.phase = PRIORITY_WAIT_TX;
         }
         else
         {

            /* do not attempt priority slot access */
            /* start waiting out priority slots */
            spmGbl.phase = PRIORITY_IDLE;

         }
      }
      else
      {
         spmGbl.phase = BETA1_IDLE; /* keep waiting */
      }
   }
   break;
case PRIORITY_IDLE:
   if ((spmGbl.rf.rxFlag == 1) ||
       (spmGbl.rf.txOn == 1)) /* busy */
   {
      /* other node must have used channel or
         xcvr is in error with bad txOn*/

      spmGbl.phase = BUSY;
   }
   else if ((macGbl.tpr == TRUE) &&
            (spmGbl.writeAltPathBit == FALSE) &&
            (spmGbl.state == IDLE)) /*in case doing status register
                                      read*/
   {
      /* tx packet is ready */
      /* always do unconfirmed write of altPathBit
         before tx */
      spmGbl.writeAltPathBit = TRUE;

      /* have to wait until next frame before
         access channel */
      spmGbl.phase = PRIORITY_IDLE;
   }
   else
   {
      /* check idle timer for priority slots to end */
      spmGbl.elapsed =
         *(spmGbl.clock) - spmGbl.idleTimerStart;
      if ( spmGbl.elapsed >=
           (spmGbl.priorityIdleTicks + spmGbl.beta1Ticks)) /* idle  */
                                                   /* timer expired */
      {
         /* timer expired so check if transmit packet ready */
         if ( (macGbl.tpr == TRUE) &&
            (spmGbl.state == IDLE)) /*in case doing status register
                                      read*/
         {
            /* start transmit timer */
            spmGbl.transmitTimerStart = spmGbl.idleTimerStart +
                                        spmGbl.beta1Ticks +
                                        spmGbl.priorityIdleTicks;
            /* start base timer */
            /* base timer runs when waiting for randomized */
            /* tx access */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            /* spmGbl.baseTimerStart = spmGbl.idleTimerStart +
                                          spmGbl.beta1Ticks +
                                          spmGbl.priorityIdleTicks; */

               spmGbl.baseTimerStart = spmGbl.transmitTimerStart;
                  /* faster way */

              /* change seed */
               srand((unsigned int) spmGbl.transmitTimerStart);

              /* attempt random slot access */
               /* random slot between 0 and backlog * wbase - 1*/

               spmGbl.randomTicks = (rand() % ((spmGbl.backlog + 1) *
                                    W_BASE)) * spmGbl.beta2Ticks ;

               spmGbl.phase = RANDOM_WAIT_TX; /* go wait for slot */
            }
            else  /* priority idle timer has expired but transmit   */
                  /* packet is not ready */
            {
               spmGbl.phase = RANDOM_IDLE; /* go wait */
            }
         }
         else /*timer not elapsed so keep waiting for end of priority
                slots*/
         {
            spmGbl.phase = PRIORITY_IDLE;
         }
      }
      break;
   case RANDOM_IDLE:
      if ((spmGbl.rf.rxFlag == 1) || (spmGbl.rf.txOn == 1)) /* busy */
      {
         /* other node must have used channel or
            xcvr is in error with bad txOn*/
         spmGbl.phase = BUSY;
      }
      else if ((macGbl.tpr == TRUE) && /* tx packet ready */
               (spmGbl.writeAltPathBit == FALSE) &&
               (spmGbl.state == IDLE)) /*in case doing status register
                                         read*/
      {
         /* always do unconfirmed write of altPathBit before tx */
         spmGbl.writeAltPathBit = TRUE;

         /* have to wait until next frame to start channel access */
         spmGbl.phase = RANDOM_IDLE;
      }
      else
      {
         /* check if transmit packet still ready  */
         if ( (macGbl.tpr == TRUE)  &&
            (spmGbl.state == IDLE) ) /* in case doing status register
                                       read /
         {
            spmGbl.transmitTimerStart = *(spmGbl.clock); /* start */
                                                         /* transmit */
                                                         /* timer */

            /* base timer runs when waiting for randomized tx access */
            /* start base timer */
            spmGbl.baseTimerStart = spmGbl.transmitTimerStart;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            srand((unsigned int) spmGbl.transmitTimerStart); /*change */
                                                             /* seed*/

            /* attempt random slot access */
            /* random slot between 0 and backlog * wbase - 1*/
            spmGbl.randomTicks = (rand() % ((spmGbl.backlog + 1) *
                                      W_BASE)) * spmGbl.beta2Ticks ;

            spmGbl.phase = RANDOM_WAIT_TX; /* go wait for slot */
         }
         else
         {
            spmGbl.phase = RANDOM_IDLE; /* nothing to send keeping */
                                        /* waiting */
         }
      }
      break;
   case PRIORITY_WAIT_TX:
      if ((spmGbl.rf.rxFlag == 1) || (spmGbl.rf.txOn == 1)) /* busy */
      {
          /* other node must have used channel or
            xcvr is in error with bad txOn*/
         spmGbl.phase = BUSY;
      }
      else
      {
         /* check tx timer wait for our priority slot to come up*/
         spmGbl.elapsed = *(spmGbl.clock) - spmGbl.transmitTimerStart;
         if (spmGbl.elapsed >= spmGbl.priorityNodeTicks)
         {
            /* approve transmission */
            spmGbl.accessApproved = TRUE; /* enable spm state machine */
                                          /*to begin tx */
            spmGbl.phase = START_TX;
         }
         else
         {
            spmGbl.phase = PRIORITY_WAIT_TX;
         }
      }
      break;
   case RANDOM_WAIT_TX:
      if ((spmGbl.rf.rxFlag == 1) || (spmGbl.rf.txOn == 1)) /* busy */
      {
         /* other node must have used channel or
            xcvr is in error with bad txOn*/
         spmGbl.phase = BUSY;
      }
      else
      {

        /* Get current time. Since used for two checks we want   */
        /* to be synchronized                                    */
         spmGbl.stopped = *(spmGbl.clock);

         /* check tx timer wait for out random slot to come up */
         spmGbl.elapsed = spmGbl.stopped - spmGbl.transmitTimerStart;
         if (spmGbl.elapsed >= spmGbl.randomTicks)
         {
            /* approve transmission */
            spmGbl.accessApproved = TRUE; /* enable spm state machine to
                                             begin tx */
            spmGbl.phase = START_TX;
         }
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            else
            {
               spmGbl.phase = RANDOM_WAIT_TX;
            }


            /* check base timer, base timer running when we do random  */
            /* slot access  */
            /* remember baseTimerStart is the same as transmitTimerStart */
             /*only  on the first iteration */
            spmGbl.elapsed = spmGbl.stopped - spmGbl.baseTimerStart;
            if (spmGbl.elapsed >= spmGbl.baseTicks)
            {
               spmGbl.baseTimerStart = spmGbl.stopped; /* restart base */
                                                       /* timer */
               DecrementBacklog(1);  /* decrement back log */
            }

         }
         break;
      case START_TX:
         if ((spmGbl.rf.rxFlag == 1) || (spmGbl.rf.txOn == 1)) /* channel */
                                                               /* busy */
         {
            /* other node must have used channel or packet has started */

            spmGbl.accessApproved = FALSE;
            spmGbl.phase = BUSY;
         }
         else
         {
            if ( macGbl.tpr == FALSE) /* macGbl.tpr == FALSE packet */
                                      /* cancelled */
            {
                spmGbl.accessApproved = FALSE; /* deny access */
                spmGbl.phase = RANDOM_IDLE;    /* back to idle */
            }
            else
            {

               spmGbl.phase = START_TX;
            }

            /* check base timer on more time in case expires while waiting
               for txOn. only if non priority slot access */

            spmGbl.stopped = *(spmGbl.clock);
            spmGbl.elapsed = spmGbl.stopped - spmGbl.baseTimerStart;
            if (spmGbl.elapsed >= spmGbl.baseTicks)
            {
               spmGbl.baseTimerStart = spmGbl.stopped; /* restart base */
                                                       /*timer */
               DecrementBacklog(1);  /* decrement back log */
            }

         }
         break;
      default:
         spmGbl.phase = RANDOM_IDLE;
         break;
   }  /* End access algo switch */


   /* Cycle Timer only runs when the Mac Layer is Idle that is when
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   the node is not tranmitting or receiving or waiting to transmit
   or counting down beta1 or counting down priority slots or counting
   down the extra 16 beta2 slots after a transmission
   This directly corresponds to the Phase RANDOM_IDLE

   Anytime the MAC layer is busy ie in some phase other than RANDOM_IDLE
   then the cycle timer stops. The cycle timer will then resume when the
   mac layer returns to the RANDOM_IDLE phase. The cycle timer is reset
   when spmGbl.cycleTimerRestart == TRUE. This is set to true
   1) following a successful transmission
   2) following a valid crc receive

   We use a time difference between the current value of the hardware
   timer and the stored value when we started the cycleTimer to determine
   when the cycleTimer expires.
   Since the hardware timer is free running and does not stop when
   the cycle timer stops , we must keep shifting the
   cycleTimerStart forward whenever the cycleTimer is stopped. This keeps
   the relative time difference correct so that when it resumes
   the expiration calculation will be valid.
*/

spmGbl.stopped = *(spmGbl.clock);
if (  (spmGbl.phase == RANDOM_IDLE) ) /* cycle timer is running */
{
   spmGbl.elapsed = spmGbl.stopped - spmGbl.cycleTimerStart;

   if (spmGbl.cycleTimerRestart) /* cycle timer needs to be restarted */
   {
      spmGbl.cycleTimerStart = spmGbl.stopped;
      spmGbl.cycleTimerRestart = FALSE;
   }
   else if (spmGbl.elapsed >= spmGbl.cycleTicks)
   {
      spmGbl.cycleTimerStart = spmGbl.stopped; /* restart */
      DecrementBacklog(1);
   }
}
else  /* shift cycleTimerStart forward since cycle timer stopped */
{
   spmGbl.cycleTimerStart += spmGbl.stopped - spmGbl.lastTime;
}
spmGbl.lastTime = spmGbl.stopped; /* save last stopped value */

/*************************************************************
  End Channel Access Algorithm State machine
  ******************************************/

/*************************************************
  Begin SPM Transfer State Machine
  ******************************************/
/* state machine for handshaking with xcvr*/
switch ( spmGbl.state)
{

   case IDLE:
      if ( spmGbl.rf.rxFlag == 1)   /* XCVR has detected packet to */
                                    /* receive */
      {
         macGbl.rc = 0; /* init receive byte count */
         macGbl.rl = 0;
         if ( macGbl.rpr == TRUE )  /* last packet not copied out */
         {
            spmGbl.mode =  OVERWRITE; /* stepped on last packet */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      macGbl.rpr = FALSE;/* what else to do ? */
   }

   /* initialize  crc stuff */
   crcGbl.poly = 0x1021;
   crcGbl.crc = 0xffff;
   crcGbl.crcBit = 0;
   crcGbl.dataBit = 0;
   crcGbl.dataByte = 0;


   /*
     spmGbl.tf.txFlag = 0;
     spmGbl.tf.txReqFlag = 0;
     spmGbl.tf.txDataValid = 0;
     spmGbl.tf.blank = 0;
     spmGbl.tf.txAddrRW = 0;
     spmGbl.tf.txAddr = 0;
     spmGbl.tf.data = 0;
     */

   *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */
   spmGbl.state = RECEIVE;
}
else if ( spmGbl.rf.txOn == 1 ) /*  XCVR xmitting on network so  */
                                /*  don't do anything            */
{
   /*
     spmGbl.tf.txFlag = 0;
     spmGbl.tf.txReqFlag = 0;
     spmGbl.tf.txDataValid = 0;
     spmGbl.tf.blank = 0;
     spmGbl.tf.txAddrRW = 0;
     spmGbl.tf.txAddr = 0;
     spmGbl.tf.data = 0;
     */

   *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

   /* increment reset count */
   spmGbl.resetCount ++;
   if (spmGbl.resetCount >= RESET_COUNT_LIMIT)
   {
      PHYHardResetSPMXCVR();
   }

   spmGbl.state = IDLE;
}
else if (( spmGbl.writeAltPathBit) && (!spmGbl.altPathBitWritten))
{
   /* need to write alt path bit unacked write */
   /*
     spmGbl.tf.txFlag = 0;
     spmGbl.tf.txReqFlag = 0;
     spmGbl.tf.txDataValid = 0;
     spmGbl.tf.blank = 0;
     spmGbl.tf.txAddrRW = 0;
     spmGbl.tf.txAddr = 1;
     spmGbl.tf.data = spmGbl.configData[0];
     */

   *((uint8 *)&(spmGbl.tf)) = (uint8) 1;  /* fast way */
   if (macGbl.altPathBit == 1)
   {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                  *((uint8 *)&(spmGbl.tf.data)) =
                     (uint8) ( 0x80 | spmGbl.configData[0]);  /* fast way */
               }
               else
               {
                  *((uint8 *)&(spmGbl.tf.data)) = (uint8) ( 0x7F &

spmGbl.configData[0]);
               }
               spmGbl.altPathBitWritten = TRUE;
               spmGbl.state = IDLE;
            }
            else if ((macGbl.tpr == TRUE) && /* transmit packet ready */
                     (spmGbl.accessApproved == TRUE)) /* channel access */
                                                      /* approved */
            {
               /*
                 spmGbl.tf.txFlag = 0;
                 spmGbl.tf.txReqFlag = 1;
                 spmGbl.tf.txDataValid = 0;
                 spmGbl.tf.blank = 0;
                 spmGbl.tf.txAddrRW = 0;
                 spmGbl.tf.txAddr = 0;
                 spmGbl.tf.data = 0;
                 */

               *((uint16 *)&(spmGbl.tf)) = (uint16) 0x4000U;  /* fast way */

               macGbl.tc = 0;
               spmGbl.state = REQ_TX;
            }

            else if ( (spmGbl.crw == TRUE) &&  /* config register write */
                      (spmGbl.cra > 0) &&      /* 0< config reg address <=7 */
                      (spmGbl.cra <= 7) )
            {
               /*
                 spmGbl.tf.txFlag = 0;
                 spmGbl.tf.txReqFlag = 0;
                 spmGbl.tf.txDataValid = 0;
                 spmGbl.tf.blank = 0;
                 spmGbl.tf.txAddrRW = 0;
                 spmGbl.tf.txAddr = spmGbl.cra;
                 spmGbl.tf.data = spmGbl.crData;
                 */

               *((uint8 *)&(spmGbl.tf)) = (uint8) spmGbl.cra;  /* fast way */
               *((uint8 *)&(spmGbl.tf.data)) = (uint8) spmGbl.crData; /*fast */
                                                                      /* way */

               spmGbl.state = WRITE;
            }
            else if ( (spmGbl.srr == TRUE) &&   /* status register read */
                      (spmGbl.sra >= 0) &&      /* 0< status reg address <=7 */
                      (spmGbl.sra <= 7) )
            {
               spmGbl.tf.txFlag = 0;
               spmGbl.tf.txReqFlag = 0;
               spmGbl.tf.txDataValid = 0;
               spmGbl.tf.blank = 0;
               spmGbl.tf.txAddrRW = 1;
               spmGbl.tf.txAddr = spmGbl.sra;
               spmGbl.tf.data = 0;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                  spmGbl.state = READ;
               }
               else   /* Nothing can be done just wait */
               {
                  /*
                    spmGbl.tf.txFlag = 0;
                    spmGbl.tf.txReqFlag = 0;
                    spmGbl.tf.txDataValid = 0;
                    spmGbl.tf.blank = 0;
                    spmGbl.tf.txAddrRW = 0;
                    spmGbl.tf.txAddr = 0;
                    spmGbl.tf.data = 0;
                    */

                  *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */
                  spmGbl.state = IDLE;
               }
               break;
            case RECEIVE:
               /*
                 spmGbl.tf.txFlag = 0;
                 spmGbl.tf.txReqFlag = 0;
                 spmGbl.tf.txDataValid = 0;
                 spmGbl.tf.blank = 0;
                 spmGbl.tf.txAddrRW = 0;
                 spmGbl.tf.txAddr = 0;
                 spmGbl.tf.data = 0;
                 */

               *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

               if (spmGbl.rf.rxFlag == 0)  /* packed ended */
               {
                  macGbl.rl = macGbl.rc; /* set last byte count */
                  /* is packet of valid length */
                  if ( (macGbl.rl >= 8) && /* minimum valid packet is 8 bytes */
                       (macGbl.rl < PKT_BUF_LEN) ) /* must be less than buffer */
                  {
                     /* do last step of crc */
                     crcGbl.crc = crcGbl.crc ^ 0xffff;


                     /* is crc valid */
                     if ( crcGbl.crc == (*((uint16 *) &macGbl.rPkt[macGbl.rl -
2])) )
                     {
                        /* copy out backlog */
                        macGbl.deltaBLRx = macGbl.rPkt[0] & 0x3F;

                        /* update cycle timer and set flag */
                        if (macGbl.deltaBLRx <= 0)
                        {
                           DecrementBacklog(1);
                        }
                        else
                        {
                           IncrementBacklog(macGbl.deltaBLRx);
                        }
                        /* Cycle Timer needs to be restarted after valid crc rx */
                        spmGbl.cycleTimerRestart = TRUE;

                        /* Attempt to copy packet into Link Layer receive Queue */
                        if ( (*gp->lkInQTailPtr == 0) && /* == 0 means free     */
                                                /* buffer big enough for packet  */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                    (gp->lkInBufSize >= macGbl.rl + 3) )
                {
                  /* Copy the packet to link layer input queue */
                   *(uint16 *)(gp->lkInQTailPtr + 1) = (uint16) macGbl.rl;
                                                      /* copy length */
                   memcpy(gp->lkInQTailPtr + 3, macGbl.rPkt, macGbl.rl);
                   /* Update Link Layer Input Queue Fields */
                   *gp->lkInQTailPtr = 1; /*lets link layer know packet */
                                        /* is ready to process        */
                   gp->lkInQTailPtr += gp->lkInBufSize;
                   if (gp->lkInQTailPtr ==
                       gp->lkInQ + gp->lkInBufSize * gp->lkInQCnt)
                   {/* past end of queue */
                      gp->lkInQTailPtr = gp->lkInQ; /* wrap to */
                                                    /* beginning */
                   }
                }
                else /* packet discarded */
                {
                   /* Discard packet as either there is no buffer */
                   /* available or it is too big for buffer */
                   INCR_STATS(nmp->stats.missedMessages); /* update */
                                                          /*stats */
                }
             }
             else /* invalid crc */
             {
                /* update stats for bad crc packet */
                INCR_STATS(nmp->stats.transmissionErrors);
             }
          }
          else /* update stats invalid packet length */
          {
             INCR_STATS(nmp->stats.transmissionErrors);
          }

          /* Reset macGbl's receive packet fields */
          macGbl.rc = 0; /* reinit to be safe */
          macGbl.rl = 0;
          macGbl.rpr = FALSE;  /* ready to receive another packet */

          spmGbl.state = IDLE; /*  all done */
       }
       else if (spmGbl.rf.rxDataValid  == 1)   /* valid byte this frame */
       {
          if(macGbl.rc < PKT_BUF_LEN) /* don't go past end of buffer */
          {
             macGbl.rPkt[macGbl.rc] = spmGbl.rf.data;  /* copy out byte */
             macGbl.rc++; /* increment counter */

             /* do 1 byte of crc. Delay by two bytes so we do not computer
                crc on the two crc bytes in the packet itself */
             if (macGbl.rc >= 3)
             {
                /* make copy of byte */
                crcGbl.dataByte = macGbl.rPkt[macGbl.rc - 3];
                for (j = 0; j < 8; j++)
                {
                   crcGbl.crcBit = (crcGbl.crc & 0x8000) ? 1 : 0;
                   crcGbl.dataBit = (crcGbl.dataByte & 0x80) ? 1 : 0;
                   crcGbl.dataByte = crcGbl.dataByte << 1;
                   crcGbl.crc = crcGbl.crc << 1;
                   if ( crcGbl.crcBit != crcGbl.dataBit)
                   {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                    crcGbl.crc = crcGbl.crc ^ crcGbl.poly;
                }
            }
        }
    }


    spmGbl.state = RECEIVE; /* look for more */
  }
  else  /* no valid data but still receiving */
  {
    spmGbl.state = RECEIVE; /* keep looking */
  }
  break;
case WRITE:
  if ( spmGbl.rf.rxFlag == 1)  /* XCVR has detected packet to */
                               /* receive */
  {
    macGbl.rc = 0; /* init receive byte count */
    macGbl.rl = 0;
    if ( macGbl.rpr == TRUE )  /* last packet not copyied out */
    {
      spmGbl.mode =  OVERWRITE; /* stepped on last packet */
      macGbl.rpr = FALSE;/* what else to do ? */
    }

    /* initialize  crc stuff */
    crcGbl.poly = 0x1021;
    crcGbl.crc = 0xffff;
    crcGbl.crcBit = 0;
    crcGbl.dataBit = 0;
    crcGbl.dataByte = 0;

    /*
      spmGbl.tf.txFlag = 0;
      spmGbl.tf.txReqFlag = 0;
      spmGbl.tf.txDataValid = 0;
      spmGbl.tf.blank = 0;
      spmGbl.tf.txAddrRW = 0;
      spmGbl.tf.txAddr = 0;
      spmGbl.tf.data = 0;
      */

    *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

    spmGbl.state = RECEIVE;
  }
  else if ((spmGbl.rf.rwAck == 0) &&  /* still not written */
           (spmGbl.crw == TRUE) &&
           (spmGbl.cra > 0) &&            /* 0< config reg address */
                                          /* <=7 */
           (spmGbl.cra <= 7) )
  {
    /*
      spmGbl.tf.txFlag = 0;
      spmGbl.tf.txReqFlag = 0;
      spmGbl.tf.txDataValid = 0;
      spmGbl.tf.blank = 0;
      spmGbl.tf.txAddrRW = 0;
      spmGbl.tf.txAddr = spmGbl.cra;
      spmGbl.tf.data = spmGbl.crData;
      */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      *((uint8 *)&(spmGbl.tf)) = (uint8) spmGbl.cra;  /* fast way */
      *((uint8 *)&(spmGbl.tf.data)) = (uint8) spmGbl.crData; /*fast */
                                                             /* way */

      spmGbl.state = WRITE; /* try again */
   }
   else  /* write successful or recalled */
   {
      spmGbl.crw = FALSE;

      /*
        spmGbl.tf.txFlag = 0;
        spmGbl.tf.txReqFlag = 0;
        spmGbl.tf.txDataValid = 0;
        spmGbl.tf.blank = 0;
        spmGbl.tf.txAddrRW = 0;
        spmGbl.tf.txAddr = 0;
        spmGbl.tf.data = 0;
        */

      *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */


      spmGbl.state = IDLE;
   }
   break;
case READ:
   if ( spmGbl.rf.rxFlag == 1) /* XCVR has detected packet to */
                               /* receive */
   {
      macGbl.rc = 0; /* init receive byte count */
      macGbl.rl = 0;
      if ( macGbl.rpr == TRUE )  /* last packet not copied out */
      {
         spmGbl.mode =  OVERWRITE; /* stepped on last packet */
         macGbl.rpr = FALSE;/* what else to do ? */
      }

      /* initialize  crc stuff */
      crcGbl.poly = 0x1021;
      crcGbl.crc = 0xffff;
      crcGbl.crcBit = 0;
      crcGbl.dataBit = 0;
      crcGbl.dataByte = 0;

      /*
        spmGbl.tf.txFlag = 0;
        spmGbl.tf.txReqFlag = 0;
        spmGbl.tf.txDataValid = 0;
        spmGbl.tf.blank = 0;
        spmGbl.tf.txAddrRW = 0;
        spmGbl.tf.txAddr = 0;
        spmGbl.tf.data = 0;
        */

      *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

      spmGbl.state = RECEIVE;
   }
   else if ((spmGbl.rf.rwAck == 0) &&  /* still not read */
            (spmGbl.srr == TRUE) &&
            (spmGbl.sra >= 0) &&           /* 0< config reg address */
                                           /* <=7 */
            (spmGbl.sra <= 7) )
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    {

       spmGbl.tf.txFlag = 0;
       spmGbl.tf.txReqFlag = 0;
       spmGbl.tf.txDataValid = 0;
       spmGbl.tf.blank = 0;
       spmGbl.tf.txAddrRW = 1;
       spmGbl.tf.txAddr = spmGbl.sra;
       spmGbl.tf.data = 0;

       spmGbl.state = READ;  /* keep trying */
    }
    else if ( spmGbl.rf.rwAck == 1)
    {
       spmGbl.srData = spmGbl.rf.data;
       spmGbl.srr = FALSE;

       /*
         spmGbl.tf.txFlag = 0;
         spmGbl.tf.txReqFlag = 0;
         spmGbl.tf.txDataValid = 0;
         spmGbl.tf.blank = 0;
         spmGbl.tf.txAddrRW = 0;
         spmGbl.tf.txAddr = 0;
         spmGbl.tf.data = 0;
         */

       *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */
       spmGbl.state = IDLE;
    }
    else
    {
       /*
         spmGbl.tf.txFlag = 0;
         spmGbl.tf.txReqFlag = 0;
         spmGbl.tf.txDataValid = 0;
         spmGbl.tf.blank = 0;
         spmGbl.tf.txAddrRW = 0;
         spmGbl.tf.txAddr = 0;
         spmGbl.tf.data = 0;
         */

       *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */
       spmGbl.state = IDLE;
    }
    break;
 case REQ_TX:
    if ( spmGbl.rf.rxFlag == 1   /* XCVR has detected packet to */
                                 /* receive */
    {
       macGbl.rc = 0; /* init receive byte count */
       macGbl.rl = 0;
       if ( macGbl.rpr == TRUE )  /* last packet not copied out */
       {
          spmGbl.mode =  OVERWRITE; /* stepped on last packet */
          macGbl.rpr = FALSE;/* what else to do ? */
       }

       /* initialize  crc stuff */
       crcGbl.poly = 0x1021;
       crcGbl.crc = 0xffff;
       crcGbl.crcBit = 0;
       crcGbl.dataBit = 0;
       crcGbl.dataByte = 0;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

```
      /*
        spmGbl.tf.txFlag = 0;
        spmGbl.tf.txReqFlag = 0;
        spmGbl.tf.txDataValid = 0;
        spmGbl.tf.blank = 0;
        spmGbl.tf.txAddrRW = 0;
        spmGbl.tf.txAddr = 0;
        spmGbl.tf.data = 0;
        */

      *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

      spmGbl.state = RECEIVE;
    }
    else if (spmGbl.rf.clrTxReqFlag == 1) /* either accept or deny */
    {
      if ( (spmGbl.rf.setTxFlag == 1) && /* request accepted */
           (spmGbl.rf.txDataCTS == 1) )
      {
        /*
        spmGbl.tf.txFlag = 1;
        spmGbl.tf.txReqFlag = 0;
        spmGbl.tf.txDataValid = 1;
        spmGbl.tf.blank = 0;
        spmGbl.tf.txAddrRW = 0;
        spmGbl.tf.txAddr = 0;
        spmGbl.tf.data =  macGbl.tPkt[macGbl.tc];
        */

        *((uint8 *)&(spmGbl.tf)) = (uint8) 0xA0;  /* fast way */
        *((uint8 *)&(spmGbl.tf.data)) = (uint8)
                                          macGbl.tPkt[macGbl.tc];

        macGbl.tc++; /* increment count */

        spmGbl.state = TRANSMIT; /* accepted */
      }
      else  /* request denied */
      {
        /*
        spmGbl.tf.txFlag = 0;
        spmGbl.tf.txReqFlag = 0;
        spmGbl.tf.txDataValid = 0;
        spmGbl.tf.blank = 0;
        spmGbl.tf.txAddrRW = 0;
        spmGbl.tf.txAddr = 0;
        spmGbl.tf.data = 0;
        */

        *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

        spmGbl.state = IDLE; /* dont want to tx anymore */

      }
    }
    else  /* wait for xcvr to respond to request */
    {
      /* typically there is one blank frame from xcvr
         after REQ_TX bit is set but before CTS or ClrTxReqFlag
*/

      /*
        spmGbl.tf.txFlag = 0;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            spmGbl.tf.txReqFlag = 0;
            spmGbl.tf.txDataValid = 0;
            spmGbl.tf.blank = 0;
            spmGbl.tf.txAddrRW = 0;
            spmGbl.tf.txAddr = 0;
            spmGbl.tf.data = 0;
            */

         *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

         spmGbl.state = REQ_TX; /* Wait for response */
      }
      break;
   case TRANSMIT:
      if ( spmGbl.rf.setCollDet == 1)  /* collision so start over */
      {
         /*
           spmGbl.tf.txFlag = 0;
           spmGbl.tf.txReqFlag = 0;
           spmGbl.tf.txDataValid = 0;
           spmGbl.tf.blank = 0;
           spmGbl.tf.txAddrRW = 0;
           spmGbl.tf.txAddr = 0;
           spmGbl.tf.data = 0;
           */

         *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

         macGbl.tc = 0;

         IncrementBacklog(1);  /* collision so increment backlog */

         spmGbl.collisionsThisPkt++; /*increment collison count this */
                                     /* packet*/

         INCR_STATS(nmp->stats.collisions);/* increment collision */
                                          /* stats */

         if (spmGbl.collisionsThisPkt >= 255)
         {
            /* throw away packet */
            macGbl.tc = 0;
            spmGbl.resetCount = 0; /* reset count for txOn */
            spmGbl.collisionsThisPkt = 0;
            macGbl.tpr = FALSE;
         }

         spmGbl.state = IDLE; /* start over */
      }
      else if (spmGbl.rf.txDataCTS == 1)
      {
         /*
           spmGbl.tf.txFlag = 1;
           spmGbl.tf.txReqFlag = 0;
           spmGbl.tf.txDataValid = 1;
           spmGbl.tf.blank = 0;
           spmGbl.tf.txAddrRW = 0;
           spmGbl.tf.txAddr = 0;
           spmGbl.tf.data =  macGbl.tPkt[macGbl.tc];
           */

         *((uint8 *)&(spmGbl.tf)) = (uint8) 0xA0;  /* fast way */
         *((uint8 *)&(spmGbl.tf.data)) = (uint8) macGbl.tPkt[macGbl.tc];
                                                 /* fast way */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            macGbl.tc++; /* increment count */

            if( macGbl.tc >= macGbl.tl) /* last byte sent this frame */
            {
               spmGbl.state = DONE_TX; /* dont want to tx anymore */
            }
            else
            {
               spmGbl.state = TRANSMIT;
            }
         }
         else if ((spmGbl.rf.txOn == 1) &&
                  (macGbl.tpr == TRUE) &&
                  (macGbl.tc < macGbl.tl) ) /* wait for cts */
         {
            /*
              spmGbl.tf.txFlag = 1;
              spmGbl.tf.txReqFlag = 0;
              spmGbl.tf.txDataValid = 0;
              spmGbl.tf.blank = 0;
              spmGbl.tf.txAddrRW = 0;
              spmGbl.tf.txAddr = 0;
              spmGbl.tf.data =  0;
              */

            *((uint8 *)&(spmGbl.tf)) = (uint8) 0x80;  /* fast way */
            *((uint8 *)&(spmGbl.tf.data)) = (uint8) 0;  /* fast way */

            spmGbl.resetCount++;
            if (spmGbl.resetCount >= RESET_COUNT_LIMIT)
            {
               PHYHardResetSPMXCVR();
               spmGbl.state = IDLE;
            }
            else
            {
               spmGbl.state = TRANSMIT;
            }
         }
         else /* something wrong here throw away packet and get out */
         {
            /*
              spmGbl.tf.txFlag = 0;
              spmGbl.tf.txReqFlag = 0;
              spmGbl.tf.txDataValid = 0;
              spmGbl.tf.blank = 0;
              spmGbl.tf.txAddrRW = 0;
              spmGbl.tf.txAddr = 0;
              spmGbl.tf.data = 0;
              */

            *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

            macGbl.tc = 0;
            spmGbl.resetCount = 0; /* reset count for txOn */
            spmGbl.collisionsThisPkt = 0;
            macGbl.tpr = FALSE;
            spmGbl.state = IDLE;
         }
         break;
      case DONE_TX: /* wait for tx_on to go off */
         if ( spmGbl.rf.setCollDet == 1)  /* collision so start over */
         {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      /*
        spmGbl.tf.txFlag = 0;
        spmGbl.tf.txReqFlag = 0;
        spmGbl.tf.txDataValid = 0;
        spmGbl.tf.blank = 0;
        spmGbl.tf.txAddrRW = 0;
        spmGbl.tf.txAddr = 0;
        spmGbl.tf.data = 0;
        */

      *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

      macGbl.tc = 0;

      IncrementBacklog(1);  /* collision so increment backlog */

      spmGbl.collisionsThisPkt++; /* increment collision count   */
                                  /* this packet                */

      INCR_STATS(nmp->stats.collisions);/* increment collision */
                                       /* stats */

      if (spmGbl.collisionsThisPkt >= 255)
      {
         /* throw away packet */
         macGbl.tc = 0;
         spmGbl.resetCount = 0; /* reset count for txOn */
         spmGbl.collisionsThisPkt = 0;
         macGbl.tpr = FALSE;
      }

      spmGbl.state = IDLE; /* start over */
   }
   else if (spmGbl.rf.txOn == 1) /* still sending last byte(s) */
   {
      /*
        spmGbl.tf.txFlag = 0;
        spmGbl.tf.txReqFlag = 0;
        spmGbl.tf.txDataValid = 0;
        spmGbl.tf.blank = 0;
        spmGbl.tf.txAddrRW = 0;
        spmGbl.tf.txAddr = 0;
        spmGbl.tf.data =  0;
        */

      *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0;  /* fast way */

      spmGbl.resetCount++;
      if (spmGbl.resetCount >= RESET_COUNT_LIMIT)
      {
         PHYHardResetSPMXCVR();
         spmGbl.state = IDLE;
      }
      else
      {
         spmGbl.state = DONE_TX;
      }
   }
   else /* (spmGbl.rf.txOn == 0) done sending */
   {
      /*
        spmGbl.tf.txFlag = 0;
        spmGbl.tf.txReqFlag = 0;
        spmGbl.tf.txDataValid = 0;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                  spmGbl.tf.blank = 0;
                  spmGbl.tf.txAddrRW = 0;
                  spmGbl.tf.txAddr = 0;
                  spmGbl.tf.data =  0;
                  */

               *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0;  /* fast way */

               /* successful transmission so update backlog */
               if (macGbl.deltaBLTx <= 0)
               {
                  DecrementBacklog(1);
               }
               else
               {
                  IncrementBacklog(macGbl.deltaBLTx);
               }

               /* Cycle Timer needs to be restarted after valid tx  */

               spmGbl.cycleTimerRestart = TRUE;


               /* clean up */
               macGbl.tc = 0;
               spmGbl.resetCount = 0; /* reset count for txOn */
               spmGbl.collisionsThisPkt = 0;
               macGbl.tpr = FALSE;
               spmGbl.state = IDLE; /* dont want to tx anymore */
            }
            break;
         case DEBUG:
         default:
            /*
              spmGbl.tf.txFlag = 0;
              spmGbl.tf.txReqFlag = 0;
              spmGbl.tf.txDataValid = 0;
              spmGbl.tf.blank = 0;
              spmGbl.tf.txAddrRW = 0;
              spmGbl.tf.txAddr = 0;
              spmGbl.tf.data = 0;
              */

            *((uint16 *)&(spmGbl.tf)) = (uint16) 0x0U;  /* fast way */

            spmGbl.state = DEBUG;
      }       /* end switch spm state machine */



#ifdef SPM_HISTORY
         hBufGbl.records[hBufGbl.index].tf = spmGbl.tf;
         /* time period of  isr */
         /* C unsigned math is congruent mod 2^n so negatives wrap around OK */
         hBufGbl.records[hBufGbl.index].duration = *(spmGbl.clock) -
            hBufGbl.records[hBufGbl.index].start;

         hBufGbl.index = (hBufGbl.index + 1) % NUM_HIST;
#endif
         /* Send Frame */
         tbdPtrGbl->r = 1;         /* buffer ready to transmit */
         spcomPtrGbl->str = 1;
      } /* end if spmGbl.mode == STOP */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }  /* endif rxb event */

   cisrPtrGbl->spi = 1; /* clear spi bit in CISR by writing a 1 */
   return;
}

/*****************************************************************
Function:  CErrIsr
Returns:   none, no arguments allowed either
Reference:
Purpose:   interrupt service routine for CPM Int. Error
Comments:  must use interrupt pragma
*****************************************************************/
/*  interrupt pragma informs compiler to make next function
        an interrupt handler with RTE and saved state of registers */

#pragma interrupt()

static void CErrIsr(void)
{

   return;
}

/*****************************************************************
Function:  SpurIntIsr
Returns:   none, no arguments allowed either
Reference:
Purpose:   interrupt service routine for spurious interrupt Error
Comments:  must use interrupt pragma
*****************************************************************/
/*  interrupt pragma informs compiler to make next function
        an interrupt handler with RTE and saved state of registers */

#pragma interrupt()

static void SpurIntIsr(void)
{
   exceptions++;

   return;
}

/*****************************************************************
Function:  BusErrIsr
Returns:   none, no arguments allowed either
Reference:
Purpose:   interrupt service routine for Bus errors
Comments:  must use interrupt pragma
*****************************************************************/
/*  interrupt pragma informs compiler to make next function
        an interrupt handler with RTE and saved state of registers */

#pragma interrupt()

static void BusErrIsr(void)
{
   exceptions++;

   return;
}

/*****************************************************************
Function:  GetTransceiverStatus
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Returns:   none
Purpose:   To determine the status of transceiver registers.
           The status registers are different from config registers.
Comments:  The array should have space for all the register values.
*************************************************************/
void GetTransceiverStatus(Byte transceiverStatusOut[])
{
   int i;
   unsigned long count;
   Boolean good;

   /* read config registers on PLT-20 transceiver */

   /* read in reverse order */
   i = NUM_COMM_PARAMS - 1; /* should be 6 */
   while ( i >= 0)
   {
      /* read status registers starting at 7 down to 1 */
      spmGbl.sra = (Byte) i + 1;
      spmGbl.srr = TRUE;
      count = 0;
      /* wait for ack but time out so not infinite loop */
      while ((spmGbl.srr != FALSE) && (count < 0xFFFFF))
      {
         count++;
      }
      if (spmGbl.srr == TRUE) /* xcvr never acked try again */
      {
         pbdatPtrGbl->d5   = 0; /* first reset xcvr */

         /* wait for a while for everything to settle down */
         good = DelayTicks(RESTART_DELAY_TICKS);


         pbdatPtrGbl->d5   = 1; /* bring xcvr out of reset */
         i = NUM_COMM_PARAMS - 1; /* start over again */
      } /* NOTE!!! if the xcvr never acks this code will hang */
      else
      {
         transceiverStatusOut[i] = spmGbl.srData;
         i--;
      }
   }

   spmGbl.srData = 0;
   spmGbl.sra = 0;


   return;

}

/************************End of spm.c************************/
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.3 LPDU Transmit Algorithm

```
/*****************************************************************************
Function:  LKSend
Returns:   None
Reference: None
Purpose:   To take the NPDU from link layer's output queue and put it
           in the queue for the physical layer.
Comments:  We assume that there will be sufficient space as we
           allocated the extra bytes based on header size etc.
*****************************************************************************/
void LKSend(void)
{
   LKSendParam     *lkSendParamPtr;
   Queue           *lkSendQueuePtr;
   Byte            *npduPtr;
   Byte            *phyQItemPtr, *phyQPtr, *tempPtr;
   Byte           **phyQTailPtrPtr;
   uint16           queueCnt, itemSize;
   uint16          *pduSizePtr;
   LPDUHeader      *lpduHeaderPtr;
   Boolean          priority;

   /* First, make variables point to the right queue. */
   if (!QueueEmpty(&gp->lkOutPriQ))
   {
      priority        = TRUE;
      lkSendQueuePtr  = &gp->lkOutPriQ;
      phyQPtr         = gp->phyOutPriQ;
      phyQItemPtr     = gp->phyOutPriQTailPtr;
      phyQTailPtrPtr  = &gp->phyOutPriQTailPtr;
      queueCnt        = gp->phyOutPriQCnt;
      itemSize        = gp->phyOutPriBufSize;
   }
   else if (!QueueEmpty(&gp->lkOutQ))
   {
      priority        = FALSE;
      lkSendQueuePtr  = &gp->lkOutQ;
      phyQPtr         = gp->phyOutQ;
      phyQItemPtr     = gp->phyOutQTailPtr;
      phyQTailPtrPtr  = &gp->phyOutQTailPtr;
      queueCnt        = gp->phyOutQCnt;
      itemSize        = gp->phyOutBufSize;
   }
   else
   {
      return; /* Nothing to send. */
   }

   lkSendParamPtr = QueueHead(lkSendQueuePtr);
   npduPtr        = (Byte *) (lkSendParamPtr + 1);

   /* Check if there is space in the physical layer's queue. */
   if (*phyQItemPtr == 1)
   {
      return; /* No space in the physical layer queue. */
   }

   /* Fill the PDU size so that the physical layer knows about it. */
   pduSizePtr  = (uint16 *) ((char *)phyQItemPtr + 1);
   *pduSizePtr = lkSendParamPtr->pduSize + 3;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* First fill the data. */
   tempPtr = phyQItemPtr + 3;

   /* Form the header. */
   lpduHeaderPtr           = (LPDUHeader *)tempPtr;
   lpduHeaderPtr->priority = priority;
   lpduHeaderPtr->altPath  = lkSendParamPtr->altPath;
   lpduHeaderPtr->deltaBL  = lkSendParamPtr->deltaBL;
   tempPtr++;

   /* Copy the NPDU. */
   if (lkSendParamPtr->pduSize <= itemSize)
   {
      memcpy(tempPtr, npduPtr, lkSendParamPtr->pduSize);
   }
   else
   {
      ErrorMsg("LKSend: NPDU seems too large to fit.\n");
   }

   /* Compute the CRC value. */
   CRC16(phyQItemPtr+3, lkSendParamPtr->pduSize + 1);

   /* Turn the flag on so that physical layer can send it. */
   *phyQItemPtr = 1;

   /* Increment tail pointer taking care of wraparound. */
   *phyQTailPtrPtr = *phyQTailPtrPtr + itemSize;
   if (*phyQTailPtrPtr == (phyQPtr + queueCnt * itemSize))
   {
      *phyQTailPtrPtr = phyQPtr; /* wrap around. */
   }

   DeQueue(lkSendQueuePtr);

   return;
}

/*****************************************************************************
Function:  CRC16
Returns:   16 bit CRC computed.
Purpose:   To compute the 16 bit CRC for a given buffer.
Comments:  None.
*****************************************************************************/
void CRC16(Byte bufInOut[], uint16 sizeIn)
{
   uint16 poly = 0x1021;       /* Generator Polynomial. */
   uint16 crc = 0xffff;
   uint16 i,j;
   unsigned char byte, crcbit, databit;

   for (i = 0; i < sizeIn; i++)
   {
      byte = bufInOut[i];
      for (j = 0; j < 8; j++)
      {
        crcbit = crc & 0x8000 ? 1 : 0;
        databit = byte & 0x80 ? 1 : 0;
        crc = crc << 1;
        if (crcbit != databit)
         {
           crc = crc ^ poly;
         }
        byte = byte << 1;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      }

   }
   crc = crc ^ 0xffff;
   bufInOut[sizeIn]     = (crc >> 8);
   bufInOut[sizeIn + 1] = (crc & 0x00FF);

   return;
}
```

A.4 LPDU Receive Algorithm

```
/*****************************************************************************

Function:  LKReceive

Returns:   None

Reference: None

Purpose:   To receive the incoming LPDUs and process them.

Comments:  Each item of the queue gp->lkInQ has the following form:

           flag pduSize LPDU

           flag is 1 byte long.

           pduSize is 2 bytes long.

           LPDU has header followed by the rest of the LPDU and then CRC.

           The LPDU header is 1 byte long. CRC uses 2 bytes.

           If a packet is in lkInQ then it should fit into nwInQ.

*****************************************************************************/

void LKReceive(void)

{

   NWReceiveParam *nwReceiveParamPtr;

   Byte           *npduPtr;

   LPDUHeader     *lpduHeaderPtr;

   Byte           *tempPtr;

   uint16          lpduSize;



   if (*(gp->lkInQHeadPtr) == 0)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
{
   /* There is nothing to receive. */
   return;
}
lpduSize = *(uint16 *)(gp->lkInQHeadPtr + 1);
lpduHeaderPtr     = (LPDUHeader *)(gp->lkInQHeadPtr + 3);

/* Throw away packets that are smaller than 8 bytes long. */
/* this check is now  made in the mac sublayer */
/* Do CRC check. */
/* this check is now made in the mac sublayer */
/* Only packets with valid CRC and >= 8 bytes are placed
   in the lkInQ by mac sublayer. */

INCR_STATS(nmp->stats.layer2Received); /* Got a good packet. */




/* We need to receive this message. */
if (QueueFull(&gp->nwInQ))
{
   /* We are losing this packet. */
   INCR_STATS(nmp->stats.missedMessages);
}
else
{
   nwReceiveParamPtr = QueueTail(&gp->nwInQ);
   npduPtr           = (Byte *)(nwReceiveParamPtr + 1);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      nwReceiveParamPtr->priority = lpduHeaderPtr->priority;
      nwReceiveParamPtr->altPath  = lpduHeaderPtr->altPath;
      tempPtr = (Byte *)((char *)lpduHeaderPtr + 1);
      nwReceiveParamPtr->pduSize  = lpduSize - 3;

      /* Copy the NPDU. */
      /* if it was in link layer's queue, then the size should be
         sufficient in network layer's queue as they differ by 3.
         However, let us play safe by checking the size first. */
      if (nwReceiveParamPtr->pduSize <= gp->nwInBufSize)
      {
         memcpy(npduPtr, tempPtr, nwReceiveParamPtr->pduSize);
      }
      else
      {
         ErrorMsg("LKReceive: NPDU size seems too large.\n");
      }
      EnQueue(&gp->nwInQ);
   }
   *(gp->lkInQHeadPtr) = 0;
   gp->lkInQHeadPtr = gp->lkInQHeadPtr + gp->lkInBufSize;
   if (gp->lkInQHeadPtr ==
       (gp->lkInQ + gp->lkInBufSize * gp->lkInQCnt))
   {
      gp->lkInQHeadPtr = gp->lkInQ; /* wrap around. */
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   return;

}
```

A.5 Routing Algorithm

```
Input:
   NPDU          the NPDU to be routed

Output:
   Decision       one of (Forward, Drop)

Uses:
   My_Domain      the domain this router is assigned to
   My_Subnet     the subnet within the domain that this side of the router is assigned to
   ROUTE_uc ()     routing table
   ROUTE_mc()     routing table
   ROUTE_bc()     routing table
   RouterType     one of: Configured, Learning, Bridge, Repeater

Begin { routingalgorithm  }

   If RouterType = Repeater Then Begin
      Decision := Forward;
      Return;
   end;

   If RouterType = Learning Then
      Execute ROUTING_EVENT of learning algorithm;

   If NPDU.Domain <> My_Domain Then
      If RouterType = Bridge or RouterType = Learning Then Begin
         Decision := Drop;
         Return;
      end;
      If NPDU.Domain <> Null_Domain Then Begin
         Decision := Drop;
         Return;
         end;
   Else If RouterType = Bridge Then Begin
      Decision := Forward;
      Return;
   end;

   Case NPDU.DestAddrFmt Of
      Subnet/Node:        Decision := ROUTE_uc (NPDU.DestSubnet);
      Group:           Decision := ROUTE_mc (NPDU.DestGroup);
      Broadcast:       Decision := ROUTE_bc (NPDU.DestSubnet);
   end case;

   Return;
end { routing algorithm};
```

A.6 Learning Algorithm

```
Inputs:

   INIT_EVENT
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      always occurs on system reboot; may also occur periodically, allowing the
router to adapt to changes  in network topology

   ROUTING_EVENT
      NPDU      the NPDU to be routed
      MySubnet      the subnet the router is configured on for this side of the router

Output:
   Defines routing function ROUTE_uc ()

begin { learning algorithm }
   case event of
      INIT_EVENT:
         Set ROUTE_uc () := Forward for all subnet addresses;
         Set ROUTE_uc (MySubnet) := Drop;
         Set ROUTE_mc () := Forward for all group addresses;

      ROUTING_EVENT:
         ROUTE_uc (NPDU.SrcSubnet) := Drop
   end case;
end { learning algorithm };
```

A.7 Transaction Control Algorithm

```
/*****************************************************************
          File:        tcs.c

       Version:        1.7

       Purpose:        Interface file for transaction control sublayer.
                       Outgoing sequencing.
                       Incoming sequencing and duplicate detection.

          Note:        For assigning TIds, a table is used. We
                       remember the last TID for each unique destination
                       address. When a new TId is requested for a
                       destination, this table is searched for that
                       destination. If found, we make sure that we
                       don't assign the same TId used for that
                       destination. If the destination is not
                       found, we make a new entry in the table.

                       We have an entry in the table for each
                       subnet/node, group, broadcast, subnet
                       broadcast, unique node id. When a table entry
                       is assigned, we remember the time stamp too.
                       If the table does not have space for a new
                       destination address, we get rid of one that has
                       remained more than 24 seconds. If there is no
                       such entry, then we fail to allocate the new
                       transaction ID. The table size is configurable.

         To Do:        None
*****************************************************************/

/* *** START INFORMATIVE - Transaction ID Allcation *** */
/* These functions represent an example means of allocating transaction IDs.
 * There are in fact several valid mechanisms for allocating transaction IDs.
 * In addition to the mechanism below, there are at least two other accepted means
 * for allocating transaction IDs.
 * 1. Allocate a transaction ID per unique destination address.  This method
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
 *    should not be used if acknowledged or request/response using unique ID or
 *    broadcast addressing are using in time proximity with the other addressing.
.*    modes
.*    Note that such combinations can be accomplished in this scheme if guardbands
 *    are placed around expected arrivals of acks/responses per transaction ID.
 * 2. Allocate all transaction IDs from a single transaction ID space without
 *    conflict checking.  If this simple scheme is used, it is recommended that
 *    conflict checking be performed by the application. */

/*------------------------------------------------------------------------------
Section: Includes
------------------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <cnp_1.h>
#include <node.h>
#include <tcs.h>

/*------------------------------------------------------------------
Section: Constant Definitions
------------------------------------------------------------------*/
/* Minimum amount of time in seconds a record in the priTbl or
   non-priTbl should stay before it can be replaced with a new
   entry. i.e., if the table is full and a new entry is needed,
   we look for an entry that has remained in the table for
   at least MIN_TABLE_TIME seconds. */
#define MIN_TABLE_TIME 24

/*------------------------------------------------------------------
Section: Type Definitions
------------------------------------------------------------------*/
/* None */

/*------------------------------------------------------------------
Section: Globals
------------------------------------------------------------------*/
/* None */

/*------------------------------------------------------------------
Section: Function Prototypes
------------------------------------------------------------------*/
/*****************************************************************
Function:  TCSReset
Returns:   None
Reference: Section 7, Protocol Specification.
Purpose:   To initialize all globals to proper values.
Comments:  None.
*****************************************************************/
void TCSReset(void)
{
   gp->priTransID    = 0; /* On node reset, transaction id 0 is used. */
   gp->nonpriTransID = 0;
   gp->priTransCtrlRec.inProgress    = FALSE;
   gp->nonpriTransCtrlRec.inProgress = FALSE;
   /* Reset the tables that keep track of (destination address
      transaction id) pairs only duing powerup or external reset.
      When resetCause is software reset or cleared, we keep this
      table to ensure that we don't send a message to a destination
      with tid same as the one used last time for that destination.
      For power-up or external reset, we also need to ensure that
      using some other technique. We will delay transport or session
      layer sends by a small amount so that no messages are pending in
      target nodes. If we don't follow these guidelines, the target
      node may throw away messages sent after a reset as duplicates. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   if (nmp->resetCause == POWER_UP_RESET || nmp->resetCause == EXTERNAL_RESET)
   {
      gp->priTblSize       = 0;
      gp->nonpriTblSize    = 0;
   }
}

/*************************************************************
Function:  NewTrans
Returns:   SUCCESS if a transaction id can be assigned.
           FAILURE if it is not possible to assign an id.
Purpose:   To get a new transaction id.
Comments:  This function implements a new algorithm to assign the
           transaction id. It does not use the one in protocol specification.
Alg Idea:  For each of the following categories, we have an
           entry in the table.
           1. Subnet/Node
           2. group
           3. broadcast (domainwide or subnet)
           4. unique node id
           When a new id is requested, we increment the tid from
           the single space. We then search the table for this
           entry. If a matching entry is found, then we check
           if that tid was used last time for the same destination.
           If so, we bump it up by one. If not we use it. In
           either case, we record this tid in the table.
           If there was no such entry, we create a new one.
           If there is no space for the new entry, we release one
           that has remained more than MIN_TABLE_TIME seconds.
           If no such entry, then we fail to assign a tid.
*************************************************************/
Status NewTrans(Boolean   priorityIn, DestinationAddress addrIn,
                TransNum *transNumOut)
{
   uint16           i;
   TransCtrlRecord *transRecPtr;
   TransNum        *transNumPtr;
   TIDTableEntry   *tbl;
   uint16          *tblSize;
   Boolean          found;

   /* Point to the appropriate control record & table. */
   if (priorityIn)
   {
      transRecPtr = &gp->priTransCtrlRec;
      transNumPtr = &gp->priTransID;
      tbl         = gp->priTbl;
      tblSize     = &gp->priTblSize;
   }
   else
   {
      transRecPtr = &gp->nonpriTransCtrlRec;
      transNumPtr = &gp->nonpriTransID;
      tbl         = gp->nonpriTbl;
      tblSize     = &gp->nonpriTblSize;
   }

   /* Check if transaction already in progress. */
   if (transRecPtr->inProgress)
   {
      /* We can't allow a new transaction. Return failure. */
      return(FAILURE);
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* We can allow the transaction. Allocate a new TID. */
transRecPtr->transNum = *transNumPtr;

/* Make sure that this dest did not use this TID last time.
   If it did, increment the TID. */
/* Note: addrIn.addressMode can never be MULTICAST_ACK
         for transactions initiated by a node. */
found = FALSE;
for (i = 0; i < *tblSize; i++)
{
   /* Update timer for this i whether match or not. */
   UpdateMsTimer(&tbl[i].timer);

   /* If domainId does not match, skip entry. */
   if (addrIn.domainIndex != FLEX_DOMAIN &&
       (eep->domainTable[addrIn.domainIndex].len == 0xFF ||
        tbl[i].len != eep->domainTable[addrIn.domainIndex].len ||
        memcmp(eep->domainTable[addrIn.domainIndex].domainId,
               tbl[i].domainId,
               eep->domainTable[addrIn.domainIndex].len) != 0)
      )
   {
      continue; /* Not flex domain but domain mismatch. */
   }

   if (addrIn.domainIndex == FLEX_DOMAIN &&
       (tbl[i].len != addrIn.flexDomainLen ||
        memcmp(addrIn.flexDomainId,
               tbl[i].domainId,
               addrIn.flexDomainLen) != 0)
      )
   {
      continue; /* Flex domain but domain mismatch. */
   }

   switch(addrIn.addressMode)
   {
      case SUBNET_NODE:
         if (tbl[i].addressMode == SUBNET_NODE &&
             memcmp(&tbl[i].addr.subnetNode, &addrIn.addr.addr2a,
                    sizeof(SubnetAddress)) == 0)
         {
            found = TRUE;
         }
         break;
      case UNIQUE_NODE_ID:
         if (tbl[i].addressMode == UNIQUE_NODE_ID &&
             memcmp(tbl[i].addr.uniqueNodeId,
                    addrIn.addr.addr3.uniqueId,
                    UNIQUE_NODE_ID_LEN) == 0)
         {
            found = TRUE;
         }
         break;
      case MULTICAST:
         if (tbl[i].addressMode == MULTICAST &&
             tbl[i].addr.group  == addrIn.addr.addr1)
         {
            found = TRUE;
         }
         break;
      case BROADCAST:
         if (tbl[i].addressMode == BROADCAST &&
             tbl[i].addr.subnet == addrIn.addr.addr0)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            {
               found = TRUE;
            }
            break;
         default:
            ErrorMsg("NewTrans: Unexpected addressMode.\n");
            /* Should not come here. */
      }
      if (found)
      {
         break; /* Need to leave for loop with matched i value. */
      }
   }

   if (found)
   {
      /* Found a match. Check if last TID is same or not. */
      /* We can reuse this entry and reinitialize timer.  */
      if (tbl[i].tid == *transNumPtr)
      {
         /* Increment TID. */
         (*transNumPtr)++;
         if (*transNumPtr == 16)
         {
            *transNumPtr = 1; /* Wrap around. */
         }
         transRecPtr->transNum = *transNumPtr;
      }
      tbl[i].tid   = *transNumPtr;
      SetMsTimer(&tbl[i].timer,
                 (uint16)(MIN_TABLE_TIME * 1000));
      *transNumOut = *transNumPtr;
      transRecPtr->inProgress = TRUE;
      return(SUCCESS);
   }

   /* No match. Make a new entry. If no space. get a space. */
   /* All the timers must have been updated in the for loop above. */
   if (*tblSize == TID_TABLE_SIZE)
   {
      /* Table is full. See if any entry can be replaced. */
      found = FALSE;
      for (i = 0; i < *tblSize; i++)
      {
         if (tbl[i].timer.curTimerValue == 0) /* Expired */
         {
            found = TRUE;
            break;
         }
      }
      if (found)
      {
         /* Replace this entry with last entry of table. */
          (*tblSize)--;
          tbl[i] = tbl[*tblSize];

         /* Fall through to code below. */
      }
      else
      {
         /* Unable to find an entry. */
         return(FAILURE);
      }
   }
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    /* Now we have space for an entry. Add new entry. */
    /* Store the domain len and domain id in table */
    if (addrIn.domainIndex == FLEX_DOMAIN)
    {
       memcpy(tbl[*tblSize].domainId,
              addrIn.flexDomainId,
              addrIn.flexDomainLen);
       tbl[i].len = addrIn.flexDomainLen;
    }
    else
    {
       memcpy(tbl[*tblSize].domainId,
              eep->domainTable[addrIn.domainIndex].domainId,
              eep->domainTable[addrIn.domainIndex].len);
       tbl[i].len = eep->domainTable[addrIn.domainIndex].len;
    }

    tbl[*tblSize].addressMode = addrIn.addressMode;
    if (addrIn.addressMode == MULTICAST)
    {
       tbl[*tblSize].addr.group = addrIn.addr.addr1;
    }
    else if (addrIn.addressMode == SUBNET_NODE)
    {
       tbl[*tblSize].addr.subnetNode = addrIn.addr.addr2a;
    }
    else if (addrIn.addressMode == UNIQUE_NODE_ID)
    {
       memcpy(tbl[*tblSize].addr.uniqueNodeId,
              addrIn.addr.addr3.uniqueId,
              UNIQUE_NODE_ID_LEN);
    }
    else if (addrIn.addressMode == BROADCAST)
    {
       tbl[*tblSize].addr.subnet = addrIn.addr.addr0;
    }
    else
    {
       /* Should not come here as addressMode was checked before too. */
       ErrorMsg("NewTrans: Invalid addressMode at unexpected place\n");
    }
    SetMsTimer(&tbl[*tblSize].timer, (uint16)(MIN_TABLE_TIME * 1000));
    tbl[*tblSize].tid = *transNumPtr;
    *transNumOut      = *transNumPtr;
    (*tblSize)++;
    transRecPtr->inProgress = TRUE;
    return(SUCCESS);
}

/*************************************************************
Function:  TransDone
Returns:   None
Reference: Section 7 Protocol Spec.
Purpose:   To release the transaction record for future assignments.
Comments:  None
*************************************************************/
void TransDone(Boolean  priorityIn)
{
    TransCtrlRecord *transRecPtr;
    TransNum        *transNumPtr;

    /* Point to the appropriate control record & table. */
    if (priorityIn)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   {
      transRecPtr = &gp->priTransCtrlRec;
      transNumPtr = &gp->priTransID;
   }
   else
   {
      transRecPtr = &gp->nonpriTransCtrlRec;
      transNumPtr = &gp->nonpriTransID;
   }

   /* Mark transaction as available. */
   transRecPtr->inProgress = FALSE;

   /* Increment the corresponding transaction id. */
   (*transNumPtr)++;
   if (*transNumPtr == 16)
   {
      *transNumPtr = 1; /* Wrap Around to 1. */
   }
}

/*****************************************************************
Function:  ValidateTrans
Returns:   TRANS_CURRENT if the transNumIn matches transaction
           in progress.
           TRANS_NOT_CURRENT othewise.
Reference: Section 7, Protocol Specification.
Purpose:   To check if a given transNumIn is current or not.
Comments:  None
*****************************************************************/
TransStatus ValidateTrans(Boolean  priorityIn,
                          TransNum transNumIn)
{
   TransCtrlRecord *transRecPtr;

   /* Point to the appropriate control record & table. */
   if (priorityIn)
   {
      transRecPtr = &gp->priTransCtrlRec;
   }
   else
   {
      transRecPtr = &gp->nonpriTransCtrlRec;
   }

   if (transRecPtr->inProgress &&
       transRecPtr->transNum == transNumIn)
   {
      return(TRANS_CURRENT);
   }
   else
   {
      return(TRANS_NOT_CURRENT);
   }
}

/* *** END INFORMATIVE - Transaction ID Allcation *** */

/*-------------------------End of tcs.c-------------------------*/
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.8 Network Layer Algorithm

```
/*****************************************************************************
          File:        network.c

       Version:        1.7

     Reference:CNP, part 1, clause 8.

       Purpose:        To implement network layer functions.

          Note:        Reference implementation does not support
                       special nodes such as routers and bridges.
                       Extra code is needed to implement these.

         To Do:        None
*****************************************************************************/
/*---------------------------------------------------------------------------
Section: Includes
---------------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>

#include <cnp_1.h>
#include <node.h>
#include <queue.h>
#include <network.h>

/*---------------------------------------------------------------------------
Section: Constants
---------------------------------------------------------------------------*/
/* #define DEBUG */

/*---------------------------------------------------------------------------
Section: Type Definitions
---------------------------------------------------------------------------*/
/*****************************************************************************
   Byte data[1] is used so that variable data has address assigned by
   the compiler. Once we know the size of the record, we will use
   data[0], data[1], etc.

   data[0] is source subnet.
   data[1] is source node.
   Based on the addrFmt field and 1st bit of data[1], the rest of
   the data array is used appropriately.
*****************************************************************************/
#pragma maxalign(1)
typedef struct
{
   Bits       protocolVersion       :2;
   Bits       pduType               :2;
   Bits       addrFmt               :2;
   Bits       domainLength          :2;
   Byte       data[1];                   /* Variable part */
} NPDU;
#pragma maxalign()

/*---------------------------------------------------------------------------
Section: Globals
---------------------------------------------------------------------------*/
/* None */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/*-------------------------------------------------------------------------------
Section: Local Function Prototypes
-------------------------------------------------------------------------------*/
static Byte DecodeDomainLength(Byte lengthCode);
static Byte EncodeDomainLength(Byte length);

/*-------------------------------------------------------------------------------
Section: Function Definitions
-------------------------------------------------------------------------------*/
/*******************************************************************************
Function:  NWReset
Returns:   None
Reference: None
Purpose:   To initialize the queues used by the network layer.
Comments:  None.
*******************************************************************************/
void   NWReset(void)
{
   uint16 queueItemSize;

   /* Allocate and initialize the input queue. */
   gp->nwInBufSize   =
      DecodeBufferSize((uint8)eep->readOnlyData.nwInBufSize);
   gp->nwInQCnt      = DecodeBufferCnt((uint8)eep->readOnlyData.nwInBufCnt);
   queueItemSize     = gp->nwInBufSize + sizeof(NWReceiveParam);


   if (QueueInit(&gp->nwInQ, queueItemSize, gp->nwInQCnt) != SUCCESS)
   {
      ErrorMsg("NWReset: Unable to init the input queue.\n");
      gp->resetOk = FALSE;
      return;
   }

   /* Allocate and initialize the output queue. */
   gp->nwOutBufSize  =
      DecodeBufferSize((uint8)eep->readOnlyData.nwOutBufSize);
   gp->nwOutQCnt     = DecodeBufferCnt((uint8)eep->readOnlyData.nwOutBufCnt);
   queueItemSize     = gp->nwOutBufSize + sizeof(NWSendParam);

   if (gp->nwOutQCnt < 2)
   {
      ErrorMsg("NWReset: Network non-pri buffers count should be >= 2.\n");
      gp->resetOk = FALSE;
      return;
   }

   if (QueueInit(&gp->nwOutQ, queueItemSize, gp->nwOutQCnt)
       != SUCCESS)
   {
      ErrorMsg("NWReset: Unable to init the output queue.\n");
      gp->resetOk = FALSE;
      return;
   }

   /* Allocate and initialize the priority output queue. */
   gp->nwOutPriBufSize = gp->nwOutBufSize;
   gp->nwOutPriQCnt    =
      DecodeBufferCnt((uint8)eep->readOnlyData.nwOutBufPriCnt);
   queueItemSize       = gp->nwOutPriBufSize + sizeof(NWSendParam);

   if (gp->nwOutPriQCnt < 2)
   {
      ErrorMsg("NWReset: Network pri buffers count should be >= 2.\n");
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      gp->resetOk = FALSE;
      return;
   }

   if (QueueInit(&gp->nwOutPriQ, queueItemSize, gp->nwOutPriQCnt)
       != SUCCESS)
   {
      ErrorMsg("NWReset: Unable to init the priority output queue.\n");
      gp->resetOk = FALSE;
      return;
   }

   return;
}

/*****************************************************************************
Function:  NWSend
Returns:   None
Reference: None. No algorithms in protocol specification.
Purpose:   To send outgoing PDUS (APDU or SPDU or TPDU or AuthPDU)
           waiting on the queue (pri or nonpri) for network layer.
           Network layer forms the NPDU and the parameters for
           sending the NPDU and writes to the queue for the
           link/mac layer.
Comments:  Network buffer size is guaranteed to be at least
           20 bytes long as the encoding table's minimum value is 20.
           The NPDU's header's worst case size is 16. So, we are
           OK. No need to check for space when writing headers.
*****************************************************************************/
void   NWSend(void)
{
   NWSendParam  *nwSendParamPtr; /* Param in nwOutQ or nwPriOutQ.   */
   LKSendParam  *lkSendParamPtr; /* Param in lkOutQ or lkPriOutQ.   */
   APPReceiveParam *appReceiveParamPtr;
   NPDU         *npduPtr;        /* Pointer to NPDU being formed.   */
   Byte         *pduPtr;         /* Pointer to PDU etc being sent.  */
   Boolean       priority;       /* TRUE if processing pri msg.     */
   Byte          selField;       /* 0 or 1. Used to form NPDU.      */
   uint8         j;              /* For temporary use.              */
   uint16        npduSize;       /* Size of NPDU formed.            */
   uint8         numDomains;     /* Number of domains for this node.*/
   Boolean       flexDomain;     /* True if sending in flex domain. */
   uint8         domainLength;   /* Length of domain value sent.    */
   Byte          domainId[DOMAIN_ID_LEN]; /* Value of domain.       */

   /* Check if there is work to do and set pointers */
   if ( !QueueEmpty(&gp->nwOutPriQ) && !QueueFull(&gp->lkOutPriQ) )
   {
      /* Process priority message if there is one and it can be processed. */
      priority        = TRUE;
      nwSendParamPtr  = QueueHead(&gp->nwOutPriQ);
      lkSendParamPtr  = QueueTail(&gp->lkOutPriQ);
   }
   else if ( !QueueEmpty(&gp->nwOutQ) && !QueueFull(&gp->lkOutQ) )
   {
      /* Process non-priority message if there is one and can be processed */
      priority        = FALSE;
      nwSendParamPtr  = QueueHead(&gp->nwOutQ);
      lkSendParamPtr  = QueueTail(&gp->lkOutQ);
   }
   else
   {
      /* Either there is nothing to send or there is no space in link layer */
      return;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
}

/* For application layer messages, we need to give completion event
   using the tag given. This is for consistency with transport/session
   layers. Thus completiong events are streamlined in one place in
   application layer rather than lots of places. */
if (nwSendParamPtr->pduType == APDU_TYPE && QueueFull(&gp->appInQ))
{
   /* Can't deliver the indication. Wait until we can send indication */
   return;
}

/* Process the waiting PDU, form the NPDU and send it */

/* ptr to APDU or TPDU or SPDU or AuthPDU. */
pduPtr  = (Byte *)(nwSendParamPtr + 1);

/* ptr to NPDU constructed. */
npduPtr = (NPDU *)(lkSendParamPtr + 1);

/* Write the NPDU header. */
npduPtr->protocolVersion = PROTOCOL_VERSION; /* See cnp_1.h */
npduPtr->pduType         = nwSendParamPtr->pduType;
switch (nwSendParamPtr->destAddr.addressMode)
{
   case BROADCAST:
      npduPtr->addrFmt = 0;
        break;
   case MULTICAST:
      npduPtr->addrFmt = 1;
      break;
   case SUBNET_NODE:
   case MULTICAST_ACK:
      npduPtr->addrFmt = 2;
      break;
   case UNIQUE_NODE_ID:
      npduPtr->addrFmt = 3;
      break;
   default:
      ErrorMsg("NWSend: Unknown address mode.\n");
      /* Discard the packet as addrmode is wrong */
      nmp->errorLog = BAD_ADDRESS_TYPE;
      /* Send completion event if it was an APDU */
      if (nwSendParamPtr->pduType == APDU_TYPE)
      {
         appReceiveParamPtr = QueueTail(&gp->appInQ);
         appReceiveParamPtr->indication = COMPLETION;
         appReceiveParamPtr->success    = FALSE;
         appReceiveParamPtr->tag        = nwSendParamPtr->tag;
         EnQueue(&gp->appInQ);
      }
      if (priority)
      {
         DeQueue(&gp->nwOutPriQ);
      }
      else
      {
         DeQueue(&gp->nwOutQ);
      }
      return;
}

/* Write the domain length. */
/* First determine the number of domains for this node */
```


--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
if (eep->readOnlyData.twoDomains == 1)
{
   numDomains = 2;
}
else
{
   numDomains = 1;
}

/* if a node is in in unconfigured state and the message is not in flex
   domain, then we discard the message. We should not use the domain table
   in unconfigured state, irrespective of whether they are valid or not.
   However, we allow acks, response, challenge and reply. The field
   dropIfUnconfigured indicates whether this check is done or not. */
if (nwSendParamPtr->dropIfUnconfigured &&
    nwSendParamPtr->destAddr.domainIndex != FLEX_DOMAIN &&
    NodeUnConfigured())
{
   /* drop this packet. */
   /* Send completion event if it was an APDU */
   if (nwSendParamPtr->pduType == APDU_TYPE)
   {
      appReceiveParamPtr = QueueTail(&gp->appInQ);
      appReceiveParamPtr->indication = COMPLETION;
      appReceiveParamPtr->success    = FALSE;
      appReceiveParamPtr->tag        = nwSendParamPtr->tag;
      EnQueue(&gp->appInQ);
   }
   if (priority)
   {
      DeQueue(&gp->nwOutPriQ);
   }
   else
   {
      DeQueue(&gp->nwOutQ);
   }
   return;
}

/* If the domain used is not in use, it cannot send any packet */
if (nwSendParamPtr->destAddr.domainIndex < numDomains &&
    eep->domainTable[nwSendParamPtr->destAddr.domainIndex].len
       == 0xFF)
{
   if (!nwSendParamPtr->dropIfUnconfigured)
   {
      /* It is not ACK, RESP etc. Don't log domain error in this case.
         LNS might use join domain to leave a domain with ACKD. So, the
         ACK send by the transport layer will be in an invalid domain
         but should be ignored. */

      /* Discard the packet as the domain table entry is not in use */
      nmp->errorLog = INVALID_DOMAIN;
   }
   /* Send completion event if it was an APDU */
   if (nwSendParamPtr->pduType == APDU_TYPE)
   {
      appReceiveParamPtr = QueueTail(&gp->appInQ);
      appReceiveParamPtr->indication = COMPLETION;
      appReceiveParamPtr->success    = FALSE;
      appReceiveParamPtr->tag        = nwSendParamPtr->tag;
      EnQueue(&gp->appInQ);
   }
   if (priority)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      {
         DeQueue(&gp->nwOutPriQ);
      }
      else
      {
         DeQueue(&gp->nwOutQ);
      }
      return;
   }

   /* Use destAddr to determine the domain and write it.
      compute and store domainLength and domainId for later use. */
   if (nwSendParamPtr->destAddr.domainIndex < numDomains)
   {
      /* One of this node's domains. */
      domainLength =
         eep->domainTable[nwSendParamPtr->destAddr.domainIndex].len;
      npduPtr->domainLength =
         EncodeDomainLength(
           eep->domainTable[nwSendParamPtr->destAddr.domainIndex].len
           );
      if (domainLength <= DOMAIN_ID_LEN)
      {
         memcpy(domainId,
                eep->domainTable[
                   nwSendParamPtr->destAddr.domainIndex].domainId,
                domainLength); /* Save id for now */
      }
      flexDomain = FALSE;
   }
   else if (nwSendParamPtr->destAddr.domainIndex == FLEX_DOMAIN)
   {
      /* Flex domain message. */
      domainLength = nwSendParamPtr->destAddr.flexDomainLen;
      npduPtr->domainLength =
         EncodeDomainLength(
               nwSendParamPtr->destAddr.flexDomainLen
               );
      if (domainLength <= DOMAIN_ID_LEN)
      {
         memcpy(domainId,
                nwSendParamPtr->destAddr.flexDomainId,
                domainLength); /* Save the id for now */
      }
      flexDomain = TRUE;
   }
   else
   {
      ErrorMsg("NWSend: Domain index is not valid.\n");
      nmp->errorLog = INVALID_DOMAIN;
      /* Send completion event if it was an APDU */
      if (nwSendParamPtr->pduType == APDU_TYPE)
      {
         appReceiveParamPtr = QueueTail(&gp->appInQ);
         appReceiveParamPtr->indication = COMPLETION;
         appReceiveParamPtr->success    = FALSE;
         appReceiveParamPtr->tag        = nwSendParamPtr->tag;
         EnQueue(&gp->appInQ);
      }
      /* Discard the packet as the domain index is invalid. */
      if (priority)
      {
         DeQueue(&gp->nwOutPriQ);
      }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   else
   {
      DeQueue(&gp->nwOutQ);
   }
   return;
}

if (domainLength != 0 && domainLength != 1 &&
    domainLength != 3 && domainLength != 6)
{
   /* Protocol specification indicates that domainLength has to be
      one of the above values. If not, it is a bad value. */
   ErrorMsg("NWSend: Domain length is not valid.\n");
   nmp->errorLog = INVALID_DOMAIN;
   /* Send completion event if it was an APDU */
   if (nwSendParamPtr->pduType == APDU_TYPE)
   {
      appReceiveParamPtr = QueueTail(&gp->appInQ);
      appReceiveParamPtr->indication = COMPLETION;
      appReceiveParamPtr->success    = FALSE;
      appReceiveParamPtr->tag        = nwSendParamPtr->tag;
      EnQueue(&gp->appInQ);
   }
   /* Discard the packet as the domain length is invalid. */
   if (priority)
   {
      DeQueue(&gp->nwOutPriQ);
   }
   else
   {
      DeQueue(&gp->nwOutQ);
   }
   return;
}

/* Write the address. It goes into data[0] onwards. */

/* First, write source subnet. */
if (flexDomain)
{
   npduPtr->data[0] = 0; /* It is 0 for flex domain response. */
}
else
{
   npduPtr->data[0] =
      eep->domainTable[nwSendParamPtr->destAddr.domainIndex].subnet;
}

/* Determine the selField value. */
/* Only MULTICAST_ACK has selector field as 0. For all others, it is 1. */
if (nwSendParamPtr->destAddr.addressMode == MULTICAST_ACK)
{
   selField = 0;
}
else
{
   selField = 1;
}

/* Write the source node. */
/* Node value is only 7 bits. The high order bit is selField. */
if (flexDomain)
{
   npduPtr->data[1] = selField << 7; /* SrcNode is 0. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
}
else
{
   npduPtr->data[1] = selField << 7 |
      eep->domainTable[nwSendParamPtr->destAddr.domainIndex].node;
}

/* Write the destination address. */
/* Set j to the index for writing the domain field. */
switch (nwSendParamPtr->destAddr.addressMode)
{
   case BROADCAST:
      npduPtr->data[2] = nwSendParamPtr->destAddr.addr.addr0;
      j = 3;
      break;
   case MULTICAST:
      npduPtr->data[2] = nwSendParamPtr->destAddr.addr.addr1;
      j = 3;
      break;
   case SUBNET_NODE:
      nwSendParamPtr->destAddr.addr.addr2a.selField = 1;
      memcpy(&npduPtr->data[2],
                     &nwSendParamPtr->destAddr.addr.addr2a,
                     2);
      j = 4;
      break;
   case MULTICAST_ACK:
      nwSendParamPtr->destAddr.addr.addr2b.subnetAddr.selField = 1;
      memcpy(&npduPtr->data[2],
             &nwSendParamPtr->destAddr.addr.addr2b,
             4);
      j = 6;
      break;
   case UNIQUE_NODE_ID:
      memcpy(&npduPtr->data[2],
              &nwSendParamPtr->destAddr.addr.addr3,
             7);
      j = 9;
      break;
   default:
      ErrorMsg("NWSend: Unknown address format.\n");
      /* Discard the packet as the address Mode is wrong. */
      nmp->errorLog = BAD_ADDRESS_TYPE;
      /* Send completion event if it was an APDU */
      if (nwSendParamPtr->pduType == APDU_TYPE)
      {
         appReceiveParamPtr = QueueTail(&gp->appInQ);
         appReceiveParamPtr->indication = COMPLETION;
         appReceiveParamPtr->success    = FALSE;
         appReceiveParamPtr->tag        = nwSendParamPtr->tag;
         EnQueue(&gp->appInQ);
      }
      if (priority)
      {
         DeQueue(&gp->nwOutPriQ);
      }
      else
      {
         DeQueue(&gp->nwOutQ);
      }
      return;
}

/* Now, j has the index of data field in which domain goes. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* Write the domain. We saved this information earlier. */
   memcpy(&npduPtr->data[j], domainId, domainLength);
   j += domainLength;

   /* Write the enclosed PDU. */
   if (1 + j + nwSendParamPtr->pduSize > gp->nwOutBufSize)
   {
      /* Discard the packet as it is too long. */
      nmp->errorLog = WRITE_PAST_END_OF_NET_BUFFER;
      /* Send completion event if it was an APDU */
      if (nwSendParamPtr->pduType == APDU_TYPE)
      {
         appReceiveParamPtr = QueueTail(&gp->appInQ);
         appReceiveParamPtr->indication = COMPLETION;
         appReceiveParamPtr->success    = FALSE;
         appReceiveParamPtr->tag        = nwSendParamPtr->tag;
         EnQueue(&gp->appInQ);
      }
      if (priority)
      {
         DeQueue(&gp->nwOutPriQ);
      }
      else
      {
         DeQueue(&gp->nwOutQ);
      }
      return;
   }

   memcpy(&npduPtr->data[j], pduPtr, nwSendParamPtr->pduSize);
   /* NPDU size is header_size + enclosed PDU size. */
   npduSize = 1 + j + nwSendParamPtr->pduSize;

   /* Write the parameters for the link layer. */
   lkSendParamPtr->deltaBL = nwSendParamPtr->deltaBL;
   lkSendParamPtr->altPath = nwSendParamPtr->altPath;
   lkSendParamPtr->pduSize = npduSize;

   /* Update both queues. */
   if (priority)
   {
      DeQueue(&gp->nwOutPriQ);
      EnQueue(&gp->lkOutPriQ);
#ifdef DEBUG
      DebugMsg("NWSend: Sending a priority packet.");
#endif
   }
   else
   {
      DeQueue(&gp->nwOutQ);
      EnQueue(&gp->lkOutQ);
#ifdef DEBUG
      DebugMsg("NWSend: Sending a non-priority packet.");
#endif
   }

   INCR_STATS(nmp->stats.layer3Transmitted);

   /* Send completion event if it was an APDU */
   if (nwSendParamPtr->pduType == APDU_TYPE)
   {
      appReceiveParamPtr = QueueTail(&gp->appInQ);
      appReceiveParamPtr->indication = COMPLETION;
      appReceiveParamPtr->success    = TRUE;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      appReceiveParamPtr->tag        = nwSendParamPtr->tag;
      EnQueue(&gp->appInQ);
   }

   return;
}

/************************************************************************
Function:  NWReceive
Returns:   None
Reference: None. No receive algorithms in protocol specification.
Purpose:   To receive packets waiting for the network layer
           from the link layer. The NPDU is retrieved, processed
           and the enclosed PDU is sent to the proper destination queue.
           If the NPDU is not for this node, it is discarded.
Comments:  Discard packets originated from this node itself.
           It might receive such packets in the presence of repeaters.

           Discard packets in which version field is not correct.

           Discard packets that match a domain but not
           subnet or group.

           When a node is not in configured state and not in
           hard-offline state, it can only receive broadcast
           or matching Unique Node ID messages. In such a case,
           the domain on which it is received need not match its
           own domain. If it does not match, then the message is
           said to be received in a flex domain.
************************************************************************/
void   NWReceive(void)
{
   NWReceiveParam      *nwReceiveParamPtr; /* Param in gp->nwInQ.   */
   APPReceiveParam     *appReceiveParamPtr;
   TSAReceiveParam     *tsaReceiveParamPtr;
   SourceAddress        srcAddr;    /* Address of source node.      */
   NPDU                *npduPtr;    /* ptr to NPDU being received.  */
   Byte                *pduPtr;     /* ptr to item in target queue. */
   uint16               pduSize;    /* Size of enclosed PDU.        */
   Boolean              flexDomain; /* TRUE => NPDU in flexdomain.  */
   uint8                numDomains; /* # of domains of this node.   */
   uint8                domainLength;  /* Domain length             */
   Byte                 domainId[DOMAIN_ID_LEN]; /* Temp.           */
   Byte                 uniqueNodeId[UNIQUE_NODE_ID_LEN]; /* Temp.  */
   SubnetAddress        destAddr;   /* Temp.                        */
   uint8                j;          /* Temp.                        */

   /* First, check if we have any packets to process. */
   if (QueueEmpty(&gp->nwInQ))
   {
      return; /* Nothing to process. */
   }

   /* Until we determine what type of PDU we have, we cannot
      check for space availability in destination queue.
      Also, it is possible that the NPDU may very well be
      discarded. */

   /* Set the pointer to NPDU in nwInQ. */
   nwReceiveParamPtr = QueueHead(&gp->nwInQ);
   npduPtr           = (NPDU *)(nwReceiveParamPtr + 1);

   /* Discard NPDU if version is not PROTOCOL_VERSION */
   if (npduPtr->protocolVersion != PROTOCOL_VERSION)
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   {
      DeQueue(&gp->nwInQ);
#ifdef DEBUG
      DebugMsg(" NWReceive: Discard packet. Wrong version.\n");
#endif
      return;
   }

   /* Determine the source address. */
   memcpy(&srcAddr.subnetAddr, npduPtr->data, 2);

   /* Determine the destination address used and set srcAddr properly. */
   /* For MULTICAST and MULTICAST_ACK address modes, the
      group and/or member values are copied into srcAddr.group
      or srcAddr.ackNode. For BROADCAST and SUBNET_NODE address
      modes, destAddr is used to store subnet and/or node
      values. For UNIQUE_NODE_ID, uniqueNodeId is used & subnet is ignored */
   /* Also, set j to domain field's index. */
   switch (npduPtr->addrFmt)
   {
      case 0:
         srcAddr.addressMode = BROADCAST;
         destAddr.subnet   = npduPtr->data[2];
         j = 3;
         break;
      case 1:
         srcAddr.addressMode = MULTICAST;
         srcAddr.group       = npduPtr->data[2];
         j = 3;
         break;
      case 2:
         if (srcAddr.subnetAddr.selField == 1)
         {
            srcAddr.addressMode = SUBNET_NODE;
            memcpy(&destAddr, &npduPtr->data[2], 2);
            j = 4;
         }
         else
         {
            srcAddr.addressMode = MULTICAST_ACK;
            memcpy(&destAddr, &npduPtr->data[2], 2);
            memcpy(&srcAddr.ackNode.subnetAddr, &destAddr, 2);
            memcpy(&srcAddr.ackNode.groupAddr, &npduPtr->data[4], 2);
            j = 6;
         }
         break;
      case 3:
         srcAddr.addressMode = UNIQUE_NODE_ID;
         destAddr.subnet = npduPtr->data[2]; /* Routing Purpose */
         memcpy(uniqueNodeId, &npduPtr->data[3], UNIQUE_NODE_ID_LEN);
         j = 3 + UNIQUE_NODE_ID_LEN;
         break;
      default:
         /* Discard it as the address format is wrong. */
         ErrorMsg("NWReceive: Unknown addFmt.\n");
         nmp->errorLog = BAD_ADDRESS_TYPE;
         DeQueue(&gp->nwInQ);
         return;
   }

   /* Determine the domain. */
   domainLength = DecodeDomainLength(npduPtr->domainLength);
   if (domainLength != 0 && domainLength != 1 &&
       domainLength != 3 && domainLength != 6)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
{
   ErrorMsg("NWReceive: Domain length is not valid.\n");
   nmp->errorLog = INVALID_DOMAIN;
   /* Discard the packet as the domain length is invalid. */
   DeQueue(&gp->nwInQ);
   return;
}

/* domainLength is good. Safe to use memcpy now. */
memcpy(domainId, &npduPtr->data[j], domainLength);

j += domainLength; /* Now j points to enclosed PDU. */

/* Determine the number of domains for this node. */
if (eep->readOnlyData.twoDomains)
{
   numDomains = 2;
}
else
{
   numDomains = 1;
}

/* Check if the NPDU is received in flexDomain.
   If domainId does not match any of this node's domains,
   then the msg is said to have been received in flex domain. */

flexDomain = FALSE; /* Assume it is not flex domain. */
if (NodeConfigured() && eep->domainTable[0].len != 0xFF &&
    domainLength == eep->domainTable[0].len &&
    memcmp(domainId, eep->domainTable[0].domainId,
           domainLength) == 0)
{
   /* Matches domainId in index 0 */
   srcAddr.domainIndex = 0;
}
else if (NodeConfigured() && numDomains == 2 &&
         eep->domainTable[1].len != 0xFF &&
         domainLength == eep->domainTable[1].len &&
         memcmp(domainId, eep->domainTable[1].domainId,
                domainLength) == 0)
{
   /* Matches domainId in index 1 */
   srcAddr.domainIndex = 1;
}
else
{
   /* Must be a flex domain. */
   srcAddr.domainIndex = FLEX_DOMAIN;
   srcAddr.flexDomainLen = domainLength;
   memcpy(srcAddr.flexDomainId, domainId, domainLength);
   flexDomain = TRUE;
}

/* Determine if the packet was sent by myself. If so, drop. */
/* We can do this check only in non-flexdomain as
   src subnet and node are 0 in flex domain. */
if (! flexDomain &&
    memcmp(&srcAddr.subnetAddr,
           &eep->domainTable[srcAddr.domainIndex].subnet, 2)
     == 0)
{
   /* Not flex domain and source addr matches. */
   DeQueue(&gp->nwInQ); /* Discard packet. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
#ifdef DEBUG
      DebugMsg("NWReceive. Discarding Self Pck\n");
#endif
      return;
   }

   /* Drop packet in various address modes if not for us. */
   switch(srcAddr.addressMode)
   {
      case BROADCAST:
         if (!flexDomain &&
             destAddr.subnet != 0 && /* subnet broadcast. */
             memcmp(&destAddr.subnet,
                    &eep->domainTable[srcAddr.domainIndex].subnet,
                    1) != 0)
         {
            /* Domain matches but destAddr does not. Not for us. */
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard BC pck. Not my subnet.\n");
#endif
            return;
         }
         srcAddr.broadcastSubnet = destAddr.subnet;
         break;
      case MULTICAST:
         if (!flexDomain &&
             !IsGroupMember(srcAddr.domainIndex,
                            srcAddr.group, NULL) )
         {
            /* Domain matches but group does not. Not for us. */
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard MC pck. Not my group.\n");
#endif
            return;
         }
         break;
      case SUBNET_NODE:
         if (!flexDomain &&
             memcmp(&destAddr,
                    &eep->domainTable[srcAddr.domainIndex].subnet,
                    2) != 0)
         {
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard unicast logical packet. "
                     "Not my subnet (or subnode).\n");
#endif
            return;
         }
         break;
      case MULTICAST_ACK:
         /* Make sure the destination subnet/node matches. */
         if (!flexDomain &&
             memcmp(&destAddr,
                    &eep->domainTable[srcAddr.domainIndex].subnet,
                    2) != 0)
         {
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard multicast ack packet. "
                     "Not my subnet (or subnode).\n");
#endif
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            return;
         }
         /* Also make sure that group matches. */
         if (!flexDomain &&
             !IsGroupMember(srcAddr.domainIndex,
                            srcAddr.ackNode.groupAddr.group,
                            NULL) )
         {
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard multicast ack packet. "
                     "Not my group.\n");
#endif
            return;
         }
         break;
      case UNIQUE_NODE_ID:
         if (memcmp(uniqueNodeId,
                  eep->readOnlyData.uniqueNodeId,
                  UNIQUE_NODE_ID_LEN) != 0)
         {
            /* Unique Node Id message but not for our id. */
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard Unique Node ID packet. Not my Id.\n");
#endif
            return;
         }
         break;
      default:
         ; /* Null statement. */
         /* Error message has been already printed in the previous switch. */
         /* Control should not come here. But, let us play safe. */
         DeQueue(&gp->nwInQ);
         return;
   }


   /* If a node is in unconfigured state,
      only broadcast and Unique Node ID messages can be received. */
   if (NodeUnConfigured() &&
       srcAddr.addressMode != BROADCAST &&
       srcAddr.addressMode != UNIQUE_NODE_ID)
   {
      /* Drop the packet. */
      DeQueue(&gp->nwInQ);
#ifdef DEBUG
      DebugMsg("NWReceive: Discard packet. We are not online.\n");
#endif
      return;
   }

   /* Drop packets received on flexDomain if the state
      is not unconfigured and it is not Unique Node ID addressed. */
   /* Unique Node ID addressed packets are always received. */
   /* i.e if node is configured, flexdomain is not possible
      unless it is Unique Node ID addressed. */
   if (flexDomain &&
       NodeConfigured() &&
       srcAddr.addressMode != UNIQUE_NODE_ID)
   {
      /* Drop the packet. */
      DeQueue(&gp->nwInQ);
#ifdef DEBUG
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      DebugMsg("NWReceive: Discard packet. Flex domain & not Neu. Id.\n");
#endif
      return;
   }

   /* We now got a packet that must be received. */
   INCR_STATS(nmp->stats.layer3Received);

   /* pduSize = npduSize - npduHeaderSize. */
   /* j is length of the variable part header of NPDU. */
   /* The fixed portion of NPDU header is always 1 byte. */
   pduSize = nwReceiveParamPtr->pduSize - j - 1;

   /* Set the pdu pointer properly. */
   switch (npduPtr->pduType)
   {
      case APDU_TYPE:
         if (QueueFull(&gp->appInQ) ||
             pduSize > gp->appInBufSize)
         {
            /* No space or insufficient space. Discard packet. */
            if (pduSize > gp->appInBufSize)
            {
               nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
            }
            INCR_STATS(nmp->stats.lostMessages);
            DeQueue(&gp->nwInQ);
#ifdef DEBUG
            DebugMsg("NWReceive: Discard packet. Insufficient space.\n");
#endif
            return;
         }
         /* Queue is not full and buffer has sufficient space. */
         appReceiveParamPtr = QueueTail(&gp->appInQ);
         pduPtr = (Byte *) appReceiveParamPtr +
                               sizeof(APPReceiveParam);

         appReceiveParamPtr->indication = MESSAGE;
         appReceiveParamPtr->srcAddr    = srcAddr;
         appReceiveParamPtr->priority   = nwReceiveParamPtr->priority;
         appReceiveParamPtr->altPath    = nwReceiveParamPtr->altPath;
         appReceiveParamPtr->pduSize    = pduSize;
         appReceiveParamPtr->auth       = FALSE;
         appReceiveParamPtr->service    = UNACKD;
         memcpy(pduPtr, &npduPtr->data[j], pduSize);
         EnQueue(&gp->appInQ);
         INCR_STATS(nmp->stats.layer6_7MsgsRcvd);
         DeQueue(&gp->nwInQ);
         return;
      case TPDU_TYPE: /* Fall through. */
      case SPDU_TYPE: /* Fall through. */
      case AUTHPDU_TYPE:
         if (QueueFull(&gp->tsaInQ) ||
             pduSize > gp->tsaInBufSize)
         {
            /* No space or insufficient space. Discard packet. */
            if (pduSize > gp->tsaInBufSize)
            {
               /* Buffer sizes are based on app buf sizes. See
                  TSAReset function. */
               nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
            }
            INCR_STATS(nmp->stats.lostMessages);
            DeQueue(&gp->nwInQ);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
#ifdef DEBUG
            DebugMsg("NWReceive: Discard packet. Insufficient space.\n");
#endif
            return;
         }
         /* Queue is not full and buffer has sufficient space. */
         tsaReceiveParamPtr = QueueTail(&gp->tsaInQ);
         pduPtr = (Byte *) tsaReceiveParamPtr +
                              sizeof(TSAReceiveParam);
         tsaReceiveParamPtr->pduType   = npduPtr->pduType;
         tsaReceiveParamPtr->srcAddr   = srcAddr;
         tsaReceiveParamPtr->priority  = nwReceiveParamPtr->priority;
         tsaReceiveParamPtr->altPath   = nwReceiveParamPtr->altPath;
         tsaReceiveParamPtr->pduSize   = pduSize;
         memcpy(pduPtr, &npduPtr->data[j], pduSize);
         EnQueue(&gp->tsaInQ);
         DeQueue(&gp->nwInQ);
         return;
      default:
         ErrorMsg("NWReceive: Unknown PDU was received.\n");
         nmp->errorLog = UNKNOWN_PDU;
         DeQueue(&gp->nwInQ);
         return;
   }

   /* Should not come here. */
}

/************************************************************************
Function:  DecodeDomainLength
Returns:   Decoded value of domain length code.
Reference: None.
Purpose:   To compute the actual domain length from code.
Comments:  None.
************************************************************************/
static Byte DecodeDomainLength(Byte lengthCodeIn)
{
   switch(lengthCodeIn)
   {
      case 0:
               return(0);
      case 1:
               return(1);
      case 2:
               return(3);
      case 3:
               return(6);
      default:
         /* Impossible to come here as lengthCode is 2 bits. */
         ;
   }
   return(0); /* To silence the compiler from complaining. */
}

/************************************************************************
Function:  EncodeDomainLength
Returns:   Encode value of domain length.
Reference: None
Purpose:   To compute the encoded value of domain length given.
Comments:
************************************************************************/
static Byte EncodeDomainLength(Byte lengthIn)
{
   switch(lengthIn)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   {
      case 0:
              return(0);
      case 1:
              return(1);
      case 3:
              return(2);
      case 6:
              return(3);
      default:
              return(0); /* should not come here. But has to return
                            something. Chose 0 arbitrarily */
   }
}

/*------------------------------End of network.c-----------------------------*/
_
```

A.9 TPDU and SPDU Send Algorithm with Authentication

```
/*********************************************************************
          File:        tsa.c (Transport Session Authentication)

       Version:        1.7

     Reference:        Sections 9, 10, Protocol Specification.

       Purpose:        Transport, Session and Authentication Layers.

          Note:        None.

         To Do:        None.
*********************************************************************/
/*-------------------------------------------------------------------------
Section: Includes
-------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <cnp_1.h>
#include <node.h>
#include <queue.h>
#include <tcs.h>
#include <tsa.h>
#include <app.h> /* For TAG related macros and constants */

/*-------------------------------------------------------------------
Section: Constant Definitions.
-------------------------------------------------------------------*/
/* #define DEBUG */
/* The last few tries for a message are sent using alternate path.
   The following constant determines how many are sent like this.
   A message is sent on alternate path if retries_left <= ALT_PATH_COUNT.
   Thus actual # of messages sent on alternate path is ALT_PATH_COUNT + 1 */
#define ALT_PATH_COUNT 1

/*-------------------------------------------------------------------
Section: Type Definitions.
-------------------------------------------------------------------*/
typedef enum
{
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   TRANSPORT,
   SESSION
} Layer;

typedef enum
{
   ACKD_MSG        = 0,  /* for Transport */
   REQUEST_MSG     = 0,  /* for Session   */
   CHALLENGE_MSG   = 0,  /* for Authentication */
   UNACK_RPT_MSG   = 1,  /* for Transport */
   ACK_MSG         = 2,  /* for Transport */
   RESPONSE_MSG    = 2,  /* for Session */
   REPLY_MSG       = 2,  /* for Authentication */
   REMINDER_MSG    = 4,  /* for Transport and Session */
   REM_MSG_MSG     = 5   /* for Transport and Session */
} PDUMsgType;  /* Type of msg sent in the PDU
                  (TPDU or SPDU or AuthPDU) */

#pragma maxalign(1)
typedef struct
{
   Bits          auth       :1;   /* Needs authentication? */
   Bits          pduMsgType :3;   /* See PDUMsgType above */
   Bits          transNum   :4;
   Byte          data[1];        /* Variable length field */
} TSPDU; /* Transport or Session PDU */

typedef TSPDU *TSPDUPtr;

typedef struct
{
   Bits  fmt            :2;   /* Same as addrfmt.                */
   Bits  pduMsgType     :2;   /* Type of AuthPDU. See PDUMsgType.*/
   Bits  transNum       :4;   /* Transaction number.             */
   union {
      Byte randomBytes[8];    /* Random number for challenge.    */
      Byte cryptoBytes[8];    /* Encrypted value in response.    */
   } value;
   Byte group;                /* Present only if fmt = 1.        */
} AuthPDU;
#pragma maxalign()

typedef AuthPDU *AuthPDUPtr;

/*-----------------------------------------------------------------
Section: Globals.
-----------------------------------------------------------------*/
/* None */


/* Array to convert address format to address mode. If format is 2,
   we convert to SUBNET_NODE instead of MULTICAST_ACK */
static Byte addrFmtToMode[4] =
{
   BROADCAST,     /* 0 */
   MULTICAST,     /* 1 */
   SUBNET_NODE,   /* 2 */
   UNIQUE_NODE_ID /* 3 */
};

/* Array to convert address mode to format. For example,
   BROADCAST to 0 MULTICAST to 1 etc. */
static Byte addrModeToFmt[6] =
{
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   0,
   2, /* SUBNET_NODE   */
   3, /* UNIQUE_NODE_ID*/
   0, /* BROADCAST     */
   1, /* MULTICAST     */
   2  /* MULTICAST_ACK */
};

/*------------------------------------------------------------------
Section: Local Function Prototypes.
------------------------------------------------------------------*/
/* Authentication related functions. */
static void InitiateChallenge(uint16 rrIndexIn);
static void SendReplyToChallenge(void);
static void ProcessReply(void);

/* Transport layer related functions. */
static void TPSendAck(uint16 rrIndexIn);
static void TPReceiveAck(void);

/* Session layer related functions. */
static void SNSendResponse(uint16 rrIndexIn, Boolean nullResponse);
static void SNReceiveResponse(void);

/* Functions that are common to both transport and session layers. */
static void XmitTimerExpiration(Layer layerIn, Boolean priorityIn);
static void TerminateTrans(Boolean priorityIn);
static void SendNewMsg(Layer layerIn, Boolean priorityIn);
static void ReceiveNewMsg(Layer layerIn);
static void ReceiveRem(Layer layerIn);
static void Deliver(uint16 rrIndexIn);

static int16 AllocateRR(void);
static int16 RetrieveRR(SourceAddress srcAddrIn, Boolean priorityIn);

static uint16 ComputeRecvTimerValue(AddrMode      addrModeIn,
                                    MulticastAddress group);
static void Encrypt(Byte rand[], APDU *apdu, uint16 apduSize,
            Byte domainIndex, Byte encryptValue[]);

/*------------------------------------------------------------------
Section: Function Definitions.
------------------------------------------------------------------*/
/*****************************************************************
Function:  TSAReset
Returns:   None
Reference: None
Purpose:   To initialize the queues used by the transport and session
           layers and to initialize transmit and receive records.
Comments:  Sets gp->resetOk to FALSE if unable to reset properly.
*****************************************************************/
void TSAReset(void)
{
   uint16  queueItemSize;
   uint16  i;

   /* Allocate and initialize the input queue. */

   /* Some TSPDUs have APDU attached and others do not.
      The max # bytes for TSPDU with no APDU is 10 (REMINDER).
      The max header size for those with APDU is 4 (REM/MSG). */
   gp->tsaInBufSize =
      DecodeBufferSize((uint8)eep->readOnlyData.appInBufSize) + 4;
   gp->tsaInBufSize = MAX(gp->tsaInBufSize, 10);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
gp->tsaInQCnt    = DecodeBufferCnt((uint8)eep->readOnlyData.appInBufCnt);
queueItemSize    = gp->tsaInBufSize + sizeof(TSAReceiveParam);

if (QueueInit(&gp->tsaInQ, queueItemSize, gp->tsaInQCnt)
    != SUCCESS)
{
   ErrorMsg("TSAReset: Unable to initialize the input queue.");
   gp->resetOk = FALSE;
   return;
}

/* Allocate and initialize the output queue. */
gp->tsaOutBufSize =
   DecodeBufferSize((uint8)eep->readOnlyData.appOutBufSize) + 4;
gp->tsaOutBufSize = MAX(gp->tsaOutBufSize, 10);
gp->tsaOutQCnt    =
   DecodeBufferCnt((uint8)eep->readOnlyData.appOutBufCnt);
queueItemSize     = gp->tsaOutBufSize + sizeof(TSASendParam);

if (QueueInit(&gp->tsaOutQ, queueItemSize, gp->tsaOutQCnt)
    != SUCCESS)
{
   ErrorMsg("TSAReset: Unable to initialize the output queue.");
   gp->resetOk = FALSE;
   return;
}

/* Allocate and initialize the priority output queue. */
gp->tsaOutPriBufSize = gp->tsaOutBufSize;
gp->tsaOutPriQCnt =
   DecodeBufferCnt((uint8)eep->readOnlyData.appOutBufPriCnt);
queueItemSize    = gp->tsaOutPriBufSize + sizeof(TSASendParam);

if (QueueInit(&gp->tsaOutPriQ, queueItemSize, gp->tsaOutPriQCnt)
    != SUCCESS)
{
   ErrorMsg("TSAReset: Unable to initialize the priority output queue.");
   gp->resetOk = FALSE;
   return;
}

/* Allocate and initialize the responses queue. */
gp->tsaRespBufSize = gp->tsaOutBufSize;
gp->tsaRespQCnt    = gp->tsaOutQCnt;
queueItemSize      = gp->tsaRespBufSize + sizeof(TSASendParam);

if (QueueInit(&gp->tsaRespQ, queueItemSize, gp->tsaRespQCnt)
    != SUCCESS)
{
   ErrorMsg("TSAReset: Unable to initialize the responses queue.");
   gp->resetOk = FALSE;
   return;
}

/* Initialize the transmit records. */
gp->xmitRec.status    = UNUSED_TX;
gp->priXmitRec.status = UNUSED_TX;

/* Initialize the receive records. */
gp->recvRecCnt = RECEIVE_TRANS_COUNT;
gp->recvRec    = AllocateStorage((uint16)(gp->recvRecCnt *
                                 sizeof(ReceiveRecord)));
if (gp->recvRec == NULL)
{
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      ErrorMsg("TSAReset: Insufficient space for allocating receive records.");
      gp->resetOk = FALSE;
      return;
   }

   for (i = 0; i < gp->recvRecCnt; i++)
   {
      gp->recvRec[i].response =
         AllocateStorage(DecodeBufferSize((uint8) eep-
>readOnlyData.appOutBufSize));
      gp->recvRec[i].apdu =
         AllocateStorage(DecodeBufferSize((uint8) eep-
>readOnlyData.appInBufSize));
      if (gp->recvRec[i].response == NULL ||
          gp->recvRec[i].apdu     == NULL)
      {
         ErrorMsg("TSAReset: Insufficient space for response or apdu.");
         gp->resetOk = FALSE;
         return;
      }
      gp->recvRec[i].status = UNUSED_RR;
   }

   /* Initialize the running count for request id assignment. */
   gp->reqId = 0;

   return;
}

/*****************************************************************
Function:  TPSend
Returns:   None
Reference: Section 8, Protocol Specification.
Purpose:   To implement send algorithm for transport layer.
           If there is anything to be sent by transport layer,
           it processes that message and calls the right function
           that sends it.
Comments:  Update the priority transaction timer, if it exists.
           Update the non-priority transaction timer, if it exists.
           If the priority transaction timer expired then
              process this event.
           else if there is a priority message to be sent and there is space
                in priority queue of the network layer then
              process the priority message.
           else if non-priority transaction timer expired then
              process that event.
           else if there is a non-priority message to be sent and there
                is space in the non-priority queue of the network layer then
              process the non-priority message.
           else
                there is nothing to do. return.
Note:
*****************************************************************/
void TPSend(void)
{

   /* Delay TPSend after power-up or external reset. */
   if ( gp->tsDelayTimer.curTimerValue > 0 &&
        (nmp->resetCause == POWER_UP_RESET ||
         nmp->resetCause == EXTERNAL_RESET)
      )
   {
      UpdateMsTimer(&gp->tsDelayTimer);
      return; /* Do nothing */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }

   /* Update transmit timers, if they do exist. */
   if (gp->priXmitRec.status == TRANSPORT_TX &&
       gp->priXmitRec.xmitTimer.curTimerValue > 0)
   {
      UpdateMsTimer(&gp->priXmitRec.xmitTimer);
   }
   if (gp->xmitRec.status == TRANSPORT_TX &&
       gp->xmitRec.xmitTimer.curTimerValue > 0)
   {
      UpdateMsTimer(&gp->xmitRec.xmitTimer);
   }

   /************************************************
     Priority transaction timer expired event.
    ************************************************/
   if (gp->priXmitRec.status == TRANSPORT_TX &&
       gp->priXmitRec.xmitTimer.curTimerValue == 0)
   {
      XmitTimerExpiration(TRANSPORT, TRUE);
      return;
   }
   /************************************************
     Send a new priority message event.
    ************************************************/
   else if (gp->priXmitRec.status == UNUSED_TX &&
            ! QueueEmpty(&gp->tsaOutPriQ)      &&
            ! QueueFull(&gp->nwOutPriQ) )
   {
      SendNewMsg(TRANSPORT, TRUE);
      return;
   }
   /************************************************
     Non-priority transaction timer expired event.
     ************************************************/
   else if (gp->xmitRec.status == TRANSPORT_TX &&
            gp->xmitRec.xmitTimer.curTimerValue == 0)
   {
      XmitTimerExpiration(TRANSPORT, FALSE);
   }
   /************************************************
     Send a new non-priority message.
    ************************************************/
   else if (gp->xmitRec.status == UNUSED_TX &&
            ! QueueEmpty(&gp->tsaOutQ)      &&
            ! QueueFull(&gp->nwOutQ) )
   {
      SendNewMsg(TRANSPORT, FALSE);
   }
   else
   {
      /* Either there is no work or there is no space. */
      return;
   }

   return;

}

/*****************************************************************
Function:  TerminateTrans
Returns:   None
Reference: None
```


--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

```
Purpose:   To terminate a transaction for transport or session
           layer and send the completion indication to application
           layer. If the application layer's input queue is full,
           we don't terminate the transaction.
Comments:  layerIn is not passed as it is not needed.
************************************************************/
static void TerminateTrans(Boolean priorityIn)
{
   Queue           *tsaQPtr;     /* Pointer to source queue */
   TSASendParam    *tsaSendParamPtr;
   APPReceiveParam *appReceiveParamPtr;
   TransmitRecord  *xmitRecPtr;  /* Ptr to xmit rec (pri or nonpri)*/
   Boolean          success;

   if (QueueFull(&gp->appInQ))
   {
      return; /* Can't send the indication. Come back later. */
   }

   if (priorityIn)
   {
      tsaQPtr         = &gp->tsaOutPriQ;
      tsaSendParamPtr = QueueHead(tsaQPtr);
      xmitRecPtr      = &gp->priXmitRec;
   }
   else
   {
      tsaQPtr         = &gp->tsaOutQ;
      tsaSendParamPtr = QueueHead(tsaQPtr);
      xmitRecPtr      = &gp->xmitRec;
   }

   appReceiveParamPtr = QueueTail(&gp->appInQ);

   if (tsaSendParamPtr->service == UNACK_RPT ||
       xmitRecPtr->destCount == xmitRecPtr->ackCount ||
       (xmitRecPtr->nwDestAddr.addressMode == BROADCAST &&
        xmitRecPtr->ackCount >= 1)
      )
   {
      /* UNACK_RPT or ACK and got all acks(or resp). */
      success = TRUE;
   }
   else
   {
      INCR_STATS(nmp->stats.transmitTXFailures);
      success = FALSE; /* REQUEST or ACK and did not get all acks. */
   }
   appReceiveParamPtr->indication = COMPLETION;
   appReceiveParamPtr->success    = success;
   appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
   EnQueue(&gp->appInQ);

   TransDone(priorityIn); /* Call to TCS. */
   xmitRecPtr->status = UNUSED_TX;
#ifdef DEBUG
   if (success)
   {
      DebugMsg("Debug: TermTran: Terminated the transaction. Success.");
   }
   else
   {
      DebugMsg("Debug: TermTran: Terminated the transaction. Fail.");
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
#endif
   /* Remove the transaction from the queue. */
   DeQueue(tsaQPtr);
   return;
}

/*************************************************************
Function:  XmitTimerExpiration
Returns:   None
Reference: None
Purpose:   To process the XmitTimer expiration event (pri or nonpri).
           Retranmission and termination of transaction are
           handled. Retransmission might be reply to already
           initiated challenge or it might be the APDU itself.
           If there is no space for retransmission, the retry is
           lost.
Comments:  None
*************************************************************/
static void XmitTimerExpiration(Layer layerIn, Boolean priorityIn)
{
   TSASendParam *tsaSendParamPtr;/* Param in tsaQ (Pri or nonPri).   */
   NWSendParam  *nwSendParamPtr; /* Param in nwQ (Pri or nonPri).    */
   TransmitRecord  *xmitRecPtr;  /* Ptr to xmit rec (pri or nonpri). */
   Queue        *tsaQPtr;        /* Pointer to source queue.         */
   Queue        *nwQPtr;         /* Pointer to target queue.         */
   TSPDUPtr      pduPtr;         /* Pointer to TSPDU being formed.   */
   uint8         deltaBL;
   uint16        pduSize;
   int8          i;
   uint8         length;         /* For length of reminder in bytes. */
   uint16        queueSpace;

   if (priorityIn)
   {
      tsaQPtr         = &gp->tsaOutPriQ;
      tsaSendParamPtr = QueueHead(tsaQPtr);
      nwQPtr          = &gp->nwOutPriQ;
      nwSendParamPtr  = QueueTail(nwQPtr);
      xmitRecPtr      = &gp->priXmitRec;
   }
   else
   {
      tsaQPtr         = &gp->tsaOutQ;
      tsaSendParamPtr = QueueHead(tsaQPtr);
      nwQPtr          = &gp->nwOutQ;
      nwSendParamPtr  = QueueTail(nwQPtr);
      xmitRecPtr      = &gp->xmitRec;
   }

   /* First, check if we really need to retry the message. */
   if (xmitRecPtr->retriesLeft == 0 ||
       xmitRecPtr->destCount == xmitRecPtr->ackCount ||
       (xmitRecPtr->nwDestAddr.addressMode == BROADCAST &&
        xmitRecPtr->ackCount >= 1)
      )
   {
      /* No More retries left or all acks have been received.
         Terminate the transaction. Send indication to the application
         layer. */
      TerminateTrans(priorityIn);
      return;
   }

   /* Check if there is space in the network buffer for retransmission */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   if (QueueFull(nwQPtr))
   {
      /* We are losing a retry chance locally due to lack of space
         in network queue. If we don't want to lose the retry, we
         simply delete the next two lines of code */
      xmitRecPtr->retriesLeft--;
      /* Start the transmit timer */
      SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);
#ifdef DEBUG
      DebugMsg("XmitTimerExp: Retry failure due to no space in net");
#endif
      return;
   }

   /* Now, we need to retransmit the message again. */
   /* Form the PDU to be sent directly in the target queue. */
   pduPtr              = (TSPDUPtr) (nwSendParamPtr + 1);
   pduPtr->auth        = tsaSendParamPtr->auth;
   pduPtr->transNum    = xmitRecPtr->transNum;

   if (tsaSendParamPtr->service == UNACK_RPT)
   {
      pduPtr->pduMsgType     = UNACK_RPT_MSG;
      memcpy(pduPtr->data, xmitRecPtr->apdu,
             xmitRecPtr->apduSize);
      pduSize = xmitRecPtr->apduSize + 1;
#ifdef DEBUG
      DebugMsg("XmitTimerExp: Resending UNACK_RPT packet.");
#endif
   }
   else if (xmitRecPtr->nwDestAddr.addressMode != MULTICAST)
   {
      if (layerIn == TRANSPORT)
      {
         pduPtr->pduMsgType = ACKD_MSG;
#ifdef DEBUG
         DebugMsg("XmitTimerExp: Resending ACKD packet.");
#endif
      }
      else if (tsaSendParamPtr->service == REQUEST)
      {
         pduPtr->pduMsgType = REQUEST_MSG;
#ifdef DEBUG
         DebugMsg("XmitTimerExp: Resending REQUEST packet.");
#endif
      }
      else
      {
         /* Response Messages are retried. Something is wrong. */
         /* Force retriesLeft to 0 so that next time we will
            terminate the transaction. */
         xmitRecPtr->retriesLeft = 0;
#ifdef DEBUG
         DebugMsg("XmitTimerExp: Response Message??. What is wrong?");
#endif
         return;
      }
      memcpy(pduPtr->data, xmitRecPtr->apdu,
             xmitRecPtr->apduSize);
      pduSize = xmitRecPtr->apduSize + 1;
   }
   else
   {
      /* Multicast Retry */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      /* Compute the highest numbered group member that has
         acknowledged and form the M_LIST up to that. Since the
         rest of the nodes have not acknowledged, they will
         respond when they see that their bit is missing.
         However, the last byte of M_LIST should be padded with
         0 as those members need to acknowledge. Note that a node
         will respond if its member number is not even present
         in the M_List. */
      /* Group members are 0..MAX_GROUP_NUMBER */
      if (xmitRecPtr->ackCount == 0)
      {
         length = 0;
      }
      else
      {
         /* ackCount > 0, So, we have at least one ack.
            Find the highest member who have responded. */
         for (i = MAX_GROUP_NUMBER; i >= 0; i--)
         {
            if (xmitRecPtr->ackReceived[i])
            {
               break;
            }
         }
         if (i == -1)
         {
            /* There should have been at least one ack. */
            ErrorMsg("XmitTimerExpiration: Something is wrong."
                     " Check code. Atleast one ack member expected.");
            xmitRecPtr->retriesLeft = 0;
            return;
         }
         length = i / 8 + 1; /* Number of bytes in M_List. */
      }

      if (length <= 2)
      {
         pduPtr->pduMsgType     = REM_MSG_MSG;
         pduPtr->data[0]     = length; /* # of bytes in M_List. */
         /* Copy the M_LIST. See Fig 8.2 in Protocol Specification. */
         /* We use the fact that ackReceived[i] is 0 or 1. */
         /* The padding of 0's of last byte is automatic as
            ackReceived[i] for those are anyway 0. */
         pduPtr->data[1] = 0; /* Init anyway even if not used. */
         pduPtr->data[2] = 0; /* Init anyway even if not used. */
         for (i = 0; i < 8*length; i++)
         {
            pduPtr->data[1 + i / 8] |=
               (xmitRecPtr->ackReceived[i] << (i % 8));
         }
         /* Copy APDU. */
         memcpy(&pduPtr->data[1+length],
                xmitRecPtr->apdu,
                xmitRecPtr->apduSize);
         /* TSPDU = 1 byte header + 1 byte for length + M_LIST. */
         pduSize = xmitRecPtr->apduSize + 2 + length;
#ifdef DEBUG
         DebugMsg("XmitTimerExp: Resending REMINDER  packet.");
#endif
      }
      else
      {
         /* Length > 2 */
         /* A Pair is sent in this case. First, send the REMINDER
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            and then send the ACKD or REQUEST message. */
         /* In this case, we are going to send two msgs.
            So, we need to make sure that the queue has space
            for 2 msgs. If not, return and come back later to
            do this case. */
         queueSpace = QueueCnt(nwQPtr) - QueueSize(nwQPtr);
         if (queueSpace < 2)
         {
            /* We are losing a retry chance locally due to lack
               of space in network queue. */
            xmitRecPtr->retriesLeft--;
            /* Start the transmit timer. */
            SetMsTimer(&xmitRecPtr->xmitTimer,
                    xmitRecPtr->xmitTimerValue);
#ifdef DEBUG
            DebugMsg("XmitTimerExp: Retry failure due to no"
                    " space in network buffer.");
#endif
            return; /* Not enough space in the queue. Come back. */
         }
         /* Send the REMINDER message. */
         if (tsaSendParamPtr->service == ACKD ||
             tsaSendParamPtr->service == REQUEST)
         {
            nwSendParamPtr = QueueTail(nwQPtr);
            nwSendParamPtr->dropIfUnconfigured = TRUE;
            pduPtr = (TSPDUPtr)
               ((char *)nwSendParamPtr + sizeof(NWSendParam));

            pduPtr->auth        = tsaSendParamPtr->auth;
            pduPtr->pduMsgType  = REMINDER_MSG;
            pduPtr->transNum    = xmitRecPtr->transNum;
            pduPtr->data[0]     = length;

            /* Copy the M_LIST. See Fig 8.2 in Protocol Specification. */
            /* First, initialize all the M_LIST fields to 0. */
            for (i = 1; i <= length; i++)
            {
               pduPtr->data[i] = 0;
            }
            /* Set the bits for M_LIST field based on received acks. */
            for (i = 0; i < 8*length; i++)
            {
               pduPtr->data[1 + i / 8] |=
                  (xmitRecPtr->ackReceived[i] << (i % 8));
            }

            pduSize = 2 + length; /* REMINDER has no APDU. */

            /* Fill in the NWSendParam structure. */
            nwSendParamPtr->destAddr = xmitRecPtr->nwDestAddr;
            if (layerIn == TRANSPORT)
            {
               nwSendParamPtr->pduType   = TPDU_TYPE;
            }
            else
            {
               nwSendParamPtr->pduType   = SPDU_TYPE;
            }
            nwSendParamPtr->deltaBL  = 0; /* REMINDER has deltaBL 0. */
            nwSendParamPtr->pduSize  = pduSize;

            /* UnAck_rpt packets do not use alt path. */
            if (tsaSendParamPtr->service != UNACK_RPT)
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            {
               nwSendParamPtr->altPath  =
                        (xmitRecPtr->retriesLeft <= (ALT_PATH_COUNT + 1));
            }
            else
            {
               nwSendParamPtr->altPath  = FALSE;
            }

            /* if altPath has override, use it */
            if (tsaSendParamPtr->altPathOverride)
            {
               nwSendParamPtr->altPath = tsaSendParamPtr->altPath;
            }

            /* Add TSPDU into the queue. */
            EnQueue(nwQPtr);
         }

         /* Send the ACKD or REQUEST. */
         nwSendParamPtr = QueueTail(nwQPtr);
         nwSendParamPtr->dropIfUnconfigured = TRUE;
         pduPtr = (TSPDUPtr)
            ((char *)nwSendParamPtr + sizeof(NWSendParam));
         pduPtr->auth        = tsaSendParamPtr->auth;
         pduPtr->transNum    = xmitRecPtr->transNum;
         if (tsaSendParamPtr->service == ACKD)
         {
            pduPtr->pduMsgType     = ACKD_MSG;
         }
         else
         {
            pduPtr->pduMsgType     = REQUEST_MSG;
         }
         memcpy(pduPtr->data,
                xmitRecPtr->apdu,
                xmitRecPtr->apduSize);
         pduSize = xmitRecPtr->apduSize + 1;

#ifdef DEBUG
         DebugMsg("XmitTimerExp: Resending REM/MSG pair.");
#endif
      }
   }


   if (tsaSendParamPtr->service != UNACK_RPT)
   {
      INCR_STATS(nmp->stats.transmitTXRetries);
   }

   /* Compute the delta backlog value. */
   deltaBL = 1; /* for subnet and unique id messages. */
   if (tsaSendParamPtr->service == UNACK_RPT)
   {
      deltaBL = 0; /* Only on first attempt, deltaBL is retries left. */
   }
   else if (xmitRecPtr->nwDestAddr.addressMode == BROADCAST)
   {
      /* Domainwide or subnet BROADCAST. */
      /* If there is no override value for deltaBL, then it is 15. */
      if (tsaSendParamPtr->destAddr.bcast.backlog)
      {
         deltaBL = tsaSendParamPtr->destAddr.bcast.backlog;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      }
      else
      {
         deltaBL = 15;
      }
   }
   else if (xmitRecPtr->nwDestAddr.addressMode == MULTICAST)
   {
      /* deltaBL is outstanding responses or acknowledgements. */
      deltaBL = xmitRecPtr->destCount - xmitRecPtr->ackCount;
   }

   /* Fill in the NWSendParam structure */
   nwSendParamPtr->destAddr = xmitRecPtr->nwDestAddr;
   if (layerIn == TRANSPORT)
   {
      nwSendParamPtr->pduType   = TPDU_TYPE;
   }
   else
   {
      nwSendParamPtr->pduType   = SPDU_TYPE;
   }
   nwSendParamPtr->deltaBL  = deltaBL;

   /* UnAck_rpt packets do not use alt path. */
   if (tsaSendParamPtr->service != UNACK_RPT)
   {
      nwSendParamPtr->altPath  =
               (xmitRecPtr->retriesLeft <= (ALT_PATH_COUNT + 1));
   }
   else
   {
      nwSendParamPtr->altPath  = FALSE;
   }
   /* if altPath has override, use it */
   if (tsaSendParamPtr->altPathOverride)
   {
      nwSendParamPtr->altPath = tsaSendParamPtr->altPath;
   }


   nwSendParamPtr->pduSize  = pduSize;

   xmitRecPtr->retriesLeft--;

   /* Add TSPDU into the queue. */
   EnQueue(nwQPtr);

   /* Start the transmit timer. */
   SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);

   return;
}

/*************************************************************
Function:  SendNewMsg
Returns:   None
Reference: None
Purpose:   To process a new request from the application layer that
           is in the tsa output queue (pri or nonpri). Request or ACKD.
Comments:  This fn is called only if there is space in the
           corresponding queue of the network layer.
*************************************************************/
static void SendNewMsg(Layer layerIn, Boolean priorityIn)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
{
   Queue          *tsaQPtr;        /* Pointer to the source queue.    */
   TSASendParam   *tsaSendParamPtr;/* Param in tsaQ (Pri or non-pri). */
   Queue          *nwQPtr;         /* Pointer to target queue.        */
   NWSendParam    *nwSendParamPtr; /* Param in nwQ (Pri or non-pri).  */
   TransmitRecord *xmitRecPtr;     /* Ptr to xmit rec.                */
   DestinationAddress nwDestAddr;  /* Destination address.            */
   TSPDUPtr        pduPtr;         /* Pointer to TSPDU being formed.  */
   APPReceiveParam *appReceiveParamPtr;
   Status          status;
   uint16          rptTimer;
   uint8           retryCount;
   uint16          txTimer;
   uint8           deltaBL;
   uint16          nwBufSize;
   int8            i;

   if (priorityIn)
   {
      tsaQPtr         = &gp->tsaOutPriQ;
      tsaSendParamPtr = QueueHead(tsaQPtr);
      nwQPtr          = &gp->nwOutPriQ;
      nwSendParamPtr  = QueueTail(nwQPtr);
      nwBufSize       = gp->nwOutPriBufSize;
      xmitRecPtr      = &gp->priXmitRec;
   }
   else
   {
      tsaQPtr         = &gp->tsaOutQ;
      tsaSendParamPtr = QueueHead(tsaQPtr);
      nwQPtr          = &gp->nwOutQ;
      nwSendParamPtr  = QueueTail(nwQPtr);
      nwBufSize       = gp->nwOutBufSize;
      xmitRecPtr      = &gp->xmitRec;
   }

   /* If processing a new message, make sure that it is for this
      layer. If not, we are done. */

   if (layerIn == TRANSPORT &&
       tsaSendParamPtr->service != ACKD &&
       tsaSendParamPtr->service != UNACK_RPT)
   {
      return;
   }

   if (layerIn == TRANSPORT && NV_LAST_TAG(tsaSendParamPtr->tag))
   {
      if (QueueFull(&gp->appInQ))
      {
         return;
      }
      /* Special tag used by application layer for synchronization.
         Send completion indication right away. */
      appReceiveParamPtr             = QueueTail(&gp->appInQ);
      appReceiveParamPtr->indication = COMPLETION;
      appReceiveParamPtr->success    = TRUE;
      appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
      EnQueue(&gp->appInQ);
      DeQueue(tsaQPtr);
      return;
   }

   if (layerIn == SESSION && tsaSendParamPtr->service != REQUEST)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   {
      /* Responses are placed in the response queue. */
      return;
   }

   /* Make sure that large group size is not used for ack
      or request service. */
   if (tsaSendParamPtr->destAddr.group.groupFlag &&
       tsaSendParamPtr->destAddr.group.groupSize == 0 &&
       tsaSendParamPtr->service != UNACK_RPT)
   {
      /* Large groups can only use unack or unack_rpt services. */
      /* Indicate failure of this message to application layer. */
      if (!QueueFull(&gp->appInQ))
      {
         appReceiveParamPtr             = QueueTail(&gp->appInQ);
         appReceiveParamPtr->indication = COMPLETION;
         appReceiveParamPtr->success    = FALSE;
         appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
         EnQueue(&gp->appInQ);
         DeQueue(tsaQPtr);
      }
      else
      {
         /* We wait until appInQ has space for the indication. */
      }
      return;
   }
   /* Make sure that groupSize is in the proper range. */
   /* *** START INFORMATIVE - Group Size *** */
   /* See "START INFORMATIVE - Group Size" below. */
#ifdef GROUP_SIZE_COMPATIBILITY
   if (tsaSendParamPtr->destAddr.group.groupFlag &&
       (tsaSendParamPtr->destAddr.group.groupSize == 1 ||
        tsaSendParamPtr->destAddr.group.groupSize > MAX_GROUP_NUMBER+1) )
#else
   if (tsaSendParamPtr->destAddr.group.groupFlag &&
        tsaSendParamPtr->destAddr.group.groupSize > MAX_GROUP_NUMBER)
#endif
   /* *** END INFORMATIVE - Group Size *** */
   {
      /* Indicate failure of this message to application layer. */
      if (!QueueFull(&gp->appInQ))
      {
         appReceiveParamPtr             = QueueTail(&gp->appInQ);
         appReceiveParamPtr->indication = COMPLETION;
         appReceiveParamPtr->success    = FALSE;
         appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
         EnQueue(&gp->appInQ);
         DeQueue(tsaQPtr);
      }
      else
      {
         /* We wait until appInQ has space for indication. */
      }
      return;
   }

   /* Make sure there is space in network buffer. If not, we fail. */
   /* apdu + (tran or session header of 1 byte) should fit. */
   if ((tsaSendParamPtr->apduSize + 1) > nwBufSize)
   {
      /* We can't send this message as it is too big for
         the network layer's buffer. */
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* Right now, we haven't allocated any transmit record. */
   /* So, we directly give the indication to application. */
   if (!QueueFull(&gp->appInQ))
   {
      appReceiveParamPtr = QueueTail(&gp->appInQ);
      appReceiveParamPtr->indication = COMPLETION;
      appReceiveParamPtr->success    = FALSE;
      appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
      EnQueue(&gp->appInQ);
      DeQueue(tsaQPtr);
   }
   else
   {
      /* We wait until appInQ has space for indication. */
   }
   return;
}

/* First, compute nwDestAddr from destAddr. */
/* First, initialize domainIndex. Only if it is COMPUTE_DOMAIN_INDEX,
   we need to recompute it based on destAddr field value. */
nwDestAddr.domainIndex = tsaSendParamPtr->domainIndex;
switch (tsaSendParamPtr->destAddr.noAddress)
{
   case UNBOUND:
      /* Not in use or turnaround format. */
      if (!QueueFull(&gp->appInQ))
      {
         ErrorMsg("SendNewMsg: UNBOUND destination address is invalid.");
         appReceiveParamPtr = QueueTail(&gp->appInQ);
         appReceiveParamPtr->indication = COMPLETION;
         appReceiveParamPtr->success    = FALSE;
         appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
         EnQueue(&gp->appInQ);
         DeQueue(tsaQPtr);
      }
      else
      {
         /* We wait until appInQ has space for indication. */
      }
      return;
   case SUBNET_NODE:
      nwDestAddr.addressMode = SUBNET_NODE;
      if (tsaSendParamPtr->domainIndex == COMPUTE_DOMAIN_INDEX)
      {
         /* Use the domainIndex from destAddr. */
         nwDestAddr.domainIndex =
            tsaSendParamPtr->destAddr.snode.domainIndex;
      }
      nwDestAddr.addr.addr2a.subnet =
         tsaSendParamPtr->destAddr.snode.subnetID;
      nwDestAddr.addr.addr2a.selField = 1; /* always 1 */
      nwDestAddr.addr.addr2a.node   =
         tsaSendParamPtr->destAddr.snode.node;
      txTimer =
         DecodeTxTimer((uint8) tsaSendParamPtr->destAddr.snode.txTimer);
      rptTimer =
         DecodeRptTimer((uint8) tsaSendParamPtr->destAddr.snode.rptTimer);
      retryCount =
         tsaSendParamPtr->destAddr.snode.retryCount;
      break;
   case UNIQUE_NODE_ID:
      nwDestAddr.addressMode = UNIQUE_NODE_ID;
      if (tsaSendParamPtr->domainIndex == COMPUTE_DOMAIN_INDEX)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
         {
            nwDestAddr.domainIndex =
               tsaSendParamPtr->destAddr.uniqueNodeId.domainIndex;
         }
         nwDestAddr.addr.addr3.subnet =
            tsaSendParamPtr->destAddr.uniqueNodeId.subnetID;
         memcpy(nwDestAddr.addr.addr3.uniqueId,
                tsaSendParamPtr->destAddr.uniqueNodeId.uniqueId,
                UNIQUE_NODE_ID_LEN);
         txTimer =
            DecodeTxTimer((uint8) tsaSendParamPtr->destAddr.uniqueNodeId.txTimer);
         rptTimer =
            DecodeRptTimer((uint8) tsaSendParamPtr-
>destAddr.uniqueNodeId.rptTimer);
         retryCount =
            tsaSendParamPtr->destAddr.uniqueNodeId.retryCount;
         break;
      case BROADCAST:
         /* The following field is not used in this mode by
            application. It is however used in BROADCAST_GROUP. */
         tsaSendParamPtr->destAddr.bcast.maxResponses = 1;
         /* Fall Through */
      /* *** START INFORMATIVE - Broadcast Group */
      /* Broadcast group addressing is optional.  It can not be assumed that all
devices
       * support this form of addressing. */
      case BROADCAST_GROUP:
         nwDestAddr.addressMode = BROADCAST;
         if (tsaSendParamPtr->domainIndex == COMPUTE_DOMAIN_INDEX)
         {
            nwDestAddr.domainIndex =
               tsaSendParamPtr->destAddr.bcast.domainIndex;
         }
         nwDestAddr.addr.addr0 =
            tsaSendParamPtr->destAddr.bcast.subnetID;
         txTimer =
            DecodeTxTimer((uint8) tsaSendParamPtr->destAddr.bcast.txTimer);
         rptTimer =
            DecodeRptTimer((uint8) tsaSendParamPtr->destAddr.bcast.rptTimer);
         retryCount =
            tsaSendParamPtr->destAddr.bcast.retryCount;
         break;
      /* *** END INFORMATIVE - Broadcast Group */
      default:
         /* Must be group format unless it is an invalid value. */
         if (tsaSendParamPtr->destAddr.group.groupFlag != 1)
         {
            /* It must be some invalid value. Let us fail. */
            nmp->errorLog = BAD_ADDRESS_TYPE;
            if (!QueueFull(&gp->appInQ))
            {
               ErrorMsg("SendNewMsg: Invalid group format addr.");
               appReceiveParamPtr = QueueTail(&gp->appInQ);
               appReceiveParamPtr->indication = COMPLETION;
               appReceiveParamPtr->success    = FALSE;
               appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
               EnQueue(&gp->appInQ);
               DeQueue(tsaQPtr);
            }
            else
            {
               /* We wait until appInQ has space for indication. */
            }
            return;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      }
      nwDestAddr.addressMode = MULTICAST;
      if (tsaSendParamPtr->domainIndex == COMPUTE_DOMAIN_INDEX)
      {
         nwDestAddr.domainIndex =
            tsaSendParamPtr->destAddr.group.domainIndex;
      }
      nwDestAddr.addr.addr1 =
         tsaSendParamPtr->destAddr.group.groupID;
      txTimer =
         DecodeTxTimer((uint8) tsaSendParamPtr->destAddr.group.txTimer);
      rptTimer =
         DecodeRptTimer((uint8) tsaSendParamPtr->destAddr.group.rptTimer);
      retryCount =
         tsaSendParamPtr->destAddr.group.retryCount;
} /* switch */

if (nwDestAddr.domainIndex == FLEX_DOMAIN)
{
   /* flex domain. Copy the domain id. */
   nwDestAddr.flexDomainLen = tsaSendParamPtr->flexDomainLen;
   if (tsaSendParamPtr->flexDomainLen <= DOMAIN_ID_LEN)
   {
      memcpy(nwDestAddr.flexDomainId,
             tsaSendParamPtr->flexDomainId,
             tsaSendParamPtr->flexDomainLen);
   }
}
/* Get transaction number using nwDestAddr. */
status = NewTrans(priorityIn, nwDestAddr, &xmitRecPtr->transNum);
if (status == FAILURE)
{
   /* Unable to get the transaction number. Give up. Try later. */
   return;
}

/* Initialize the xmit record. */
if (layerIn == TRANSPORT)
{
   xmitRecPtr->status = TRANSPORT_TX;
}
else
{
   xmitRecPtr->status = SESSION_TX;
}
xmitRecPtr->nwDestAddr = nwDestAddr; /* Save it for future use. */
for (i = 0; i <= MAX_GROUP_NUMBER; i++)
{
   xmitRecPtr->ackReceived[i] = FALSE;
}

if (nwDestAddr.addressMode == MULTICAST)
{
   /* If the node is a member of the group, then groupsize
      field is set to 1 more than actual group size
      by in app layer or app pgm.
      However, provide an option to set this to the
      true group size and transport and session layers
      will take care of this. */
   xmitRecPtr->destCount =
      tsaSendParamPtr->destAddr.group.groupSize - 1;
   /* *** START INFORMATIVE - Group Size *** */
   /* To be fully compatible with all applications, you must use the
       GROUP_SIZE_COMPATIBILITY option.  This behavior only affects explicitly
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
           addressed messages originated by the application. */
#ifndef GROUP_SIZE_COMPATIBILITY
      if (!IsGroupMember(tsaSendParamPtr->destAddr.group.domainIndex,
                         tsaSendParamPtr->destAddr.group.group,
                         &groupMember))
      {
         /* node is not a member & group size is true size */
         xmitRecPtr->destCount =
            tsaSendParamPtr->destAddr.group.groupSize;
      }
#endif
      /* *** END INFORMATIVE - Group Size *** */
      if (xmitRecPtr->destCount == 0)
      {
         /* If the value is incorrect, set it to 1. We need at least
            one acknowledgement or response. */
#ifdef DEBUG
         DebugMsg("SendNewMsg: groupSize incorrect. Default assumed.");
#endif
         xmitRecPtr->destCount = 1;
      }
   }
   else
   {
      xmitRecPtr->destCount = 1;
   }

   xmitRecPtr->retriesLeft = retryCount;
   xmitRecPtr->ackCount    = 0;
   xmitRecPtr->apdu        = (APDU *) (tsaSendParamPtr + 1);
   xmitRecPtr->apduSize    = tsaSendParamPtr->apduSize;
   if (tsaSendParamPtr->service == UNACK_RPT)
   {
      xmitRecPtr->xmitTimerValue = rptTimer;
   }
   else
   {
      xmitRecPtr->xmitTimerValue = txTimer;
   }

   /* Form the TSPDU to be sent directly in queue. */
   pduPtr              = (TSPDUPtr)
      ((char *)nwSendParamPtr + sizeof(NWSendParam));
   pduPtr->auth        = tsaSendParamPtr->auth;
   /* Save auth status so that if a challenge comes later on
      we can at least verify that it was legitimate challenge */
   xmitRecPtr->auth    = tsaSendParamPtr->auth;
   if (tsaSendParamPtr->service == UNACK_RPT)
   {
      pduPtr->pduMsgType     = UNACK_RPT_MSG;
#ifdef DEBUG
         DebugMsg("SendNewMsg: Sending a new UNACK_RPT packet.");
#endif
   }
   else if (layerIn == TRANSPORT)
   {
      pduPtr->pduMsgType     = ACKD_MSG;
#ifdef DEBUG
         DebugMsg("SendNewMsg: Sending a new ACKD packet.");
#endif
   }
   else if (tsaSendParamPtr->service == REQUEST)
   {
      pduPtr->pduMsgType     = REQUEST_MSG;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
#ifdef DEBUG
         DebugMsg("SendNewMsg: Sending a new REQ packet.");
#endif
   }
   else
   {
#ifdef DEBUG
      DebugMsg("SendNewMsg: Something is wrong. Unknown service.");
#endif
   }
   pduPtr->transNum    = xmitRecPtr->transNum;
   memcpy(pduPtr->data, xmitRecPtr->apdu,
          xmitRecPtr->apduSize);

   /* Compute the delta backlog value. */
   deltaBL = 1; /* For subnet and unique node id messages. */
   if (tsaSendParamPtr->service == UNACK_RPT_MSG)
   {
      deltaBL = xmitRecPtr->retriesLeft;
   }
   else if (nwDestAddr.addressMode == BROADCAST)
   {
      if (tsaSendParamPtr->destAddr.bcast.backlog)
      {
         deltaBL = tsaSendParamPtr->destAddr.bcast.backlog;
      }
      else
      {
         deltaBL = 15;
      }
   }
   else if (nwDestAddr.addressMode == MULTICAST)
   {
      deltaBL = xmitRecPtr->destCount;
   }

   /* Fill in the NWSendParam structure. */
   nwSendParamPtr->dropIfUnconfigured = TRUE;
   nwSendParamPtr->destAddr = nwDestAddr;
   if (layerIn == TRANSPORT)
   {
      nwSendParamPtr->pduType  = TPDU_TYPE;
   }
   else
   {
      nwSendParamPtr->pduType  = SPDU_TYPE;
   }
   nwSendParamPtr->deltaBL  = deltaBL;

   /* UnAck_rpt packets do not use alternate path. */
   if (tsaSendParamPtr->service != UNACK_RPT)
   {
      nwSendParamPtr->altPath  = (retryCount <= ALT_PATH_COUNT);
   }
   else
   {
      nwSendParamPtr->altPath  = FALSE;
   }

   /* if altPath has override, use it */
   if (tsaSendParamPtr->altPathOverride)
   {
      nwSendParamPtr->altPath = tsaSendParamPtr->altPath;
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   nwSendParamPtr->pduSize  = xmitRecPtr->apduSize + 1;

   /* Add the TSPDU into the queue. */
   EnQueue(nwQPtr);

   /* Start the transmit timer. */
   SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);

   return;
}


/*************************************************************
Function:  TPReceive
Returns:   None
Reference: None
Purpose:   To receive and process incoming TPDU's by calling
           apprpriate functions. Also handle receive timers.
Comments:  None
*************************************************************/
void TPReceive(void)
{
   TSAReceiveParam *tsaReceiveParamPtr;/* Param in tsaQ. */
   TSPDUPtr          pduInPtr;         /* Pointer to TPDU received. */
   int16           i;

   /* Update all the receive timers, if they do exist */
   for (i = 0; i < gp->recvRecCnt; i++)
   {
      if (gp->recvRec[i].status == TRANSPORT_RR)
      {
         if (gp->recvRec[i].recvTimer.curTimerValue > 0)
         {
            UpdateMsTimer(&gp->recvRec[i].recvTimer);
         }
         if (gp->recvRec[i].recvTimer.curTimerValue == 0)
         {
            /* Timer expired. See if RR can be released. */
#ifdef DEBUG
            DebugMsg("TPReceive: Receive timer expired.");
#endif

            /* Deliver the message if it has not been delivered. */
            /* Deliver fn checks for authentication */
            if (gp->recvRec[i].transState == DELIVERED ||
                gp->recvRec[i].transState == DONE)
            {
               gp->recvRec[i].status = UNUSED_RR; /* Release the RR */
            }
            else
            {
               Deliver(i);
               if (gp->recvRec[i].serviceType == ACKD &&
                   gp->recvRec[i].transState == DELIVERED)
               {
                  TPSendAck(i); /* may get lost due to no space. */
               }
            }
         }
         else if ((gp->recvRec[i].transState == JUST_RECEIVED &&
                                              !gp->recvRec[i].needAuth) ||
                  gp->recvRec[i].transState == AUTHENTICATED )
         {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            /* The request does not need authentication and just received
               or it has been authenticated */
            Deliver(i);
            if (gp->recvRec[i].serviceType == ACKD &&
                gp->recvRec[i].transState == DELIVERED)
            {
               TPSendAck(i); /* may get lost due to no space. */
            }
            /* We can't release RR as it is not expired yet. */
         }
      }
   }

   /* Check if there is TPDU to be processed. */
   if (QueueEmpty(&gp->tsaInQ))
   {
      /* There is nothing to process. */
      return;
   }

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduInPtr           = (TSPDUPtr) (tsaReceiveParamPtr + 1);

   /* Check if the PDU is for transport layer. If not, we are done. */
   if (tsaReceiveParamPtr->pduType != TPDU_TYPE)
   {
      return;
   }

   /* We have a TPDU to be processed. Check the type and
      process accordingly. */
   switch (pduInPtr->pduMsgType)
   {
      case ACK_MSG:
         TPReceiveAck();
         return;
      case ACKD_MSG:
         /* Fall through. */
      case UNACK_RPT_MSG:
         ReceiveNewMsg(TRANSPORT);
         return;
      case REMINDER_MSG:
         /* Fall through. */
      case REM_MSG_MSG:
         ReceiveRem(TRANSPORT);
         return;
      default:
         ErrorMsg("TPReceive: Unknown TPDU type received.");
         DeQueue(&gp->tsaInQ);
         nmp->errorLog = UNKNOWN_PDU;
   }

   return;
}

/*****************************************************************
Function:  TPReceiveAck
Returns:   None
Reference: None
Purpose:   To process ACK message received by the transport layer.
Comments:  ACK might correspond to a transmit record or it may be
           stale, in which case we ignore it.
*****************************************************************/
static void TPReceiveAck(void)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
{
   TSAReceiveParam *tsaReceiveParamPtr;
   TSPDUPtr          pduPtr;
   TransmitRecord  *xmitRecPtr;  /* Ptr to xmit rec (pri or nonpri). */

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduPtr             = (TSPDUPtr) (tsaReceiveParamPtr + 1);

   /* Set the corresponding transmit pointer. */
   if (tsaReceiveParamPtr->priority)
   {
      xmitRecPtr = &gp->priXmitRec;
   }
   else
   {
      xmitRecPtr = &gp->xmitRec;
   }

   if (ValidateTrans(tsaReceiveParamPtr->priority, pduPtr->transNum)
       == TRANS_NOT_CURRENT)
   {
      /* Stale ACK. Ignore it. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: TPReceiveAck: Stale Ack Discarded");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
   }

   /* Check if the ACK really corresponds to the transaction in progress. */
   if (xmitRecPtr->status != TRANSPORT_TX ||
       xmitRecPtr->nwDestAddr.domainIndex !=
       tsaReceiveParamPtr->srcAddr.domainIndex)
   {
      /* Transmit record is not ours or the domain index for Ack
         and the transaction do not match. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: TPReceiveAck: Stale ack discarded.");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
   }

   if (xmitRecPtr->nwDestAddr.addressMode == SUBNET_NODE &&
       (xmitRecPtr->nwDestAddr.addr.addr2a.subnet !=
        tsaReceiveParamPtr->srcAddr.subnetAddr.subnet ||
        xmitRecPtr->nwDestAddr.addr.addr2a.node !=
        tsaReceiveParamPtr->srcAddr.subnetAddr.node))
   {
      /* Source address of ACK does not match destination address of
         transmit record. */
      /* ACK does not seem to correspond to what we are expecting. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: TPReceiveAck: Stale Ack discarded.");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
   }

   if (xmitRecPtr->nwDestAddr.addressMode == MULTICAST &&
       (tsaReceiveParamPtr->srcAddr.addressMode != MULTICAST_ACK ||
```

```
        xmitRecPtr->nwDestAddr.addr.addr1 !=
        tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.group))
   {
      /* MULTICAST message but not MULTICAST_ACK or the ack's group
         # does not match that of xmitRecord. */
      /* ACK does not seem to correspond to what we are expecting. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: TPReceiveAck: Stale ack discarded.");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
   }

   /* For broadcast messages, ACK can come back with subnet value of 0
      from unconfigured nodes. We should accept these acks too. */
   if (xmitRecPtr->nwDestAddr.addressMode == BROADCAST &&
       tsaReceiveParamPtr->srcAddr.subnetAddr.subnet != 0 &&
       xmitRecPtr->nwDestAddr.addr.addr0 != 0 &&
       xmitRecPtr->nwDestAddr.addr.addr0 !=
          tsaReceiveParamPtr->srcAddr.subnetAddr.subnet)
   {
      /* Subnet broadcast but the ACK's subnet does not match .*/
      /* ACK does not seem to correspond to what we are expecting. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: TPReceiveAck: Stale ack discarded.");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
   }

   switch (xmitRecPtr->nwDestAddr.addressMode)
   {
      case BROADCAST:    /* Fall Through. */
      case SUBNET_NODE:  /* Fall Through. */
      case UNIQUE_NODE_ID:
          /* Normally the first ack should have terminated the transaction.
             If it did not, we can try to terminate again. There is no
             harm in doing this. Also, there is no harm in increment
             ackCount as XmitTimerExpiration checks for ackCount >= 1. */
          xmitRecPtr->ackCount++; /* Got one more ack. */
          TerminateTrans(tsaReceiveParamPtr->priority);
         break;
      case MULTICAST:
         /* Group acknowledgement. */
         if (tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member
             > MAX_GROUP_NUMBER)
         {
            ErrorMsg("TPReceiveAck: Invalid group number.");
            break;
         }
         if (!xmitRecPtr->ackReceived[
               tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member])
         {
            /* We did not receive this ack in past. */
            xmitRecPtr->ackReceived[
              tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member]
               = TRUE;
            xmitRecPtr->ackCount++;
#ifdef DEBUG
            DebugMsg("TPReceiveAck: A new multicast ACK recvd.");
#endif
         }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
         else
         {
            /* Else, it is a duplicate Ack. Ignore it. */
#ifdef DEBUG
            DebugMsg("TPReceiveAck: A Duplicate Mul. ACK ignored");
#endif
         }
         if (xmitRecPtr->destCount == xmitRecPtr->ackCount)
         {
            TerminateTrans(tsaReceiveParamPtr->priority);
         }
         break;
      default:
         nmp->errorLog = BAD_ADDRESS_TYPE;
         ErrorMsg("TPReceiveAck: Invalid address mode");
   }

#ifdef DEBUG
   DebugMsg("TPReceiveAck: Ack was received.");
#endif

   DeQueue(&gp->tsaInQ);

   /* Restart the Xmit Timer if the Xmit record is still active. */
   if (xmitRecPtr->status != UNUSED_TX)
   {
      SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);
   }

   return;
}


/*****************************************************************
Function:  SNReceiveResponse
Returns:   None
Reference: None
Purpose:   To process response message received by the session layer.
Comments:  Response should correspond to current transaction in
           progress Or else it is thrown away.
*****************************************************************/
static void SNReceiveResponse(void)
{
   TSAReceiveParam *tsaReceiveParamPtr;
   TSASendParam    *tsaSendParamPtr;
   TSPDUPtr         pduPtr;
   TransmitRecord  *xmitRecPtr;  /* Ptr to xmit rec (pri or nonpri). */
   APPReceiveParam *appReceiveParamPtr;
   APDU            *apduPtr;

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduPtr             = (TSPDUPtr) (tsaReceiveParamPtr + 1);

   /* Set the corresponding transmit pointer. */
   if (tsaReceiveParamPtr->priority)
   {
      xmitRecPtr      = &gp->priXmitRec;
      tsaSendParamPtr = QueueHead(&gp->tsaOutPriQ);
   }
   else
   {
      xmitRecPtr      = &gp->xmitRec;
      tsaSendParamPtr = QueueHead(&gp->tsaOutQ);
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   if (ValidateTrans(tsaReceiveParamPtr->priority, pduPtr->transNum)
       == TRANS_NOT_CURRENT)
   {
      /* Unsolicited response. Ignore it. */
#ifdef DEBUG
      DebugMsg("SNReceiveResponse: Unsolicited response ignored.");
#endif
      DeQueue(&gp->tsaInQ);
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
   }

   /* Check if possible if the Response really corresponds to the
      transaction in progress. */
   if (xmitRecPtr->status != SESSION_TX ||
       xmitRecPtr->nwDestAddr.domainIndex !=
       tsaReceiveParamPtr->srcAddr.domainIndex)
   {
      /* Transmit record is not ours or the domain indices for
         the response and the transaction record do not match. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: SNReceiveResponse: Stale response discarded.");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
   }

   if (xmitRecPtr->nwDestAddr.addressMode == SUBNET_NODE &&
       (xmitRecPtr->nwDestAddr.addr.addr2a.subnet !=
        tsaReceiveParamPtr->srcAddr.subnetAddr.subnet ||
        xmitRecPtr->nwDestAddr.addr.addr2a.node !=
        tsaReceiveParamPtr->srcAddr.subnetAddr.node))
   {
      /* Source address of response does not match dest addr of
         transmit record. */
      /* Response does not seem to correspond to what we are expecting. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: SNReceiveResponse: Stale response discarded.");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
   }

   if (xmitRecPtr->nwDestAddr.addressMode == MULTICAST &&
       (tsaReceiveParamPtr->srcAddr.addressMode != MULTICAST_ACK ||
        xmitRecPtr->nwDestAddr.addr.addr1 !=
        tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.group))
   {
      /* MULTICAST message but not MULTICAST_ACK or the response's group
         # does not match that of xmitRecord. */
      /* Response does not seem to correspond to what we are expecting. */
      DeQueue(&gp->tsaInQ);
#ifdef DEBUG
      DebugMsg("Debug: SNReceiveResponse: Stale response discarded.");
#endif
      INCR_STATS(nmp->stats.lateAcknowledgements);
      return;
   }

   /* For brodacast messages, the subnet in the response can be 0 from
      unconfigured nodes. Another possibility is that the response might
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   come through a router in a scenerio where the router replaces the
   subnet with its own. Thus the response can be valid even if the
   subnet does not match. So, we should skip this test. */


/* Check if there is space in application layer's input queue. */
if (QueueFull(&gp->appInQ))
{
   return; /* Can't give the response yet. Come back later. */
}

appReceiveParamPtr = QueueTail(&gp->appInQ);
apduPtr            = (APDU *)(appReceiveParamPtr + 1);

/* Deliver the partial response to application layer. */
appReceiveParamPtr->indication = MESSAGE;
appReceiveParamPtr->srcAddr    = tsaReceiveParamPtr->srcAddr;
appReceiveParamPtr->service    = RESPONSE;
appReceiveParamPtr->priority   = tsaReceiveParamPtr->priority;
appReceiveParamPtr->pduSize    = tsaReceiveParamPtr->pduSize - 1;
appReceiveParamPtr->auth       = FALSE;
appReceiveParamPtr->tag        = tsaSendParamPtr->tag;
memcpy(apduPtr, pduPtr->data, tsaReceiveParamPtr->pduSize - 1);
/* Don't deliver the response to application yet. It could be
   a duplicate. */
switch (xmitRecPtr->nwDestAddr.addressMode)
{
   case BROADCAST:
      /* We deliver up to N responses to app layer where N =
         tsaSendParamPtr->destAddr.bcast.maxResponses.
         But we succeed if at least one resp is received. */
      if (xmitRecPtr->ackCount <
          tsaSendParamPtr->destAddr.bcast.maxResponses)
      {
         EnQueue(&gp->appInQ);
         INCR_STATS(nmp->stats.layer6_7RespRcvd);
         xmitRecPtr->ackCount++;
         if (xmitRecPtr->ackCount ==
             tsaSendParamPtr->destAddr.bcast.maxResponses)
         {
            TerminateTrans(tsaReceiveParamPtr->priority);
         }
      }
      /* else, we don't want this response. Ignore it. */
      break;
   case SUBNET_NODE:  /* Fall through. */
   case UNIQUE_NODE_ID:
      if (xmitRecPtr->ackCount == 0)
      {
         EnQueue(&gp->appInQ);
         INCR_STATS(nmp->stats.layer6_7RespRcvd);
         xmitRecPtr->ackCount++; /* First response. */
         TerminateTrans(tsaReceiveParamPtr->priority);
      }
      /* else, it is a duplicate response. Ignore it. */
      break;
   case MULTICAST:
      /* Group request message. Check response address mode. */
      if (tsaReceiveParamPtr->srcAddr.addressMode
          != MULTICAST_ACK)
      {
         ErrorMsg("SNReceiveResponse: RESPONSE should be "
                  "MULTICAST_ACK.");
         break;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
         }
         if (tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member
             > MAX_GROUP_NUMBER)
         {
            ErrorMsg("SNReceiveResponse: Invalid group number.");
            break;
         }
         if (!xmitRecPtr->ackReceived[
               tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member])
         {
#ifdef DEBUG
            DebugMsg("SNReceiveResp: A multicast resp delivered.");
#endif
            EnQueue(&gp->appInQ);
            INCR_STATS(nmp->stats.layer6_7RespRcvd);
            xmitRecPtr->ackReceived[
              tsaReceiveParamPtr->srcAddr.ackNode.groupAddr.member]
               = TRUE;
            xmitRecPtr->ackCount++;
         }
         else
         {
            /* Else, it is a duplicate response. Ignore it */
#ifdef DEBUG
            DebugMsg("SNReceiveResp: A duplicate multicast response ignored.");
#endif
         }
         if (xmitRecPtr->destCount == xmitRecPtr->ackCount)
         {
            TerminateTrans(tsaReceiveParamPtr->priority);
         }
         break;
      default:
         nmp->errorLog = BAD_ADDRESS_TYPE;
         ErrorMsg("SNReceiveResponse: Invalid address mode.");
   }

   DeQueue(&gp->tsaInQ);

   /* Restart the timer if the xmit rec is still active. */
   if(xmitRecPtr->status != UNUSED_TX)
   {
      SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);
   }

   return;
}

/*************************************************************
Function:  ReceiveNewMsg
Returns:   None
Reference: None
Purpose:   To process either a new message received by transport
           (ACKD or UNACK_RPT) or session layer (REQUEST).
Comments:  The message could be new or a duplicate. If new, we
           need to allocate a RR. If not, we use the existing RR.
*************************************************************/
static void ReceiveNewMsg(Layer layerIn)
{
   TSAReceiveParam *tsaReceiveParamPtr;
   TSPDUPtr         pduPtr;
   uint16           recvTimerValue;
   int16            i;
   Boolean          initRR; /* Used to check if RR needs to init. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* We have a new msg in TSA In queue. */

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduPtr          = (TSPDUPtr) (tsaReceiveParamPtr + 1);

   /* First retrieve associated RR, if it exists. */
   /* Since we can receive at most one message from a node
      (one for priority & one for non-priority), we can reuse
      receive records if the sender determines that a message
      transmission is over by transmitting a new message with a
      new transaction number or different service (i.e layer) or
      a new APDU. */

   i = RetrieveRR(tsaReceiveParamPtr->srcAddr,
                  tsaReceiveParamPtr->priority);
   if (i == -1)
   {
      /* No Associated RR found. We need to allocate a new one. */
      i = AllocateRR();
      if (i == -1)
      {
         /* Unable to allocate a new RR. */
         INCR_STATS(nmp->stats.receiveTXFull);
#ifdef DEBUG
         DebugMsg("ReceiveNewMsg: Unable to allocate RR. Msg is lost.");
#endif
         DeQueue(&gp->tsaInQ); /* Remove item from queue. */
         return;
      }
      /* We did allocate a new RR. We need to init this RR. */
      initRR = TRUE;
   }
   else if (gp->recvRec[i].transNum != pduPtr->transNum ||
            gp->recvRec[i].status   != (layerIn ==
TRANSPORT?TRANSPORT_RR:SESSION_RR)
                                       || tsaReceiveParamPtr->pduSize - 1 !=
                                          gp->recvRec[i].apduSize  ||
         /* *** START INFORMATIVE - APDU Contents and Duplicate Detection *** */
         /* This check causes the contents of the APDU to always be factored I
          * the duplicate detection algorithm.
          * It is considered a requirement that the APDU contents be
          * factored into the duplicate detection algorithm only in the case where
the
          * message is authenticated and the request is idempotent (because retries
          * are not re-authenticated).  Whether this is done by storing the entire
          * APDU for comparison or storing only the APDU length and a checksum of
the
          * APDU data is implementation dependent.  If a checksum method is used,
the
          * checksum should be at least 24 bits in length.  In the case where a
          * non-matching APDU vectors into an existing authenticated transaction,
two
          * choices are possible:
          * a. Treat the message as if it were not a duplicate (as is done here).
          * b. Treat the message as a duplicate but mark it as not authenticated
          *    (if it is necessary to deliver the retry to the application - see
          *    "START INFORMATIVE - Saved Response Length" below). */
             memcmp(pduPtr->data, gp->recvRec[i].apdu,
                    gp->recvRec[i].apduSize) != 0)
             /* *** END INFORMATIVE - APDU Contents and Duplicate Detection *** */
   {
      /* Something does not match. Must be a new message from
         the sender. Let us reuse this RR, even if it is a
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      request. If it is a request and response comes later,
      it won't match this record anyway due to reqid. */
   initRR = TRUE;
   /* If the old message was not deliverd, increment lost msg stat */
   if (gp->recvRec[i].transState != DELIVERED &&
       gp->recvRec[i].transState != DONE      &&
       gp->recvRec[i].transState != RESPONDED)
   {
      INCR_STATS(nmp->stats.lostMessages);
   }
}
else
{
   /* It is an existing RR with everything matching. */
   /* Nothing to do. We use the information in existing RR. */
   /* Set the alternate path bit based on new message so that
      acks, challenge etc will use the same path. Change only if
      it is not already using alternate path. */
   if (! gp->recvRec[i].altPath)
   {
      gp->recvRec[i].altPath = tsaReceiveParamPtr->altPath;
   }
   initRR = FALSE;
}

if (initRR)
{
   if (layerIn == TRANSPORT)
   {
      gp->recvRec[i].status          = TRANSPORT_RR;
   }
   else
   {
      gp->recvRec[i].status          = SESSION_RR;
   }
   gp->recvRec[i].srcAddr         = tsaReceiveParamPtr->srcAddr;
   gp->recvRec[i].transNum        = pduPtr->transNum;
   gp->recvRec[i].transState      = JUST_RECEIVED;
   gp->recvRec[i].priority        = tsaReceiveParamPtr->priority;
   gp->recvRec[i].altPath         = tsaReceiveParamPtr->altPath;
   gp->recvRec[i].auth            = FALSE;
   gp->recvRec[i].reqId           = 0; /* Init to invalid reqid. */
   if (layerIn == TRANSPORT)
   {
      gp->recvRec[i].serviceType     =
         pduPtr->pduMsgType == ACKD_MSG? ACKD: UNACK_RPT;
   }
   else
   {
      gp->recvRec[i].serviceType     = REQUEST;
      /* reqId can wrap around. Never use 0 as reqId so that
         a valid reqId will never match a receive record that
         does not correspond to corresponding request. */
      if (gp->reqId == 0)
      {
         gp->reqId++;
      }
      gp->recvRec[i].reqId           = gp->reqId++;
   }
   gp->recvRec[i].apduSize = tsaReceiveParamPtr->pduSize - 1;
   memcpy(gp->recvRec[i].apdu,
          pduPtr->data,
          gp->recvRec[i].apduSize); /* Store the APDU. */
   /* Compute the recvTimer value to be used. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      recvTimerValue = ComputeRecvTimerValue(
                         tsaReceiveParamPtr->srcAddr.addressMode,
                         tsaReceiveParamPtr->srcAddr.group);
      SetMsTimer(&gp->recvRec[i].recvTimer, recvTimerValue);
   }

   DeQueue(&gp->tsaInQ); /* Remove item from queue. */

   /* Now, we have a RR for this message. */
   /* Determine if the msg needs authentication and store. */
   /* Allow authentication for UnAck_Rpt. */
   /* flexDomain messages cannot be authenticated. Force it. */

   if (pduPtr->auth && tsaReceiveParamPtr->srcAddr.domainIndex != FLEX_DOMAIN)
   {
      gp->recvRec[i].needAuth = TRUE;
   }
   else
   {
      gp->recvRec[i].needAuth = FALSE;
   }

   /* Authentication is performed if it is a new message or
      in the process of authenticating and it needs to be
      autheneticated. */
   if ((gp->recvRec[i].transState == JUST_RECEIVED ||
        gp->recvRec[i].transState == AUTHENTICATING) &&
       gp->recvRec[i].needAuth)
   {
      /* The message needs authentication. Initiate Challenge. */
      /* Or ReInitiate Challenge as the prev one is probably lost. */
#ifdef DEBUG
      DebugMsg("ReceiveNewMsg: Initiating Challenge.");
#endif
      InitiateChallenge(i);
      return;
   }

   if (gp->recvRec[i].transState != DELIVERED &&
       gp->recvRec[i].transState != RESPONDED &&
       gp->recvRec[i].transState != DONE)
   {
      /* Deliver the message to the application layer. */
      Deliver(i);
   }
   else
   {
#ifdef DEBUG
      DebugMsg("ReceiveNewMsg: Msg received was already delivered.");
#endif
   }

   if (layerIn == TRANSPORT)
   {
      if (gp->recvRec[i].transState  == DELIVERED &&
          gp->recvRec[i].serviceType == ACKD)
      {
         /* Compose and send the acknowledgement. */
         TPSendAck(i);
      }
   }
   else if (gp->recvRec[i].transState == RESPONDED)
   {
      /* Must be a request that was already responded. */
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      /* We already have a resp. Simply send it. */
      SNSendResponse(i, FALSE); /* It can't be a null response. */
   }
   else if (gp->recvRec[i].transState == DONE)
   {
#ifdef DEBUG
      DebugMsg("ReceiveNewMsg: Nothing to do.");
#endif
   }
   return;
}

/*************************************************************
Function:  ReceiveRem
Returns:   None
Reference: None
Purpose:   To process REMINDER and REM/MSG messages.
           We either do nothing or send an ack (for Transport)
           or a response(for Session) if possible.
Comments:  None
*************************************************************/
static void ReceiveRem(Layer layerIn)
{
   TSAReceiveParam *tsaReceiveParamPtr;
   TSPDUPtr         pduPtr;
   int16            i;
   APDU            *apduPtr;
   uint16           apduSize;
   uint8            length;
   uint8            member;
   uint8            mlistIndex, mlistOffset;
   Byte            *mlistPtr;
   uint16           recvTimerValue;

   /* We have a REMINDER or REM/MSG in TSA input queue. */

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduPtr             = (TSPDUPtr) (tsaReceiveParamPtr + 1);
   length             = pduPtr->data[0];

   if (pduPtr->pduMsgType == REMINDER_MSG)
   {
      apduSize = 0;
      apduPtr  = NULL;
   }
   else
   {
      /* Must be REM_MSG_MSG. */
      /* TSPDU header size before APDU is actual 1 byte header,
         1 byte length field and the length of M_List field. */
      apduSize = tsaReceiveParamPtr->pduSize - 2 - length;
      apduPtr  = (APDU *)&pduPtr->data[1 + length];
   }

   /* First retrieve the associated RR, if it exists. */
   i = RetrieveRR(tsaReceiveParamPtr->srcAddr,
                  tsaReceiveParamPtr->priority);


   /* Check if the REMINDER msg has an associated RR. */
   if (i == -1 && pduPtr->pduMsgType == REMINDER_MSG)
   {
      /* No associated RR. Discard this REMINDER msg. */
#ifdef DEBUG
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      DebugMsg("ReceiveRem: REMINDER msg has no associated RR");
#endif
      DeQueue(&gp->tsaInQ);
      return;
   }

   if (i != -1 &&
       (gp->recvRec[i].srcAddr.addressMode != MULTICAST ||
        gp->recvRec[i].transNum != pduPtr->transNum ||
        gp->recvRec[i].status != (layerIn == TRANSPORT?TRANSPORT_RR:SESSION_RR) ||
        (apduPtr && apduSize != gp->recvRec[i].apduSize) ||
        (apduPtr &&
           /* *** START INFORMATIVE - APDU Contents and Duplicate Detection *** */
           /* See "START INFORMATIVE - APDU Contents and Duplicate Detection".  */
           memcmp(apduPtr, gp->recvRec[i].apdu, apduSize) != 0))
           /* *** END INFORMATIVE - APDU Contents and Duplicate Detection  *** */
      )

   {
      /* Associate RR does not match properly. Either it is not
         a multicast message or transNum does not match or
         APDU does not match. This is not OK for REMINDER_MSG.
         If it is however REM_MSG_MSG, we want to treat it as a
         new message. */
      if (pduPtr->pduMsgType == REMINDER_MSG)
      {
#ifdef DEBUG
         DebugMsg("ReceiveRem: Associated RR does not match.");
#endif
         DeQueue(&gp->tsaInQ);
         return;
      }
      if (gp->recvRec[i].transState == DELIVERED ||
          gp->recvRec[i].transState == DONE)
      {
         /* Reuse this receive record. We will free this record and allocate
            a new one. */
         gp->recvRec[i].status = UNUSED_RR;
         i = -1;
      }
      else
      {
         i = -1; /* Allocate new one. */
      }
   }

   if (i != -1 && gp->recvRec[i].serviceType == UNACK_RPT)
   {
      if (pduPtr->pduMsgType == REMINDER_MSG)
      {
         /* We should not have gotton this REMINDER as the original
            message was UNACK_RPT. So, ignore this message. */
#ifdef DEBUG
         DebugMsg("ReceiveRem: Original msg is UNACK_RPT.");
#endif
         DeQueue(&gp->tsaInQ);
         return;
      }
      /* Treat REM_MSG_MSG as new one. Fack this by setting i to -1. */
      i = -1;
   }

   if (i != -1)
   {
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      /* We have already received the corresponding APDU. */
      /* Probably the ack or response sent earlier was lost. */
      /* Sent the ack or response again. */
      if (!IsGroupMember(gp->recvRec[i].srcAddr.domainIndex,
                         gp->recvRec[i].srcAddr.group,
                         &member))
      {
         /* We are not member of the group used for this msg.
            Strange! Ignore this msg too!
            Network layer should not have delivered such msg
            to upper layers. */
#ifdef DEBUG
         DebugMsg("ReceiveRem: Strange! Not a group member!!.");
#endif
         DeQueue(&gp->tsaInQ);
         return;
      }
      mlistIndex  = member / 8;
      mlistOffset = member % 8;
      mlistPtr    = &pduPtr->data[1];
      if (
          length == 0
          ||
          mlistIndex >= length
          ||
          (mlistPtr[mlistIndex] & (1 << mlistOffset)) == 0
         )
      {
         /* We are asked to respond. Server did not get our ack or response. */

         /* If it is a REMINDER_MSG, it will be followed by the retry.
            So, we simply ignore it. */
         if (pduPtr->pduMsgType == REMINDER_MSG)
         {
            DeQueue(&gp->tsaInQ);
            return;
         }

         if (gp->recvRec[i].needAuth &&
               (gp->recvRec[i].transState == JUST_RECEIVED ||
                gp->recvRec[i].transState == AUTHENTICATING)
            )
         {
            /* Need authentication. Either a new msg or prev challenge
               was lost. */
#ifdef DEBUG
            DebugMsg("ReceiveRem: Initiating Challenge.");
#endif
            InitiateChallenge(i);
            DeQueue(&gp->tsaInQ);
            return;
         }
         if (gp->recvRec[i].transState != DELIVERED &&
             gp->recvRec[i].transState != RESPONDED &&
             gp->recvRec[i].transState != DONE)
         {
            /* Deliver the message to application layer. */
            Deliver(i);
         }

         /* Send ack or response. */
         if (gp->recvRec[i].transState  == DELIVERED ||
             gp->recvRec[i].transState  == RESPONDED)
         {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            DeQueue(&gp->tsaInQ);
            if (layerIn == TRANSPORT)
            {
               /* Compose and send the acknowledgement. */
               TPSendAck(i);
            }
            else if (gp->recvRec[i].transState == RESPONDED)
            {
               SNSendResponse(i, FALSE); /* can't be null resp. */
            }
            else
            {
               /* we are still waiting for a response. */
#ifdef DEBUG
               DebugMsg("ReceiveRem: Don't have response yet"
                        " or have null response.");
#endif
            }
         }
         else
         {
            /* We are still waiting for resp & so can't respond
               or in DONE state indicating server already received
               the ack or response. */
#ifdef DEBUG
            DebugMsg("ReceiveRem: Can't respond or no need to respond.");
#endif
            DeQueue(&gp->tsaInQ);
         }
      }
      else
      {
         /* We are not asked to acknowledege or respond. Ignore msg. */
         if (gp->recvRec[i].transState == DELIVERED ||
             gp->recvRec[i].transState == RESPONDED)
         {
            /* Note down the fact that the ack or resp has been received */
            gp->recvRec[i].transState = DONE;
         }
#ifdef DEBUG
         DebugMsg("ReceiveRem: We are not asked to ack or respond.");
#endif
         DeQueue(&gp->tsaInQ);
      }
      return;
   }

   /* Now we must have a REM_MSG_MSG with no associated RR. */
   /* REMINDER msg with no associated record was taken care of
      earlier. */
   i = AllocateRR();
   if (i == -1)
   {
      /* Unable to allocate a new RR. Give up. */
      return;
   }
   if (layerIn == TRANSPORT)
   {
      gp->recvRec[i].status          = TRANSPORT_RR;
   }
   else
   {
      gp->recvRec[i].status          = SESSION_RR;
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   gp->recvRec[i].srcAddr         = tsaReceiveParamPtr->srcAddr;
   gp->recvRec[i].transNum        = pduPtr->transNum;
   gp->recvRec[i].transState      = JUST_RECEIVED;
   gp->recvRec[i].priority        = tsaReceiveParamPtr->priority;
   gp->recvRec[i].altPath         = tsaReceiveParamPtr->altPath;
   gp->recvRec[i].auth            = FALSE; /* Not authenticated yet. */
   if (layerIn == TRANSPORT)
   {
      /* We have REM/MSG. So service type must be ACKD. */
      gp->recvRec[i].serviceType     = ACKD;
   }
   else
   {
      gp->recvRec[i].serviceType     = REQUEST;
      /* reqId can wrap around. Never use 0 as reqId so that
         a valid reqId will never match a receive record that
         does not correspond to corresponding request. */
      if (gp->reqId == 0)
      {
         gp->reqId++;
      }
      gp->recvRec[i].reqId           = gp->reqId++;
   }
   gp->recvRec[i].apduSize = apduSize;
   memcpy(gp->recvRec[i].apdu, apduPtr, apduSize);
   /* Compute the recvTimer value to be used. */
   recvTimerValue =
      ComputeRecvTimerValue(
                            tsaReceiveParamPtr->srcAddr.addressMode,
                            tsaReceiveParamPtr->srcAddr.group
                           );
   DeQueue(&gp->tsaInQ); /* Remove the item from queue. */

   SetMsTimer(&gp->recvRec[i].recvTimer, recvTimerValue);

   /* Flex domain messages cannot be authenticated. */
   if (pduPtr->auth &&
       tsaReceiveParamPtr->srcAddr.domainIndex != FLEX_DOMAIN)
   {
      gp->recvRec[i].needAuth = TRUE; /* Remember that we need auth. */
   }
   else
   {
      gp->recvRec[i].needAuth = FALSE;
   }

   /* Now, we have a RR for this message. */
   if (gp->recvRec[i].needAuth)
   {
      /* The message needs authentication. Initiate Challenge. */
#ifdef DEBUG
      DebugMsg("ReceiveRem: Initiating Challenge.");
#endif
      InitiateChallenge(i);
      return;
   }

   /* Deliver the message to the application layer. */
   Deliver(i);

   if (layerIn == TRANSPORT)
   {
      if (gp->recvRec[i].transState  == DELIVERED)
      {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
         /* Compose and send the acknowledgement. */
         TPSendAck(i);
      }
   }
   else
   {
      /* We just delivered. Can't have a response yet. */
   }
   return;
}

/*************************************************************
Function:  TPSendAck
Returns:   None
Reference: None
Purpose:   To Send an ACK message for ACKD msg received by the
           transport layer. If there is no space in network
           output queue, then nothing is sent.
Comments:  None.
*************************************************************/
static void TPSendAck(uint16 rrIndexIn)
{
   NWSendParam     *nwSendParamPtr;   /* Ptr to NW Send Param. */
   TSPDUPtr          pduPtr;          /* Ptr to TPDU sent.     */
   Queue           *nwQueuePtr;
   DestinationAddress    destAddr;

   if (gp->recvRec[rrIndexIn].priority)
   {
      nwQueuePtr = &gp->nwOutPriQ;
   }
   else
   {
      nwQueuePtr = &gp->nwOutQ;
   }

   if (QueueFull(nwQueuePtr))
   {
      return; /* Can't send the acknowledgement now. */
   }

   nwSendParamPtr = QueueTail(nwQueuePtr);
   pduPtr        = (TSPDUPtr) (nwSendParamPtr + 1);

   /* Form the TPDU directly in the network layer's queue. */
   pduPtr->auth     = FALSE; /* Acks are not authenticated. */
   pduPtr->pduMsgType  = ACK_MSG;
   pduPtr->transNum = gp->recvRec[rrIndexIn].transNum;

   /* Fill in the details for the network layer. */

   /* First, form sender's (i.e our) address. */
   destAddr.domainIndex = gp->recvRec[rrIndexIn].srcAddr.domainIndex;
   if (destAddr.domainIndex == FLEX_DOMAIN)
   {
      /* Flex domain was used. Copy flex domain information. */
      destAddr.flexDomainLen =
         gp->recvRec[rrIndexIn].srcAddr.flexDomainLen;
      /* Make sure we have a good length field. */
      if (destAddr.flexDomainLen <= DOMAIN_ID_LEN)
      {
         memcpy(destAddr.flexDomainId,
                gp->recvRec[rrIndexIn].srcAddr.flexDomainId,
                destAddr.flexDomainLen);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      }
   }

   /* We send unicast ACK or multicast ACK depending on msg recvd. */
   if (gp->recvRec[rrIndexIn].srcAddr.addressMode == MULTICAST)
   {
      /* Send ACK in address format 2b. */
      destAddr.addressMode = MULTICAST_ACK;
      destAddr.addr.addr2b.subnetAddr =
         gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
      destAddr.addr.addr2b.groupAddr.group  =
         gp->recvRec[rrIndexIn].srcAddr.group;
      if (!IsGroupMember(destAddr.domainIndex,
destAddr.addr.addr2b.groupAddr.group,
                         &destAddr.addr.addr2b.groupAddr.member))
      {
           ErrorMsg("SendACKTPDU: Ack msg for a non-existing group.");
           return;
      }
   }
   else
   {
      /* Send Ack in address format 2a. */
      destAddr.addressMode = SUBNET_NODE;
      destAddr.addr.addr2a =
         gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
   }

   nwSendParamPtr->dropIfUnconfigured = FALSE; /* acks are not dropped */
   nwSendParamPtr->destAddr = destAddr;
   nwSendParamPtr->pduType  = TPDU_TYPE;
   nwSendParamPtr->deltaBL  = 0;
   nwSendParamPtr->altPath  = gp->recvRec[rrIndexIn].altPath;
   nwSendParamPtr->pduSize  = 1;

#ifdef DEBUG
   DebugMsg("Debug: TPSendAck. Sending an ACK.");
#endif
   EnQueue(nwQueuePtr);
}

/*************************************************************
Function:  SNSendResponse
Returns:   None
Reference: None
Purpose:   To send a response message for REQUEST msg received by the
           session layer. If there is no space in network
           output queue, then nothing is sent.
           If it is a null response, then the receive record is freed
           and no response goes out.
Comments:  This function is called only when the application layer
           has already given the response. (Or else how can this
           function know about how to respond anyway?)
*************************************************************/
static void SNSendResponse(uint16 rrIndexIn, Boolean nullResponse)
{
   NWSendParam       *nwSendParamPtr;   /* Ptr to NW Send Param. */
   TSPDUPtr           pduPtr;          /* Ptr to TPDU sent.     */
   Queue             *nwQueuePtr;
   DestinationAddress destAddr;

   if (gp->recvRec[rrIndexIn].priority)
   {
      nwQueuePtr = &gp->nwOutPriQ;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }
   else
   {
      nwQueuePtr = &gp->nwOutQ;
   }

   if (QueueFull(nwQueuePtr) && !nullResponse)
   {
      return; /* Can't sent the response now. */
   }

   if (gp->recvRec[rrIndexIn].transState != RESPONDED &&
       gp->recvRec[rrIndexIn].transState != DONE)
   {
      ErrorMsg("SNSendResponse: How can I be called without "
               " atleast one resp?");
      return;
   }

   if (nullResponse)
   {
      /* Set state to DONE and do not send the response. */
#ifdef DEBUG
      DebugMsg("SendResponse: Null response. Nothing goes out.");
#endif
      gp->recvRec[rrIndexIn].transState = DONE;
      return;
   }

   nwSendParamPtr = QueueTail(nwQueuePtr);
   pduPtr         = (TSPDUPtr) (nwSendParamPtr + 1);

   if (gp->nwOutBufSize < gp->recvRec[rrIndexIn].rspSize + 1)
   {
      ErrorMsg("SNSendResponse: Network buf too small for response.");
      return;
   }
   pduPtr->auth     = FALSE; /* Responses are not authenticated. */
   pduPtr->pduMsgType  = RESPONSE_MSG;
   pduPtr->transNum = gp->recvRec[rrIndexIn].transNum;
   /* Copy the existing response from RR */
   if (gp->recvRec[rrIndexIn].rspSize <=
         DecodeBufferSize((uint8) eep->readOnlyData.appOutBufSize))
   {
   /* *** START INFORMATIVE - Saved Response Length *** */
/* This implementation saves the entire response in the receive transaction record.
 * Thus, if and when a retry of the request is received, the response can be
 * re-transmitted without re-delivering the request to the application for
 * construction of a response.  It is acceptable to save only responses of certain
 * length (e.g., as little as 1 byte) in order to minimize RAM use.
 * Under these conditions, a retry of a request that illicited a response of
 * length greater than that minimium length is considered to be idempotent (can be
 * safely executed again) and may be re-delivered to the application for
 * reformulation of the response.  It is also required that under these conditions
 * that the application be informed of the duplicate nature of the request in case
 * the application chooses to treat the request as non-idempotent by saving the
 * response for retransmission without recomputation.  In this case a key must
 * also be provided for the application to use to map the request to a saved
 * response. */
      memcpy(pduPtr->data, gp->recvRec[rrIndexIn].response,
             gp->recvRec[rrIndexIn].rspSize);
   /* *** END INFORMATIVE - Saved Response Length *** */
   }
   else
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   {
#ifdef DEBUG
      DebugMsg("SNSendResponse: Discarding a long response.");
#endif
      return;
   }
   /* Fill in the details for the network layer. */

   destAddr.domainIndex = gp->recvRec[rrIndexIn].srcAddr.domainIndex;
   if (destAddr.domainIndex == FLEX_DOMAIN)
   {
      /* Flex domain was used. Copy flex domain information. */
      destAddr.flexDomainLen =
         gp->recvRec[rrIndexIn].srcAddr.flexDomainLen;
      if (destAddr.flexDomainLen <= DOMAIN_ID_LEN)
      {
         memcpy(destAddr.flexDomainId,
                gp->recvRec[rrIndexIn].srcAddr.flexDomainId,
                destAddr.flexDomainLen);
      }
   }

   /* We send unicast response or multicast response depending on
      the request received. */
   if (gp->recvRec[rrIndexIn].srcAddr.addressMode == MULTICAST)
   {
      /* Send the response in address format 2b. */
      destAddr.addressMode = MULTICAST_ACK;
      destAddr.addr.addr2b.subnetAddr =
         gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
      destAddr.addr.addr2b.groupAddr.group  =
         gp->recvRec[rrIndexIn].srcAddr.group;
      if (!IsGroupMember(destAddr.domainIndex,
destAddr.addr.addr2b.groupAddr.group,
                         &destAddr.addr.addr2b.groupAddr.member))
      {
         ErrorMsg("SNSendResponse: Response for a non-existing group.");
         return;
      }
   }
   else
   {
      /* Send response in address format 2a. */
      destAddr.addressMode = SUBNET_NODE;
      destAddr.addr.addr2a =
         gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
   }

   nwSendParamPtr->dropIfUnconfigured = FALSE; /* responses are not dropped */
   nwSendParamPtr->destAddr = destAddr;
   nwSendParamPtr->pduType  = SPDU_TYPE;
   nwSendParamPtr->deltaBL  = 0;
   nwSendParamPtr->altPath  = gp->recvRec[rrIndexIn].altPath;
   nwSendParamPtr->pduSize  = gp->recvRec[rrIndexIn].rspSize + 1;

#ifdef DEBUG
   DebugMsg("SNSendResponse: Sending a response.");
#endif
   EnQueue(nwQueuePtr);
}

/*************************************************************
Function:  Deliver
Returns:   None
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Reference: None
Purpose:   To deliver a message received by tranport or session layer
           to the application layer.
Comments:  layerIn is not needed as it is not used.
*************************************************************/
static void Deliver(uint16 rrIndexIn)
{
   APPReceiveParam    *appReceiveParamPtr;
   char               *apduInPtr;
   uint16              i = rrIndexIn; /* Use i instead of rrIndexIn. */

   if (gp->recvRec[i].needAuth && gp->recvRec[i].transState != AUTHENTICATED)
   {
      /* The message needs authentication but was not authenticated. */
      gp->recvRec[i].transState = DONE;
      return;
   }

   if (QueueFull(&gp->appInQ))
   {
      /* We can wait, but then we may not be able to guarantee delivery in
         sequence from a given source node. So, it is better to drop the
         message and let the retry mechanism take care of redelivery. */
      INCR_STATS(nmp->stats.lostMessages);
      gp->recvRec[i].transState = DONE; /* indicate that this one is done */
      return;
   }

   appReceiveParamPtr = QueueTail(&gp->appInQ);
   apduInPtr          = (char *)(appReceiveParamPtr + 1);

   appReceiveParamPtr->indication = MESSAGE;
   appReceiveParamPtr->srcAddr    = gp->recvRec[i].srcAddr;
   appReceiveParamPtr->service    = gp->recvRec[i].serviceType;
   appReceiveParamPtr->priority   = gp->recvRec[i].priority;
   appReceiveParamPtr->altPath    = gp->recvRec[i].altPath;
   appReceiveParamPtr->auth       = gp->recvRec[i].auth;
   appReceiveParamPtr->pduSize    = gp->recvRec[i].apduSize;
   appReceiveParamPtr->reqId      = gp->recvRec[i].reqId;

   if (gp->appInBufSize < gp->recvRec[i].apduSize)
   {
      ErrorMsg("Deliver: APDU size too big");
      /* We can never deliver this APDU. So, make it look like
         it was delivered so that it can be discarded. We don't
         want to send any ACK or response for this message. */
      gp->recvRec[i].transState      = DONE;
      nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
      return;
   }
   /* Now it should be safe to do memcpy. */
   memcpy(apduInPtr, gp->recvRec[i].apdu, gp->recvRec[i].apduSize);
   EnQueue(&gp->appInQ);
   INCR_STATS(nmp->stats.layer6_7MsgsRcvd);
   gp->recvRec[i].transState      = DELIVERED;
#ifdef DEBUG
   DebugMsg("Debug: Deliver: Packet has been delivered"
            " to the application layer.");
#endif
}

/*************************************************************
Function:  RetrieveRR
Returns:   Index of RR Table that matches given input parameters.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
           -1 if none exists.
Reference: None
Purpose:   To retrieve the receive record from the RR table
           that corresponds to a message received. For matching,
           we use priority, domainIndex, addressMode, and source address.
           We also match subnet if the message is broadcast
           or group if the message is multicast.
Comments:  None
************************************************************/
static int16 RetrieveRR(SourceAddress srcAddrIn,
                        Boolean priorityIn)
{
   int16 i;

   /* Search through all the receive records for a match. */

   for (i = 0; i < gp->recvRecCnt; i++)
   {
      if (
           priorityIn == gp->recvRec[i].priority
           &&
           /* Destination subnet/node match is based on domainIndex. */
           srcAddrIn.domainIndex == gp->recvRec[i].srcAddr.domainIndex
           &&
           (srcAddrIn.addressMode == gp->recvRec[i].srcAddr.addressMode)
           &&
           /* Source node address should always match. */
           (memcmp(&srcAddrIn.subnetAddr,
                   &gp->recvRec[i].srcAddr.subnetAddr,
                   sizeof(SubnetAddress)) == 0)
           &&
           /* Make sure BROADCAST address matches for broadcast messages. */
           ( srcAddrIn.addressMode != BROADCAST ||
             srcAddrIn.broadcastSubnet == gp->recvRec[i].srcAddr.broadcastSubnet )
           &&
           /* Make sure MULTICAST address matches for multicast messages. */
            ( srcAddrIn.addressMode != MULTICAST ||
              srcAddrIn.group == gp->recvRec[i].srcAddr.group )
        )

      {
         break;
      }
   }

   if (i == gp->recvRecCnt)
   {
      return(-1); /* Matching RR was not found. */
   }

   return(i); /* Found matching RR. */
}


/************************************************************
Function:  AllocateRR
Returns:   Index of RR table that can be used for a new msg.
Reference: None
Purpose:   To find an index in the RR table that is UNUSED.
Comments:  None
************************************************************/
static int16 AllocateRR(void)
{
   int16 i;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* First search for one that is unused. */
   for (i = 0; i < gp->recvRecCnt; i++)
   {
      if (gp->recvRec[i].status == UNUSED_RR)
      {
         return(i); /* Found one that is unused. */
      }
   }

   return(-1);
}

/*****************************************************************
Function:  ComputeRecvTimerValue
Returns:   Receive Timer value in milli seconds.
Reference: None
Purpose:   To compute the value to be used for receive timer
           based on the address format of the received message
           and group id.
Comments:  None
*****************************************************************/
static uint16 ComputeRecvTimerValue(AddrMode      addrModeIn,
                                    MulticastAddress groupIdIn)
{
   uint16 i, max = 0, temp;

   if (addrModeIn == UNIQUE_NODE_ID)
   {
      return((uint16)(NGTIMER_SPCL_VAL * 1000.0));
   }
   if (addrModeIn == MULTICAST)
   {
      /* Search through the address table to find the receiver timer val. */
      /* If there is more than one entry with the same group,
         use the one with the max rcv timer value */
      for (i = 0; i < NUM_ADDR_TBL_ENTRIES; i++)
      {
         if (eep->addrTable[i].addrFormat >= 128 &&
             eep->addrTable[i].groupEntry.groupID == groupIdIn)
         {
            /* Group format match */
            temp = DecodeRcvTimer((uint8)
                        eep->addrTable[i].groupEntry.rcvTimer);
            /* *** START INFORMATIVE - Multicast Receive Timer *** */
            /* Using the maximum receive timer for the group is not required.  It
             * is acceptable to use the receive timer for the first group entry
             * found in the table. */
            if (temp > max)
            {
               max = temp;
            }
            /* *** END INFORMATIVE - Multicast Receive Timer *** */
         }
      }
      return(max);
   }
   /* All other messages use non-group timer value. */
   return(DecodeRcvTimer((uint8) eep->configData.nonGroupTimer));
}

/*****************************************************************
Function:  SNSend
Returns:   None
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Reference:CNP, part 1, sub-clause 10.7.
Purpose:   To implement Send algorithm for the session layer.
           If there is anything to be sent by the session layer,
           this function will process that message and sends it.
           If there is any ongoing message, it might need some
           processing such as retry, timer expiry etc. This
           function will process such events too.
Comments:  Update the priority transmit timer, if it exists.
           Update the non-priority transmit timer, if it exists.
           If the priority transmit timer expired then
              process this event.
           else if there is priority message to be sent and there is space
                in priority queue of the network layer then
              process the priority message.
           else if the non-priority transmit timer expired then
              process that event.
           else if there is non-priority message to be sent and there is space
                in non-priority queue of network layer then
              process the non-priority message.
           else
                nothing to do. return.
Note:
*****************************************************************/
void SNSend(void)
{
   TSASendParam    *tsaSendParamPtr;
   uint16           i;

   /* Delay SNSend after power-up or external reset. */
   if ( gp->tsDelayTimer.curTimerValue > 0 &&
        (nmp->resetCause == POWER_UP_RESET ||
         nmp->resetCause == EXTERNAL_RESET)
      )
   {
      UpdateMsTimer(&gp->tsDelayTimer);
      return; /* Do nothing */
   }

   /************************************************
     Send response, if any, first. Responses are not like transactions
     and hence it is better to send it first.
   ************************************************/
   if (!QueueEmpty(&gp->tsaRespQ) &&
            !QueueFull(&gp->nwOutQ))
   {
      /* We have a response to be sent out.
         Make sure the response is not stale.
         Response should have a reqId.
         If there is no RR for this reqId, then it is stale.
         Throw the response away, if it is stale. */
      tsaSendParamPtr = QueueHead(&gp->tsaRespQ);

      if (tsaSendParamPtr->service != RESPONSE)
      {
         /* Throw away this message. Only responses are
            allowed in this queue. */
#ifdef DEBUG
         DebugMsg("SNSend: Response queue is only for responses.");
#endif
         DeQueue(&gp->tsaRespQ);
         return;
      }

      /* Search for the associated RR for this response. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      for (i = 0; i < gp->recvRecCnt; i++)
      {
         if (gp->recvRec[i].status == SESSION_RR &&
             gp->recvRec[i].reqId == tsaSendParamPtr->reqId)
         {
            break;
         }
      }
      if (i == gp->recvRecCnt ||
          gp->recvRec[i].serviceType != REQUEST ||
          gp->recvRec[i].transState != DELIVERED)
      {
         /* Stale or duplicate response. Ignore it. */
         DeQueue(&gp->tsaRespQ);
#ifdef DEBUG
         DebugMsg("SNSend: Discarding stale or duplicate response.");
#endif
         INCR_STATS(nmp->stats.lateResponses);
         return;
      }
      /* Copy the response to the receive record and send response. */
      gp->recvRec[i].rspSize = tsaSendParamPtr->apduSize;
      /* Resp should fit in response field as it was found to be
         fit in the TSA queue. */
      memcpy(gp->recvRec[i].response,
             (char *) (tsaSendParamPtr + 1),
             gp->recvRec[i].rspSize);
      gp->recvRec[i].transState = RESPONDED;
      SNSendResponse(i, tsaSendParamPtr->nullResponse);
      INCR_STATS(nmp->stats.layer6_7RespSent);
      DeQueue(&gp->tsaRespQ);
      return;
   }

   /* Update transmit timers, if they do exist */
   if (gp->priXmitRec.status == SESSION_TX &&
       gp->priXmitRec.xmitTimer.curTimerValue > 0)
   {
      UpdateMsTimer(&gp->priXmitRec.xmitTimer);
   }
   if (gp->xmitRec.status == SESSION_TX &&
       gp->xmitRec.xmitTimer.curTimerValue > 0)
   {
      UpdateMsTimer(&gp->xmitRec.xmitTimer);
   }

   /************************************************
     Priority transmit timer expired event.
    ************************************************/
   if (gp->priXmitRec.status == SESSION_TX &&
       gp->priXmitRec.xmitTimer.curTimerValue == 0)
   {
      XmitTimerExpiration(SESSION, TRUE);
      return;
   }
   /************************************************
     Send a new priority message event.
    ************************************************/
   else if (gp->priXmitRec.status == UNUSED_TX &&
            ! QueueEmpty(&gp->tsaOutPriQ)      &&
            ! QueueFull(&gp->nwOutPriQ) )
   {
      SendNewMsg(SESSION, TRUE);
      return;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }
   /*************************************************
     Non-priority timer expired event.
     ************************************************/
   else if (gp->xmitRec.status == SESSION_TX &&
            gp->xmitRec.xmitTimer.curTimerValue == 0)
   {
      XmitTimerExpiration(SESSION, FALSE);
   }
   /*************************************************
     Send a new non-priority message.
    ************************************************/
   else if (gp->xmitRec.status == UNUSED_TX &&
            ! QueueEmpty(&gp->tsaOutQ)      &&
            ! QueueFull(&gp->nwOutQ) )
   {
      SendNewMsg(SESSION, FALSE);
   }
   else
   {
      /* Either there is no work or there is no space. */
      return;
   }

   return;

}

/*****************************************************************
Function:  SNReceive
Returns:   None
Reference: None
Purpose:   To receive and process incoming SPDU's.
Comments:  None
*****************************************************************/
void SNReceive(void)
{
   TSAReceiveParam *tsaReceiveParamPtr;/* Param in tsa input queue.  */
   TSPDUPtr         spduInPtr;         /* Pointer to SPDU received. */
   int16            i;

   /* Update Receive Timers, if they do exist */
   for (i = 0; i < gp->recvRecCnt; i++)
   {
      if (gp->recvRec[i].status == SESSION_RR)
      {
         if (gp->recvRec[i].recvTimer.curTimerValue > 0)
         {
            UpdateMsTimer(&gp->recvRec[i].recvTimer);
         }
         if (gp->recvRec[i].recvTimer.curTimerValue == 0)
         {
            /* Timer expired. Release the receive record. */
#ifdef DEBUG
            DebugMsg("SNReceive: Receive timer expired.");
#endif
            /* Deliver the request if it has not been delivered.
               Deliver fn checks for authentication. */
            if (gp->recvRec[i].transState == DELIVERED ||
                gp->recvRec[i].transState == DONE ||
                gp->recvRec[i].transState == RESPONDED)
            {
               gp->recvRec[i].status = UNUSED_RR; /* Release the RR */
            }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            else
            {
               Deliver(i);
            }
         }
         else if ((gp->recvRec[i].transState == JUST_RECEIVED &&
                   !gp->recvRec[i].needAuth) ||
                   gp->recvRec[i].transState == AUTHENTICATED )
         {
            /* The request does not need authentication and was just received
               or it has been authenticated. */
            Deliver(i);
         }
      }
   }

   /* Check if there is SPDU to be processed. */
   if (QueueEmpty(&gp->tsaInQ))
   {
      /* There is nothing to process. */
      return;
   }

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   spduInPtr          = (TSPDUPtr) ((char *)tsaReceiveParamPtr
                                    + sizeof(TSAReceiveParam));

   /* Check if the PDU is for session layer. If not, we are done. */
   if (tsaReceiveParamPtr->pduType != SPDU_TYPE)
   {
      return;
   }

   /* We have a SPDU to be processed. Check the type and
      process accordingly. */
   switch (spduInPtr->pduMsgType)
   {
      case RESPONSE_MSG:
         SNReceiveResponse();
         return;
      case REQUEST_MSG:
         ReceiveNewMsg(SESSION);
         return;
      case REMINDER_MSG:
         /* Fall through. */
      case REM_MSG_MSG:
         ReceiveRem(SESSION);
         return;
      default:
         nmp->errorLog = UNKNOWN_PDU;
         ErrorMsg("SNReceive: Unknown SPDU type received.");
         DeQueue(&gp->tsaInQ);
   }

   return;
}

/*****************************************************************
Function:  AuthSend
Returns:   None
Reference: None
Purpose:   To send challenges that are pending in receive records.
           These are challenges that could not be sent earlier due
           to unavailable space in network queue.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Comments:  None
*********************************************************************/
void AuthSend(void)
{
   int16 i;

   /* Only thing the authentication layer can do here is to check if any
      challenges need to be sent. */
   for (i = 0; i < gp->recvRecCnt; i++)
   {
      if (gp->recvRec[i].status != UNUSED_RR &&
          gp->recvRec[i].needAuth &&
          gp->recvRec[i].transState == JUST_RECEIVED)
      {
         InitiateChallenge(i);
      }
   }

   return;
}

/*********************************************************************
Function:  AuthReceive
Returns:   None
Reference: None
Purpose:   To receive an incoming AUTH_PDU packet and process.
Comments:  None
*********************************************************************/

void AuthReceive(void)
{
   TSAReceiveParam *tsaReceiveParamPtr; /* Parameter in tsa input queue. */
   AuthPDUPtr       pduInPtr;           /* Ptr to PDU received. */

   /* Check if there is AuthPDU to be processed. */
   if (QueueEmpty(&gp->tsaInQ))
   {
      /* There is nothing to process. */
      return;
   }

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduInPtr           = (AuthPDUPtr) (tsaReceiveParamPtr + 1);

   /* Check if the PDU is for session layer. If not, we are done. */
   if (tsaReceiveParamPtr->pduType != AUTHPDU_TYPE)
   {
      return;
   }

   /* We have a AuthPDU to be processed. Check the type and
      process accordingly. */
   if (pduInPtr->pduMsgType == CHALLENGE_MSG)
   {
      SendReplyToChallenge();
   }
   else if (pduInPtr->pduMsgType == REPLY_MSG)
   {
      ProcessReply();
   }
   else
   {
      ErrorMsg("AuthReceive: Unknown pdu type received.");
      nmp->errorLog = UNKNOWN_PDU;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      DeQueue(&gp->tsaInQ);
      return;
   }

}

/*****************************************************************
Function:  InitiateChallenge
Returns:   None
Reference: None
Purpose:   To send a challenge message for a message received.
Comments:  None
*****************************************************************/
static void InitiateChallenge(uint16 rrIndexIn)
{
   AuthPDUPtr       pduOutPtr;          /* Ptr to PDU sent. */
   NWSendParam     *nwSendParamPtr;
   Queue           *nwQueuePtr;
   Byte             randomValue[8];
   uint8            i;

   /* Since authentication is not allowed in flex domain, if
      this fn is called when a message received is in flex domain,
      let us force the authentication to be a failure */
   if (gp->recvRec[rrIndexIn].srcAddr.domainIndex == FLEX_DOMAIN)
   {
      gp->recvRec[rrIndexIn].auth = FALSE;
      gp->recvRec[rrIndexIn].transState = AUTHENTICATED;
      return;
   }

   if (gp->recvRec[rrIndexIn].priority)
   {
      nwQueuePtr = &gp->nwOutPriQ;
   }
   else
   {
      nwQueuePtr = &gp->nwOutQ;
   }

   if (QueueFull(nwQueuePtr))
   {
      /* No space to send challenge anyway. Come back later. */
      return;
   }

   nwSendParamPtr = QueueTail(nwQueuePtr);
   pduOutPtr      = (AuthPDU *)(nwSendParamPtr + 1);

   /* First compute the random bytes to be sent */
   if (gp->recvRec[rrIndexIn].transState != AUTHENTICATING)
   {
      /* Generate random number only the first time we are called
         for this message. subsequent calls use the same rand. */
      for (i = 0; i < 8; i++)
      {
        randomValue[i] = (Byte)((gp->prevChallenge[i] + rand() % 256 +
                         GetCurrentTime() % 256) % 256);
        gp->prevChallenge[i] = randomValue[i];
      }
      memcpy(gp->recvRec[rrIndexIn].rand, randomValue, 8); /* Save */
   }

   /* Form the challenge AuthPDU. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   pduOutPtr->fmt = addrModeToFmt[gp->recvRec[rrIndexIn].srcAddr.addressMode];
   if (gp->recvRec[rrIndexIn].srcAddr.addressMode == MULTICAST)
   {
      /* For Multicast message, send the group info with AuthPDU. */
      nwSendParamPtr->pduSize = 10;
      pduOutPtr->group    = gp->recvRec[rrIndexIn].srcAddr.group;
   }
   else
   {
      nwSendParamPtr->pduSize = 9;
   }
   pduOutPtr->pduMsgType  = CHALLENGE_MSG;
   pduOutPtr->transNum = gp->recvRec[rrIndexIn].transNum;
   memcpy(pduOutPtr->value.randomBytes,
          gp->recvRec[rrIndexIn].rand,
          8);

   /* Now fill in the NWSendParam structure. */
   nwSendParamPtr->dropIfUnconfigured = FALSE; /* Challenges are not dropped */
   /* Challenge must have come from a particular node.
      Use format 2b for multicast and format 2a for others. */
   nwSendParamPtr->destAddr.domainIndex =
      gp->recvRec[rrIndexIn].srcAddr.domainIndex;
   if (gp->recvRec[rrIndexIn].srcAddr.addressMode == MULTICAST)
   {
      nwSendParamPtr->destAddr.addressMode = MULTICAST_ACK;
      nwSendParamPtr->destAddr.addr.addr2b.subnetAddr =
         gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
      nwSendParamPtr->destAddr.addr.addr2b.groupAddr.group =
         gp->recvRec[rrIndexIn].srcAddr.group;
      if (!IsGroupMember(gp->recvRec[rrIndexIn].srcAddr.domainIndex,
                         gp->recvRec[rrIndexIn].srcAddr.group,
                         &nwSendParamPtr->destAddr.addr.addr2b.groupAddr.member))
      {
         ErrorMsg("InitiateChallenge: Strange: We are not member of group???.");
         return; /* Don't challenge. This case should not happen. */
      }
   }
   else
   {
      nwSendParamPtr->destAddr.addressMode = SUBNET_NODE;
      nwSendParamPtr->destAddr.addr.addr2a =
         gp->recvRec[rrIndexIn].srcAddr.subnetAddr;
   }

   nwSendParamPtr->pduType = AUTHPDU_TYPE;
   nwSendParamPtr->deltaBL = 0;
   nwSendParamPtr->altPath = gp->recvRec[rrIndexIn].altPath;
   gp->recvRec[rrIndexIn].transState = AUTHENTICATING;
   EnQueue(nwQueuePtr);
#ifdef DEBUG
   DebugMsg("InitiateChallenge: Sending a challenge.");
#endif
   return;
}

/*************************************************************
Function:  SendReplyToChallenge
Returns:   None
Reference: None
Purpose:   To send a reply to a challenge just received.
Comments:  None
*************************************************************/
static void SendReplyToChallenge(void)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
{
   TSAReceiveParam *tsaReceiveParamPtr; /* Parameter in tsa input queue. */
   AuthPDUPtr       pduInPtr;           /* Ptr to PDU received. */
   AuthPDUPtr       pduOutPtr;          /* Ptr to PDU sent.     */
   NWSendParam     *nwSendParamPtr;
   Queue           *nwQueuePtr;
   TransmitRecord  *xmitRecPtr;
   Byte             encryptValue[8];



   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduInPtr           = (AuthPDUPtr) (tsaReceiveParamPtr + 1);

   if (tsaReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN)
   {
      /* Challenge was received in flex domain that is not
         possible. We don't initiate challenge in flex domain.
         Ignore the challenge. */
      DeQueue(&gp->tsaInQ);
      return;
   }

   if (tsaReceiveParamPtr->priority)
   {
      nwQueuePtr = &gp->nwOutPriQ;
      xmitRecPtr = &gp->priXmitRec;
   }
   else
   {
      nwQueuePtr = &gp->nwOutQ;
      xmitRecPtr = &gp->xmitRec;
   }


   /* Make sure that this challenge for current transaction
      in progress. If not, it is stale. Ignore it. */
   /* Also do not reply if we did not set auth bit or
      the group value in challenge msg, if present,
      does not match the one in transmit record. */
   if (
       xmitRecPtr->status == UNUSED_TX ||
       ! xmitRecPtr->auth ||
       pduInPtr->transNum != xmitRecPtr->transNum ||
       addrFmtToMode[pduInPtr->fmt] != xmitRecPtr->nwDestAddr.addressMode ||
       (pduInPtr->fmt == 1 &&
         xmitRecPtr->nwDestAddr.addr.addr1 != pduInPtr->group)
      )
   {
      DeQueue(&gp->tsaInQ);
      INCR_STATS(nmp->stats.lateChallenges);
      return;
   }

   if (QueueFull(nwQueuePtr))
   {
      /* No Space to send reply anyway. Come back later. */
      return;
   }
   nwSendParamPtr = QueueTail(nwQueuePtr);
   pduOutPtr      = (AuthPDU *)(nwSendParamPtr + 1);

   /* First compute the cryptoBytes to be sent. */
   Encrypt(pduInPtr->value.randomBytes,
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            xmitRecPtr->apdu, xmitRecPtr->apduSize,
            tsaReceiveParamPtr->srcAddr.domainIndex,
            encryptValue);

   /* Form the reply AuthPDU. */
   pduOutPtr->fmt = pduInPtr->fmt;
   if (pduInPtr->fmt == 1)
   {
      pduOutPtr->group = pduInPtr->group;
      nwSendParamPtr->pduSize = 10;
   }
   else
   {
      nwSendParamPtr->pduSize = 9;
   }
   pduOutPtr->pduMsgType = REPLY_MSG;
   pduOutPtr->transNum = xmitRecPtr->transNum;
   memcpy(pduOutPtr->value.cryptoBytes, encryptValue, 8);

   /* Now fill in the NWSendParam structure */
   nwSendParamPtr->dropIfUnconfigured = FALSE; /* Replies are not dropped */
   /* Challenge must have come from a particular node.
      Use subnet addressing to send to that node. */
   nwSendParamPtr->destAddr.domainIndex =
      tsaReceiveParamPtr->srcAddr.domainIndex;
   nwSendParamPtr->destAddr.addressMode = SUBNET_NODE;
   nwSendParamPtr->destAddr.addr.addr2a =
      tsaReceiveParamPtr->srcAddr.subnetAddr;
   nwSendParamPtr->pduType = AUTHPDU_TYPE;
   nwSendParamPtr->deltaBL = 0;
   nwSendParamPtr->altPath = tsaReceiveParamPtr->altPath;
   EnQueue(nwQueuePtr);
   DeQueue(&gp->tsaInQ);
#ifdef DEBUG
   DebugMsg("SendReply: Sending a reply msg.");
#endif
   /* Restart the transmit timer. */
   SetMsTimer(&xmitRecPtr->xmitTimer, xmitRecPtr->xmitTimerValue);
   return;
}

/*************************************************************
Function:  ProcessReply
Returns:   None
Reference: None
Purpose:   To process a reply msg received to see if it meets the
           challenge sent earlier.
Comments:  None
*************************************************************/
static void ProcessReply(void)
{
   TSAReceiveParam *tsaReceiveParamPtr;  /* Parameter in tsa input queue. */
   AuthPDUPtr       pduInPtr;            /* Ptr to PDU received. */
   int16            i;
   Byte             encryptValue[8];
   Byte             domainIndex;

   tsaReceiveParamPtr = QueueHead(&gp->tsaInQ);
   pduInPtr           = (AuthPDUPtr) (tsaReceiveParamPtr + 1);

   if (tsaReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN)
   {
      /* Reply received in flex domain. Ignore it. */
      DeQueue(&gp->tsaInQ);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      return;
   }

   /* If the reply indicates that the original address format is
      multicast, then make sure that the receive record matches
      this group number. Force it so that RetriveRR will do the
      match. */
   tsaReceiveParamPtr->srcAddr.group = pduInPtr->group;

   /* The address format of the reply will be different from the address
      format of the original message. The original address format is in
      the Reply PDU. Change the addressmode in source address before
      calling RetrieveRR. */
   tsaReceiveParamPtr->srcAddr.addressMode = addrFmtToMode[pduInPtr->fmt];

   /* Retrieve the associated record.

      If the original address format is broadcast, we don't have
      the subnet number in the current reply(as the reply itself does
      not use broadcast address mode) and hence we can't match
      subnet number. So, we don't support authentication in
      broadcast mode. */

   i = RetrieveRR(tsaReceiveParamPtr->srcAddr,
                  tsaReceiveParamPtr->priority);

   if (i == -1)
   {
      /* We did not find an associated RR. Ignore reply. */
      DeQueue(&gp->tsaInQ);
      INCR_STATS(nmp->stats.lateReplies);
      return;
   }

   /* We have a RR. */

   /* Check if we have already authenticated. */
   if (gp->recvRec[i].transState != AUTHENTICATING)
   {
      /* Ignore this reply as it is probably a duplicate. */
      DeQueue(&gp->tsaInQ);
      return;
   }

   /* Check if other things match to make sure that we are
      authenticating the right RR. */
   if (pduInPtr->fmt == 1 &&
       pduInPtr->group != gp->recvRec[i].srcAddr.group)
   {
      /* Group in AuthPDU does not match one in RR. */
#ifdef DEBUG
      DebugMsg("ProcessReply: Group does not match. Ignore reply.");
#endif
      DeQueue(&gp->tsaInQ);
      return;
   }

   if (pduInPtr->transNum != gp->recvRec[i].transNum)
   {
      /* Transaction Number does not match. Ignore reply. */
#ifdef DEBUG
      DebugMsg("ProcessReply: TId does not match. Ignore reply.");
#endif
      DeQueue(&gp->tsaInQ);
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      return;
   }

   /* Now check if the reply matches our encryption. */
   /* First compute the value of E(range, apdu) */

   domainIndex = gp->recvRec[i].srcAddr.domainIndex;
   if (domainIndex == FLEX_DOMAIN)
   {
      /* We should not have initiated challenge in flexDomain. */
      /* Somthing is wrong. Ignore this reply. */
#ifdef DEBUG
      DebugMsg("ProcessReply: Oops! Challenge was unnecessary.");
#endif
      gp->recvRec[i].auth = FALSE; /* Flex Domain? Force it */
      DeQueue(&gp->tsaInQ);
      return;
   }

   Encrypt(gp->recvRec[i].rand,
           gp->recvRec[i].apdu, gp->recvRec[i].apduSize,
           domainIndex, encryptValue);

   if (memcmp(encryptValue, pduInPtr->value.cryptoBytes, 8) == 0)
   {
      /* Matches */
#ifdef DEBUG
      DebugMsg("ProcessReply: Reply matches.");
#endif
      gp->recvRec[i].auth = TRUE;
   }
   else
   {
#ifdef DEBUG
      DebugMsg("ProcessReply: Reply Does not match");
#endif
      nmp->errorLog = AUTHENTICATION_MISMATCH;
      gp->recvRec[i].auth = FALSE;
   }
   gp->recvRec[i].transState = AUTHENTICATED;

   /* Deliver the message now to the application layer. */
   Deliver(i);

   /* Send ack message if it is for transport layer and ackd msg. */
   if (gp->recvRec[i].status == TRANSPORT_RR &&
       gp->recvRec[i].serviceType == ACKD    &&
       gp->recvRec[i].transState == DELIVERED)
   {
      TPSendAck(i);
   }

   DeQueue(&gp->tsaInQ);
   return;
}

/*****************************************************************
Function:  Encrypt
Returns:   None
Reference: CNP, part 1, sub-clause 11.12
Purpose:   To compute the encryption key based on authentication
           key in the domain table, the APDU, and the random
           number given.
Comments:  None
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
*****************************************************************/
static void Encrypt(Byte randIn[], APDU *apduIn, uint16 apduSizeIn,
             Byte domainIndexIn, Byte encryptValueOut[])
{
   int8 i,j,k;
   Byte m, n;
   Byte *apduBytes = (Byte *)apduIn;
   uint16 apduSize = apduSizeIn;

   for (i = 0; i < 8; i++)
   {
      encryptValueOut[i] = randIn[i];
   }

   do
   {
      for (i = 0; i <= 5; i++)
      {
         for (j = 7; j >= 0; j--)
         {
            k = (j + 1) % 8;
            if (apduSize > 0)
            {
               m = apduBytes[--apduSize];
            }
            else
            {
               m = 0;
            }
            n = ~(encryptValueOut[j] + j);
            if (eep->domainTable[domainIndexIn].key[i] &
                (1 << (7 - j)) )
            {
               encryptValueOut[j] =
                  encryptValueOut[k] + m + ((n << 1) + (n >> 7));
            }
            else
            {
               encryptValueOut[j] =
                  encryptValueOut[k] + m - ((n >> 1) + (n << 7));
            }
         }
      }
   } while (apduSize > 0);
}
/*------------------------End of tsa.c-------------------------*/
_
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.10 Application Layer

```
/************************************************************************
          File:        app.c

       Version:        1.7

     Reference:        Section 11, Protocol Specification

       Purpose:        To implement session/application layer.

          Note:        None

         To Do:        None
************************************************************************/

/************************************************************************

 Overview:

 All messages coming into the application layer are handled by
 the APPReceive() function, that dispatches messages to the
 appropriate "Handle" function. Some of the functions explained here
 are local to the application layer and are not available to the application
 program. See "Local Function Prototypes" section below.

 HandleNormal():   Handles normal messages bound to the application program.
                   These are messages from other nodes that are bound to the
                   application program.

 HandleResponse(): Handles normal responses bound to the application program.
                   These are responses due to requests generated by the
                   application program.

 ProcessNV():      Handles incoming NV update and poll messages.
                   These are messages from other nodes or responses for
                   requests generated by the application layer.

 SendVar(): Takes care of network variable updates waiting to be sent.
            i.e implicit messages generated due to output network variable
            updates in the application program.

 PollVar(): Takes care of network variable polls waiting to be dispatched.
            These are messages explicitly generated by the application
            program by calling various forms of poll functions.
            The corresponding variables are input network variables.

 HandleNM(): Processes network management messages. Defn is in netmgmt.c.

 HandleND(): Processes network diagnostic messages. Defn is in netmgmt.c.

 HandleProxyResponse(): Handles responses from proxy messages. Defn is in
                        netmgmt.c. i.e responses received from the target node
                        are relayed back to the original node.

 HandleMsgCompletion(): Handles transaction completion indications from
                        transport, session, and network layers. Some of them
                        are for transactions originated by the application
                        layer itself. Others are for transactions originated
                        by the application program. In that case, the
                        application program is notified of these completion
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                        events by a call to MsgCompletes or NVUpdateCompletes
                        function.

 TryMsgSend():    Helper function that moves messages in the application queue
                  to lower layer queues as appropriate.

 All messages generated by the application program (implicit or explicit)
 are buffered and then sent to lower layers by the APPSend() function.

 APPReset(): Takes care of the application layer reset operations. Allocates
             memory for all queue structures used by the application. Also,
             performs some initialization that are done after each reset.

 APPInit():  Takes care of initializations that are done only during
             power up.

 AddNV():    Takes care of network variable declarations in the application
             program. This function is normally called during power up
             to register all network variables in application program.
             This is done using the function AppInit(). Note that APPInit()
             is application layer function whereas AppInit() is application
             program function.

 RefImpToAlt(), AltToRefImp(): Functions to transform data between a format
 often used by 709 applications (such as msg_in, msg_out, resp_in etc) and
 the corresponding structures for reference implementation (msgIn, msgOut, etc).

 The functions in this file also support the API interface used by application
 programs. e.g. msg_send, resp_send, propagate() etc.

 There is no implicit way of sending network variable updates in the reference
 implementation. The application program should call Propagate or one
 of its variants to actually propagate network variable updates.
*************************************************************************/
/*-------------------------------------------------------------------------
Section: Includes
-------------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>

#include <cnp_1.h>
#include <node.h>
#include <queue.h>
#include <app.h>
#include <api.h>
#include <netmgmt.h>

/*-------------------------------------------------------------------------
Section: Constant and Macro Definitions
-------------------------------------------------------------------------*/
#define MAX_NV_SELF_DOC_LENGTH 1023

/* Depending on whether ALTERNATE_STRUCTURES are used in application
   program or not, define the macros REF_IMP_TO_ALT and ALT_TO_REF_IMP
   accordingly. */
#if defined(ALTERNATE_STRUCTURES_NEEDED)
#define REF_IMP_TO_ALT(a,b) RefImpToAlt(a,b)
#define ALT_TO_REF_IMP(a,b) AltToRefImp(a,b)
#else
#define REF_IMP_TO_ALT(a,b)
#define ALT_TO_REF_IMP(a,b)
#endif

/*-------------------------------------------------------------------------
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Section: Type Definitions
-------------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------------------
Section: Globals
-------------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------------------
Section: Local Function Prototypes
-------------------------------------------------------------------------------*/
static void ProcessNV(APPReceiveParam *appReceiveParamPtr,
                      APDU            *apduPtr);
static void ProcessNVUpdate(APPReceiveParam *appReceiveParamPtr,
                            APDU            *apduPtr);
static void ProcessNVPoll(APPReceiveParam *appReceiveParamPtr,
                          APDU            *apduPtr);

static void HandleMsgCompletion(APPReceiveParam *appReceiveParamPtr,
                                APDU            *apduPtr);
static void HandleResponse(APPReceiveParam *appReceiveParamPtr,
                           APDU            *apduPtr);
static void HandleNormal(APPReceiveParam *appReceiveParamPtr,
                         APDU *apduPtr);

static Status  TryMsgSend(APPSendParam *appSendParamPtr,
                          APDU *apduPtr,
                          Queue   *tsaOutQPtr,
                          Queue   *nwOutQPtr);

static void ReinitMsgOut();
static void ReinitRespOut();

static Status PropagateThisIndex(int16 nvIndexIn, int16 primaryIndex);
static void   PropagateThisPrimary(int16 nvIndexIn);
static void   SendVar(void);

static Status PollThisIndex(int16 nvIndexIn);
static void   PollThisPrimary(int16 nvIndexIn);
static void   PollVar(void);

static Boolean IsArrayNV(int16 nvIndexIn,
                         uint16 *dimOut, int16 *indexOut);

static void ReinitMsgOut(void);
static void ReinitRespOut(void);
/*-------------------------------------------------------------------------------
Section: Function Definitions
-------------------------------------------------------------------------------*/

/*****************************************************************************
Function:  APPInit
Returns:   None
Reference: None
Purpose:   To perform initializations that should be done only
           once when the node is powered on. These are not
           done during node reset.
Comments:  None.
*****************************************************************************/
void APPInit(void)
{
   uint16  len;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
gp->unboundSelector       = 0x3FFF;  /* Countdown as we assign */
gp->nvArrayTblSize        = 0;
gp->nextBindableMsgTag    = 0;
gp->nextNonbindableMsgTag = NUM_ADDR_TBL_ENTRIES;

/************************************************************************
  The SNVT area has the following layout (as expected by Network
  management tools):

  Header:
  uint16 length;
  uint8  numNetvars;
  uint8  version;
  uint8  msbNumNetvars;
  uint8  mtagCount;

  SNVT Desc:
  SNVTdescStruct[];   One struct (2 bytes) per var

  Node Self Doc String:
  char[];      Null terminated; just "\0" if empty

  SNVT Extension Records:
  one byte + various lengths

  Alias Field
  three bytes

  In the reference implementation, the SNVT information is
  stored in nmp->snvt, of type SNVTstruct. It can be
  stored in EEPROM too.

  In SNVTstruct the header fields are stored in explicit
  structure members, but the remaining information is stored
  in the 'sb' member that is simply a buffer of type char.

  When a network variable is added via AddNV():
  1. The Alias Field is saved.
  2. Room is made for the new SNVT Desc by moving all data
  past the last SNVT Desc forward by sizeof(SNVTdesc).
  The descPtr pointer indicates where the move starts.
  3. The new SNVT Desc is added (and descPtr is incremented).
  4. The new extension records are appended after the
  existing extension records.
  5. The Alias Field is restored (placed after the last
  extension record, and pointed to by aliasPtr).

  ************************************************************************/

/* Initialize SNVT area */
nmp->snvt.version       = 1;
nmp->snvt.numNetvars    = 0;
nmp->snvt.msbNumNetvars = 0;
nmp->snvt.mtagCount     = 0;

/* Copy NODE_DOC info if there is sufficient space */
/* If not, init to null string. */
len = strlen(NODE_DOC) + 1;
if (len <= SNVT_SIZE - sizeof(AliasField))
{
   strcpy((char *)&nmp->snvt.sb[0], NODE_DOC);
}
else
{
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      nmp->snvt.sb[0] = '\0';
      len = 1;
   }

   /* Initially, there is no network var related info */
   nmp->snvt.length   = 6 + len + sizeof(AliasField);
   nmp->snvt.descPtr  = (SNVTdescStruct *)&nmp->snvt.sb[0];
   nmp->snvt.aliasPtr = (AliasField *)&nmp->snvt.sb[len];
   nmp->snvt.aliasPtr->bindingII  = TRUE;
   nmp->snvt.aliasPtr->queryStats = TRUE;
   nmp->snvt.aliasPtr->aliasCount = 0x3F;
   nmp->snvt.aliasPtr->hostAlias  = NV_ALIAS_TABLE_SIZE;

   nmp->nvTableSize  = 0;
}

/*****************************************************************************
Function:  APPReset
Returns:   None
Reference: None
Purpose:   To initialize the Queues used by the application layer.
           May be some more.
Comments:  None.
*****************************************************************************/
void APPReset(void)
{
   uint16 queueItemSize;

   /* Allocate and Initialize Input Queue */
   gp->appInBufSize  =
      DecodeBufferSize((uint8)eep->readOnlyData.appInBufSize);
   gp->appInQCnt     = DecodeBufferCnt((uint8)eep->readOnlyData.appInBufCnt);
   queueItemSize     = gp->appInBufSize + sizeof(APPReceiveParam);

   if (QueueInit(&gp->appInQ, queueItemSize, gp->appInQCnt)
       != SUCCESS)
   {
      ErrorMsg("APPReset: Unable to init Input Queue.\n");
      gp->resetOk = FALSE;
      return;
   }

   /* Allocate and Initialize Output Queue */
   gp->appOutBufSize =
      DecodeBufferSize((uint8)eep->readOnlyData.appOutBufSize);
   gp->appOutQCnt    =
      DecodeBufferCnt((uint8)eep->readOnlyData.appOutBufCnt);
   queueItemSize     = gp->appOutBufSize + sizeof(APPSendParam);

   if (QueueInit(&gp->appOutQ, queueItemSize, gp->appOutQCnt)
       != SUCCESS)
   {
      ErrorMsg("APPReset: Unable to init Output Queue.\n");
      gp->resetOk = FALSE;
      return;
   }

   /* Allocate and Initialize Pri Output Queue */
   gp->appOutPriBufSize = gp->appOutBufSize;
   gp->appOutPriQCnt =
      DecodeBufferCnt((uint8)eep->readOnlyData.appOutBufPriCnt);
   queueItemSize     = gp->appOutPriBufSize + sizeof(APPSendParam);

   if (QueueInit(&gp->appOutPriQ, queueItemSize, gp->appOutPriQCnt)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        != SUCCESS)
    {
       ErrorMsg("APPReset: Unable to init Priority Output Queue.\n");
       gp->resetOk = FALSE;
       return;
    }

    /* Allocate Queue for NV output variable scheduling */
    gp->nvOutIndexQCnt    = MAX_NV_OUT;
    gp->nvOutIndexBufSize = 2 + MAX_NV_LENGTH;
    if (QueueInit(&gp->nvOutIndexQ, gp->nvOutIndexBufSize,
                  gp->nvOutIndexQCnt)  != SUCCESS)
    {
       ErrorMsg("APPReset: Unable to init NV Out Index Queue.\n");
       gp->resetOk = FALSE;
       return;
    }
    gp->nvOutStatus      = SUCCESS; /* Propagate succeeds if all the scheduled
                                       transactions complete successfully. */
    gp->nvOutCanSchedule = TRUE;
    gp->nvOutIndex       = 0; /* Not relevant initially */

    /* Allocate Queue for NV input variable scheduling */
    gp->nvInIndexQCnt = MAX_NV_IN;
    if (QueueInit(&gp->nvInIndexQ, 2, gp->nvInIndexQCnt)
        != SUCCESS)
    {
       ErrorMsg("APPReset: Unable to init NV In Index Queue.\n");
       gp->resetOk = FALSE;
       return;
    }
    gp->nvInDataStatus  = FAILURE; /* See node.h for usage */
    gp->nvInTranStatus  = SUCCESS; /* See node.h for usage */
    gp->nvInCanSchedule = TRUE;
    gp->nvInIndex       = 0; /* Not relevant initially */

    /* Set flags to correct state */
    gp->msgReceive      = FALSE;       /* TRUE if data is in gp->msgIn */
    gp->respReceive     = FALSE;       /* TRUE if data is in gp->respIn */
    gp->callMsgFree     = FALSE;
    gp->callRespFree    = FALSE;
    gp->selectQueryFlag = FALSE;       /* FALSE until selected          */

    /* Init msg and resp */
    memset(&gp->msgIn,   0, sizeof(gp->msgIn));
    memset(&gp->msgOut,  0, sizeof(gp->msgOut));
    memset(&gp->respIn,  0, sizeof(gp->respIn));
    memset(&gp->respOut, 0, sizeof(gp->respOut));
    memset(&gp->nvInAddr, 0, sizeof(gp->nvInAddr));
    gp->nvArrayIndex = 0;
#if defined(ALTERNATE_STRUCTURES_NEEDED)
    memset(&msg_in,   0, sizeof(msg_in));
    memset(&msg_out,  0, sizeof(msg_out));
    memset(&resp_in,  0, sizeof(resp_in));
    memset(&resp_out, 0, sizeof(resp_out));
    memset(&nv_in_addr, 0, sizeof(nv_in_addr));
    nv_array_index = 0;
#endif
}

/*****************************************************************************
Function:  HandleMsgCompletion
Returns:   None
Reference: None
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Purpose:   Pass a message completion event to the application
Comments:  None.
********************************************************************************/
static void HandleMsgCompletion(APPReceiveParam *appReceiveParamPtr,
                                APDU            *apduPtr)
{
   Status stat;
   int16 primaryIndex, baseIndex;
   uint16 dim;
   Boolean pollCompletion = TRUE;

   if(appReceiveParamPtr->success)
   {
      stat = SUCCESS;
   }
   else
   {
      stat = FAILURE;
   }

   /* Since App Layer sends messages for NV update
      we may also get completion indication for these.
      See app.h for tag usage. */
   if (appReceiveParamPtr->tag < 0)
   {
      /* Negative tags belong to application layer. */
      /* See app.h for explanation on how the tag is used. */
      if (PROXY_TAG(appReceiveParamPtr->tag))
      {
         /* Proxy Completion. Release the proxy command structure */
         nmp->pxyData.pxyType = -1;
      }
      else if (MANUAL_SERVICE_REQUEST_TAG(appReceiveParamPtr->tag))
      {
         ; /* Ignore. Nothing to do. */
      }
      else
      {
         /* Must be a tag for NV message generated by application layer. */
         if (NV_POLL_TAG(appReceiveParamPtr->tag))
         {
            /* A network variable poll for an input variable will
               succeed if both gp->nvInDataStatus and gp->nvInTranStatus succeed.
               The gp->nvInDataStatus flag is updated by ProcessNVUpdate
               function. */
            if (NV_LAST_TAG(appReceiveParamPtr->tag))
            {
               primaryIndex   = gp->nvInIndex;
               IsArrayNV(primaryIndex, &dim, &baseIndex);
               gp->nvArrayIndex = primaryIndex - baseIndex;
               REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
               /* Set poll completion status */
               if ( (gp->nvInDataStatus == SUCCESS) &&
                    (gp->nvInTranStatus == SUCCESS)
                  )
               {
                  stat = SUCCESS;
               }
               else
               {
                  stat = FAILURE;
               } /* poll completion status */
               gp->nvInDataStatus = FAILURE; /* Reinit */
               gp->nvInTranStatus = SUCCESS;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            if (AppPgmRuns())
            {
               NVUpdateCompletes(stat, baseIndex, gp->nvArrayIndex);
            }
         }
         else
         {
            /* Update the index. Since the last tag does not have the
               index, we need to save it. */
            gp->nvInIndex = NV_INDEX_OF_TAG(appReceiveParamPtr->tag);
            if (stat == FAILURE)
            {
               /* Set the flag to FAILURE as this transaction failed. */
               gp->nvInTranStatus = FAILURE;
            }
         }
         gp->nvInCanSchedule = TRUE; /* Resume scheduling */
      }
      else
      {
         /* NV_UPDATE_TAG */
         /* A network variable update for an output variable will
            succeed if all the transactions scheduled for that
            variable succeed. So, we need to update status flag. */
         if (NV_LAST_TAG(appReceiveParamPtr->tag))
         {
            primaryIndex   = gp->nvOutIndex;
            IsArrayNV(primaryIndex, &dim, &baseIndex);
            gp->nvArrayIndex = primaryIndex - baseIndex;
            REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
            stat = gp->nvOutStatus;
            gp->nvOutStatus = SUCCESS; /* Reinit */
            if (AppPgmRuns())
            {
               NVUpdateCompletes(stat, baseIndex, gp->nvArrayIndex);
            }
         }
         else
         {
            /* Set index. Update gp->nvOutStatus */
            gp->nvOutIndex = NV_INDEX_OF_TAG(appReceiveParamPtr->tag);
            if (stat == FAILURE)
            {
               gp->nvOutStatus = FAILURE;
            }
         }
         gp->nvOutCanSchedule = TRUE; /* Resume scheduling */
      }
   }
}
else
{
   /* Non-negative tags belong to application. */
   MsgCompletes(stat, appReceiveParamPtr->tag);
}

/* Message processing completed - remove it from queue */
DeQueue(&gp->appInQ);
}

/*****************************************************************************
Function:  HandleResponse
Returns:   None
Reference: None
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Purpose:   Handle incoming response message, passing it to the
           application via the gp->respIn global.
Comments:  None.
*******************************************************************************/
static void HandleResponse(APPReceiveParam *appReceiveParamPtr,
                           APDU            *apduPtr)
{
   /* Discard responses received when the node is not CNFG_ONLINE.
      In CNFG_ONLINE, the application can be either running (online)
      or not running (soft-offline). In either of these cases,
      we receive the response and store it in resp_in. Note that
      one reponse can be received at a time from resp_in. If the
      application is offline, then at most one response can be
      stored in resp_in */
   if(eep->readOnlyData.nodeState != CNFG_ONLINE)
   {
      DeQueue(&gp->appInQ); /* Discard response, nothing to do */
      return;
   }

   /* If the application has not processed the previous
    * response, then do nothing, we'll try again next time
    */
   if(gp->respReceive)
   {
      return;
   }

   /* Pass response to application */

   /* Setup gp->respIn */
   gp->respIn.tag  = appReceiveParamPtr->tag;
   gp->respIn.code = apduPtr->code.allBits;
   gp->respIn.len = appReceiveParamPtr->pduSize - 1;

   /* Copy domainIndex even if it is 2. respIn.domainIndex is
      only one bit anyway, so the result is 0 or 1 */
   gp->respIn.addr.domain = appReceiveParamPtr->srcAddr.domainIndex;
   gp->respIn.addr.flexDomain =
      (appReceiveParamPtr->srcAddr.domainIndex == 2);
   memcpy(&gp->respIn.addr.srcAddr,
          &appReceiveParamPtr->srcAddr.subnetAddr,
          sizeof(appReceiveParamPtr->srcAddr.subnetAddr));
   gp->respIn.addr.srcAddr.snodeFlag =
      appReceiveParamPtr->srcAddr.addressMode != MULTICAST_ACK;
   if (gp->respIn.addr.srcAddr.snodeFlag == 0)
   {
      memcpy(&gp->respIn.addr.destAddr.group,
             &appReceiveParamPtr->srcAddr.ackNode,
             sizeof(appReceiveParamPtr->srcAddr.ackNode));
   }
   else if (!gp->respIn.addr.flexDomain)
   {
      /* Fill snode entry only for non-flex domain response */
      gp->respIn.addr.destAddr.snode.subnet =
         eep->domainTable[appReceiveParamPtr->
                          srcAddr.domainIndex].subnet;
      gp->respIn.addr.destAddr.snode.node   =
         eep->domainTable[appReceiveParamPtr->
                          srcAddr.domainIndex].node;
   }
   if (gp->respIn.len <= gp->appInBufSize)
   {
      memcpy(gp->respIn.data, &apduPtr->data, gp->respIn.len);
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      gp->respReceive = TRUE;
#if defined(ALTERNATE_STRUCTURES_NEEDED)
           REF_IMP_TO_ALT(RESP_IN, &resp_in);
#endif
   }
   else
   {
      nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
   }
   /* Message processing completed - remove it from queue */
   DeQueue(&gp->appInQ);
}

/*************************************************************************
Function:  HandleNormal
Returns:   None
Reference: None
Purpose:   Handle incoming normal message
Comments:  If the application program is not running and a request message
           for the application program is received, we send a offline
           message. Note that the node state could be unconfigured or
           soft off-line.
*************************************************************************/
static void HandleNormal(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
   Queue        *tsaOutQPtr;
   TSASendParam *tsaSendParamPtr;
   APDU         *apduRespPtr;

   if(!AppPgmRuns())
   {
      if(appReceiveParamPtr->service == REQUEST)
      {
         /* If not online, send offline response */
         tsaOutQPtr = &gp->tsaRespQ;
         if(QueueFull(tsaOutQPtr))
         {
            /* Can't send response yet - try later */
            return;
         }
         tsaSendParamPtr = QueueTail(tsaOutQPtr);
         tsaSendParamPtr->altPathOverride = FALSE;
         tsaSendParamPtr->service = RESPONSE;
         tsaSendParamPtr->reqId = appReceiveParamPtr->reqId;
         tsaSendParamPtr->nullResponse = FALSE;
         tsaSendParamPtr->apduSize = 1; /* Just the code */
         apduRespPtr = (APDU *)(tsaSendParamPtr + 1);
         if(apduPtr->code.ff.ffFlag == 0x4)
         {
            apduRespPtr->code.allBits = FOREIGN_FRAME_OFFLINE;
         }
         else
         {
            apduRespPtr->code.allBits = APPL_MSG_OFFLINE;
         }
         EnQueue(tsaOutQPtr);
      }
      DeQueue(&gp->appInQ); /* Discard msg, nothing to do */
      return;
   }

   /* If the application has not processed the previous
    * message, then do nothing, we'll try again next time  */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
if(gp->msgReceive)
{
   return;
}

/* Pass message to application */

/* Setup gp->msgIn */
gp->msgIn.code = apduPtr->code.allBits;
gp->msgIn.len = appReceiveParamPtr->pduSize - 1;
if (gp->msgIn.len <= gp->appInBufSize)
{
   memcpy(gp->msgIn.data, &apduPtr->data, gp->msgIn.len);
}
else
{
   nmp->errorLog = WRITE_PAST_END_OF_APPL_BUFFER;
   DeQueue(&gp->appInQ);
   return;

}

gp->msgIn.authenticated = appReceiveParamPtr->auth;
gp->msgIn.service = appReceiveParamPtr->service;
gp->msgIn.reqId = appReceiveParamPtr->reqId;
/* Fill in addr structure */
gp->msgIn.addr.domain = appReceiveParamPtr->srcAddr.domainIndex;
gp->msgIn.addr.flexDomain =
   (appReceiveParamPtr->srcAddr.domainIndex == 2);
/* Copy Source Address */
memcpy(&gp->msgIn.addr.srcAddr,
       &appReceiveParamPtr->srcAddr.subnetAddr,
       sizeof(gp->msgIn.addr.srcAddr));
switch(appReceiveParamPtr->srcAddr.addressMode)
{
   case BROADCAST:
      gp->msgIn.addr.format  = 0;
      gp->msgIn.addr.destAddr.bcastSubnet =
         appReceiveParamPtr->srcAddr.broadcastSubnet;
      break;
   case MULTICAST:
      gp->msgIn.addr.format  = 1;
      gp->msgIn.addr.destAddr.group =
         appReceiveParamPtr->srcAddr.group;
      break;
   case SUBNET_NODE:
      gp->msgIn.addr.format  = 2;
      if (!gp->msgIn.addr.flexDomain)
      {
         gp->msgIn.addr.destAddr.snode.subnet =
            eep->domainTable[gp->msgIn.addr.domain].subnet;
         gp->msgIn.addr.destAddr.snode.node  =
            eep->domainTable[gp->msgIn.addr.domain].node;
      }
      break;
   case UNIQUE_NODE_ID:
      gp->msgIn.addr.format  = 3;
      gp->msgIn.addr.destAddr.uniqueNodeId.subnet = 0; /* Not stored */
      memcpy(gp->msgIn.addr.destAddr.uniqueNodeId.uniqueId,
             eep->readOnlyData.uniqueNodeId,
             UNIQUE_NODE_ID_LEN);
      break;
   default:
      /* should not come here */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        gp->msgIn.addr.format  = 5; /* unknown. arbitrary 5 */
    }

    gp->msgReceive = TRUE;
    REF_IMP_TO_ALT(MSG_IN, &msg_in);
    /* Message processing completed - remove it from queue */
    DeQueue(&gp->appInQ);
}


/************************************************************************
Function:  APPSend
Returns:   None
Reference: None
Purpose:   Process send side of the application layer
           Send one message of each of the following types if there is space.
            pri msg, nv update message, nv poll message, non-pri msg.
Comments:  Called by scheduler loop.
************************************************************************/
void APPSend(void)
{
    Queue         *appOutQPtr;
    Queue         *tsaOutQPtr;
    Queue         *nwOutQPtr;
    APPSendParam  *appSendParamPtr;
    APDU          *apduPtr;
    Status        status;

    if(gp->manualServiceRequest)
    {
        ManualServiceRequestMessage();
        gp->manualServiceRequest = FALSE;
    }

    /* Call MsgFree or RespFree implicitely if needed. */
    if (gp->callMsgFree)
    {
        MsgFree();
    }
    if (gp->callRespFree)
    {
        RespFree();
    }

    /* Send a priority message if we can */
    if (!QueueEmpty(&gp->appOutPriQ))
    {
        appOutQPtr      = &gp->appOutPriQ;
        tsaOutQPtr      = &gp->tsaOutPriQ;
        nwOutQPtr       = &gp->nwOutPriQ;
        appSendParamPtr = QueueHead(appOutQPtr);
        apduPtr         = (APDU *)(appSendParamPtr + 1);
        status          = TryMsgSend(appSendParamPtr, apduPtr,
                                     tsaOutQPtr, nwOutQPtr);
        if (status == SUCCESS)
        {
            /* We have moved this message. Discard it */
            DeQueue(appOutQPtr);
        }
    }

    /* Process one NV output variable scheduled, if any. */
    SendVar();
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* Process one NV input variable (Poll) scheduled, if any. */
   PollVar();

   /* Send a non-priority message if we can */
   if (!QueueEmpty(&gp->appOutQ))
   {
      appOutQPtr      = &gp->appOutQ;
      tsaOutQPtr      = &gp->tsaOutQ;
      nwOutQPtr       = &gp->nwOutQ;
      appSendParamPtr = QueueHead(appOutQPtr);
      apduPtr         = (APDU *)(appSendParamPtr + 1);
      status          = TryMsgSend(appSendParamPtr, apduPtr,
                                   tsaOutQPtr, nwOutQPtr);
      if (status == SUCCESS)
      {
         /* We have moved this message. Discard it */
         DeQueue(appOutQPtr);
      }

   }
}


/************************************************************************
Function:  APPReceive
Returns:   None
Reference: None
Purpose:   Process receive side of the application layer.
Comments:  Called by scheduler loop.
************************************************************************/
void APPReceive(void)
{
   APPReceiveParam     *appReceiveParamPtr;
   APDU                *apduPtr;    /* ptr to APDU being received  */

   /* Check if anything to process */
   if(QueueEmpty(&gp->appInQ))
   {
      return; /* Nothing to process */
   }

   /* Set the pointer to APDU in appInQ */
   appReceiveParamPtr = QueueHead(&gp->appInQ);
   apduPtr            = (APDU *)(appReceiveParamPtr + 1);

   if(appReceiveParamPtr->indication == COMPLETION)
   {
      HandleMsgCompletion(appReceiveParamPtr,apduPtr);
   }
   else if(appReceiveParamPtr->service == RESPONSE &&
      appReceiveParamPtr->tag >= 0)
   {
      /* The response belongs to the application program.
         Deliver it to the application irrespective of
         what type of message it is. If the application
         sends network management/diagnostic messages,
         it is its responsibility to handle them */
      HandleResponse(appReceiveParamPtr,apduPtr);
   }
   /* Handle Response from a Network Diagnostic Proxy Command */
   /* pxyType is initialized to -1. If a proxy command has been
      received, it is set to the command requested (0 or 1 or 2).
      The command is forwarded using a tag value of PROXY_TAG.
      So, check if this message is a response and is a response
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      to a previously forwarded proxy command.
      The response should match (success or failure) of
      original the request */
   else if(nmp->pxyData.pxyType != -1 &&
           PROXY_TAG(appReceiveParamPtr->tag) &&
           appReceiveParamPtr->service == RESPONSE &&
           (apduPtr->code.allBits == 0x32 ||
            apduPtr->code.allBits == 0x12)
          )
   {
      /* Indeed we have a good proxy response. Forward to source */
      HandleProxyResponse(appReceiveParamPtr,apduPtr);
   }
   else if (appReceiveParamPtr->service == RESPONSE &&
            PROXY_TAG(appReceiveParamPtr->tag))
   {
      /* ill-formed proxy response. Ignore. */
      DeQueue(&gp->appInQ);
   }
   else if(apduPtr->code.nm.nmFlag == 0x3)
   {
      /* Network management messages */
      HandleNM(appReceiveParamPtr,apduPtr);
   }
   else if(apduPtr->code.nd.ndFlag == 0x5)
   {
      /* Network diagnostic messages */
      HandleND(appReceiveParamPtr,apduPtr);
   }
   else if(apduPtr->code.nv.nvFlag == 0x1)
   {
      /* Network variable update/poll messages */
      ProcessNV(appReceiveParamPtr,apduPtr);
   }
   else if(apduPtr->code.ap.apFlag == 0x0 || apduPtr->code.ff.ffFlag == 0x4)
   {
      /* Messages bound to the application program. App Msg or Foriegn Frames. */
      HandleNormal(appReceiveParamPtr,apduPtr);
   }
   else
   {
      /* Not supported - just discard */
      DeQueue(&gp->appInQ);
   }

}

/************************************************************************
Function:  TryMsgSend
Returns:   SUCCESS if message sent, FAILURE otherwise
Reference: None
Purpose:   If room is available, move message from app queue to
           to tsa or nw output queue.
Comments:  None.
************************************************************************/
static Status TryMsgSend(APPSendParam *appSendParamPtr,
                         APDU *apduPtr,
                         Queue   *tsaOutQPtr,
                         Queue   *nwOutQPtr)
{
   TSASendParam  *tsaSendParamPtr;
   NWSendParam   *nwSendParamPtr;
   APDU          *apduSendPtr;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* If the address is bad, don't even bother sending it */
   if(appSendParamPtr->addr.noAddress == UNBOUND)
   {
      /* TurnAround is not possible with MsgOutAddr */
      MsgCompletes(SUCCESS, appSendParamPtr->tag);
      return(SUCCESS);
   }

   /* Simple unacknowledged messages go to network layer */
   if(appSendParamPtr->service == UNACKD)
   {
      if(QueueFull(nwOutQPtr))
      {
         return(FAILURE); /* Can't send message yet - try later */
      }

      nwSendParamPtr = QueueTail(nwOutQPtr);

      nwSendParamPtr->dropIfUnconfigured = TRUE;

      switch(appSendParamPtr->addr.noAddress)
      {
         case UNBOUND:
            /* Can't happen. We took care above */
            return(SUCCESS); /* doesn't matter what we return */
         case SUBNET_NODE:
            nwSendParamPtr->destAddr.addressMode = SUBNET_NODE;
            nwSendParamPtr->destAddr.domainIndex =
               appSendParamPtr->addr.snode.domainIndex;
            nwSendParamPtr->destAddr.addr.addr2a.subnet =
               appSendParamPtr->addr.snode.subnetID;
            nwSendParamPtr->destAddr.addr.addr2a.node   =
               appSendParamPtr->addr.snode.node;
            break;

         case UNIQUE_NODE_ID:
            nwSendParamPtr->destAddr.addressMode = UNIQUE_NODE_ID;
            nwSendParamPtr->destAddr.domainIndex =
               appSendParamPtr->addr.uniqueNodeId.domainIndex;
            nwSendParamPtr->destAddr.addr.addr3.subnet =
               appSendParamPtr->addr.uniqueNodeId.subnetID;
            memcpy(nwSendParamPtr->destAddr.addr.addr3.uniqueId,
                  appSendParamPtr->addr.uniqueNodeId.uniqueId,
UNIQUE_NODE_ID_LEN);
            break;

         case BROADCAST:
            nwSendParamPtr->destAddr.addressMode = BROADCAST;
            nwSendParamPtr->destAddr.domainIndex =
               appSendParamPtr->addr.bcast.domainIndex;
            nwSendParamPtr->destAddr.addr.addr0 =
               appSendParamPtr->addr.bcast.subnetID;
            break;

         default: /* MUST BE GROUP FORMAT */
            nwSendParamPtr->destAddr.addressMode = MULTICAST;
            nwSendParamPtr->destAddr.domainIndex =
               appSendParamPtr->addr.group.domainIndex;
            nwSendParamPtr->destAddr.addr.addr1 =
               appSendParamPtr->addr.group.groupID;
            break;
      } /* switch */

      nwSendParamPtr->tag     = appSendParamPtr->tag;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      nwSendParamPtr->pduType = APDU_TYPE;
      nwSendParamPtr->deltaBL = 0;
      /* unacknowledged messages do not use alternate path */
      nwSendParamPtr->altPath = FALSE;
      nwSendParamPtr->pduSize = appSendParamPtr->len + 1;

      apduSendPtr = (APDU *)(nwSendParamPtr + 1);
      if (nwSendParamPtr->pduSize <= gp->nwOutBufSize)
      {
         memcpy(apduSendPtr, apduPtr, appSendParamPtr->len + 1);
         EnQueue(nwOutQPtr);
         /* Don't give completion yet. The network layer will send the
            completion indication in gp->appInQ */
      }
      else
      {
         /* Losing this packet as it is too large */
         MsgCompletes(FAILURE, appSendParamPtr->tag);
      }

      return(SUCCESS);
   }

   if(QueueFull(tsaOutQPtr))
   {
      return(FAILURE); /* Can't send message yet - try later */
   }

   /* All other service types go to TSA layers */
   tsaSendParamPtr = QueueTail(tsaOutQPtr);

   tsaSendParamPtr->domainIndex = 3;
   memcpy(&tsaSendParamPtr->destAddr,
          &appSendParamPtr->addr,
          sizeof(appSendParamPtr->addr));
   tsaSendParamPtr->service   = appSendParamPtr->service;
   tsaSendParamPtr->auth      = appSendParamPtr->authenticated;
   tsaSendParamPtr->reqId     = appSendParamPtr->reqId;
   tsaSendParamPtr->tag       = appSendParamPtr->tag;
   tsaSendParamPtr->apduSize  = appSendParamPtr->len + 1;
   tsaSendParamPtr->altPathOverride = FALSE;

   apduSendPtr = (APDU *)(tsaSendParamPtr + 1);
   if (tsaSendParamPtr->apduSize <= gp->tsaOutBufSize)
   {
      memcpy(apduSendPtr, apduPtr, tsaSendParamPtr->apduSize);
      EnQueue(tsaOutQPtr);
   }
   else
   {
      /* Losing this message */
      MsgCompletes(FAILURE, appSendParamPtr->tag);
   }

   return(SUCCESS);
}

static void ReinitMsgOut(void)
{
   /* Reinit whatever fields need to be reinitialized after each msg_send */
   memset(&gp->msgOut, 0, sizeof(gp->msgOut));
#if defined(ALTERNATE_STRUCTURES_NEEDED)
   memset(&msg_out, 0, sizeof(msg_out));
#endif
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
}

static void ReinitRespOut(void)
{
   /* Reinit whatever fields need to be reinitialized after each resp_send */
   memset(&gp->respOut, 0, sizeof(gp->respOut));
#if defined(ALTERNATE_STRUCTURES_NEEDED)
   memset(&resp_out, 0, sizeof(resp_out));
#endif
}

/*************************************************************************
Function:  MsgAlloc
Returns:   None
Reference: None
Purpose:   Determines if there is room for a message to be sent.
Comments:  None.
*************************************************************************/
Boolean MsgAlloc(void)
{
   return(!QueueFull(&gp->appOutQ));
}

/* For nc compatibility */
Boolean msg_alloc(void)
{
   return(!QueueFull(&gp->appOutQ));
}

/*************************************************************************
Function:  MsgAllocPriority
Returns:   None
Reference: None
Purpose:   Determines if there is room for a pri message to be sent.
Comments:  None.
*************************************************************************/
Boolean MsgAllocPriority(void)
{
   return(!QueueFull(&gp->appOutPriQ));
}

/* For nc compatibility */
Boolean msg_alloc_priority(void)
{
   return(!QueueFull(&gp->appOutPriQ));
}

/*************************************************************************
Function:  MsgSend
Returns:   None
Reference: None
Purpose:   The application calls this function to send a message.
           The message is placed in the application queue for
           processing by APPSend.
Comments:  None.
*************************************************************************/
void MsgSend(void)
{
   Queue           *outQptr;
   APPSendParam    *appSendParamPtr;
   APDU            *apduPtr;
   AddrTableEntry  *ap;
   uint16           addrIndex;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
ALT_TO_REF_IMP(MSG_OUT, &msg_out);

if(gp->msgOut.priorityOn)
{
   outQptr = &gp->appOutPriQ;
}
else
{
   outQptr = &gp->appOutQ;
}

/* Negative tags are reserved for application layer.
   Applications are not allowed to use negative tags */
if(QueueFull(outQptr) || gp->msgOut.tag < 0 ||
   (gp->msgOut.service >= RESPONSE))
{
   /* Bad Tag  OR
      Bad Service OR
      No place to put the message - discard it. This should
      not happen if application called MsgAlloc or
      MsgPriorityAlloc before forming the message */
   MsgCompletes(FAILURE, gp->msgOut.tag);
   ReinitMsgOut();
   return;
}

appSendParamPtr                = QueueTail(outQptr);
appSendParamPtr->tag           = gp->msgOut.tag;
appSendParamPtr->len           = gp->msgOut.len;
appSendParamPtr->authenticated = gp->msgOut.authenticated;
appSendParamPtr->service       = gp->msgOut.service;
appSendParamPtr->addr          = gp->msgOut.addr;
apduPtr                        = (APDU *)(appSendParamPtr + 1);
apduPtr->code.allBits          = gp->msgOut.code;
if (appSendParamPtr->len + 1 <= gp->appOutBufSize)
{
   /* There is space in the queue item to copy data */
   memcpy((char *)apduPtr+1,
          gp->msgOut.data,
          appSendParamPtr->len);
}
else
{
   /* We are losing this message as it is too big. */
   MsgCompletes(FAILURE, appSendParamPtr->tag);
   ReinitMsgOut();
   return;
}

/* Use implicit addressing if the tag value corresponds to an address
   table entry and the explicit address is unbound or turnaround.
   Thus explicit address can be used to override implicit addressing.
   Note that turnaround is not allowed with explicit messages. */
if(appSendParamPtr->tag < NUM_ADDR_TBL_ENTRIES &&
   gp->msgOut.addr.noAddress == UNBOUND)
{
   addrIndex = appSendParamPtr->tag;
   ap = AccessAddress(addrIndex); /* ap cannot be NULL */
   if (ap == NULL || ap->addrFormat == UNBOUND)
   {
      /* ap cannot be NULL, but we can be safe in checking it anyway.
         We lose this message as the address table entry is unbound
         or turnaround. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
         MsgCompletes(FAILURE, appSendParamPtr->tag);
         ReinitMsgOut();
         return;
      }
      memcpy(&appSendParamPtr->addr,
             ap,
             sizeof(MsgOutAddr));
   }
   EnQueue(outQptr);
   ReinitMsgOut();
}

/* For nc compatibility */
void msg_send(void)
{
   MsgSend();
}

/************************************************************************
Function:  MsgCancel
Returns:   None
Reference: None
Purpose:   The application calls this function to cancel a
           previous message allocation.
Comments:  None.
************************************************************************/
void MsgCancel(void)
{
   /* Nothing to do */
}

/* For nc compatibility */
void msg_cancel()
{
}

/************************************************************************
Function:  MsgFree
Returns:   None
Reference: None
Purpose:   Releases gp->msgIn (by clearning gp->msgReceive) so that the
           next message can be copied to gp->msgIn.
Comments:  None.
************************************************************************/
void MsgFree(void)
{
   gp->msgReceive = FALSE; /* TRUE when data is in gp->msgIn */
   gp->callMsgFree = FALSE;
}

/* For nc compatibility */
void msg_free(void)
{
   gp->msgReceive = FALSE;
   gp->callMsgFree = FALSE;
}

/* TRUE if there is msg to be received */
Boolean  msgReceive(void)
{
   if (gp->msgReceive)
   {
      /* There is a message to be received. Need to call msg_free() */
      gp->callMsgFree = TRUE;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }
   return(gp->msgReceive);
}

/* For nc compatibility */
Boolean  msg_receive(void)
{
   if (gp->msgReceive)
   {
      /* There is a message to be received. Need to call msg_free() */
      gp->callMsgFree = TRUE;
   }
   return(gp->msgReceive);
}

/************************************************************************
Function:  RespAlloc
Returns:   None
Reference: None
Purpose:   Check if there is space for sending a response.
Comments:  None.
************************************************************************/
Boolean RespAlloc(void)
{
   if (!QueueFull(&gp->tsaRespQ))
   {
      return(TRUE);
   }
   return(FALSE);
}

/* For nc compatibility */
Boolean resp_alloc(void)
{
   return(RespAlloc());
}

/************************************************************************
Function:  RespSend
Returns:   None
Reference: None
Purpose:   Reads gp->respOut, sends the response message
Comments:  None.
************************************************************************/
void RespSend(void)
{
   TSASendParam *tsaSendParamPtr;
   APDU         *apduRespPtr;

   ALT_TO_REF_IMP(RESP_OUT, &resp_out);

   if (gp->respOut.reqId == 0)
   {
      /* Application did not initialize it. Let us use as default
         the reqId of message currently in gp->msgIn */
      gp->respOut.reqId = gp->msgIn.reqId;
   }

   /* Place the response in tsaRespQ. If it is full, discard the
      response. There should be space if application called
      RespAlloc before forming the response */
   if (!QueueFull(&gp->tsaRespQ))
   {
      tsaSendParamPtr               = QueueTail(&gp->tsaRespQ);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);
      tsaSendParamPtr->altPathOverride = FALSE;
      tsaSendParamPtr->service      = RESPONSE;
      tsaSendParamPtr->reqId        = gp->respOut.reqId;
      tsaSendParamPtr->apduSize     = gp->respOut.len + 1;
      tsaSendParamPtr->nullResponse = gp->respOut.nullResponse;
      apduRespPtr->code.allBits     = gp->respOut.code;
      if (tsaSendParamPtr->apduSize <= gp->tsaRespBufSize)
      {
         memcpy((char *)apduRespPtr + 1,
                gp->respOut.data,
                gp->respOut.len);
         EnQueue(&gp->tsaRespQ);
      }
      else
      {
         /* Lose this response as it is too big to fit */
         /* There is no errorLog or stat to record this */
         ;
      }
   }
   else
   {
      /* Sorry! Application is sending a response without checking
         for space first */
      ;
   }
   ReinitRespOut();
}

/* For nc compatibility */
void resp_send(void)
{
   RespSend();
}

/*****************************************************************************
Function:  RespCancel
Returns:   None
Reference: None
Purpose:   The application calls this function to cancel a
           previous response allocation.
Comments:  None.
*****************************************************************************/
void RespCancel(void)
{
   /* Nothing to do */
}

/* For nc compatibility */
void resp_cancel(void)
{
}


/*****************************************************************************
Function:  RespFree
Returns:   None
Reference: None
Purpose:   Sets gp->respReceive to FALSE, which will allow a new
           response to be received.
Comments:  None.
*****************************************************************************/
void RespFree(void)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
{
   gp->respReceive = FALSE; /* TRUE when data is in gp->respIn */
   gp->callRespFree = FALSE;
}

/* For nc compatibility */
void resp_free(void)
{
   gp->respReceive = FALSE;
   gp->callRespFree = FALSE;
}

/* Returns TRUE if there is resp to be received */
Boolean RespReceive(void)
{
   if (gp->respReceive)
   {
      /* There is a response to be received. Need to call resp_free() */
      gp->callRespFree = TRUE;
   }
   return(gp->respReceive);
}

/* For nc comptability */
Boolean resp_receive(void)
{
   if (gp->respReceive)
   {
      /* There is a response to be received. Need to call resp_free() */
      gp->callRespFree = TRUE;
   }
   return(gp->respReceive);
}

void RefImpToAlt(RefImpAltStruct whatIn, void *destAddr)
{
   switch(whatIn)
   {
      case MSG_IN:
         memcpy(destAddr, &gp->msgIn, sizeof(gp->msgIn));
         break;
      case MSG_OUT:
         memcpy(destAddr, &gp->msgOut, sizeof(gp->msgOut));
         break;
      case RESP_IN:
         memcpy(destAddr, &gp->respIn, sizeof(gp->respIn));
         break;
      case RESP_OUT:
         memcpy(destAddr, &gp->respOut, sizeof(gp->respOut));
         break;
      case NV_IN_ADDR:
         memcpy(destAddr, &gp->nvInAddr, sizeof(gp->nvInAddr));
         break;
      case NV_ARRAY_INDEX:
         memcpy(destAddr, &gp->nvArrayIndex, sizeof(gp->nvArrayIndex));
         break;
      default:
         /* Do nothing */
         ;
   }
}
void AltToRefImp(RefImpAltStruct whatIn, void *srcAddr)
{
   switch(whatIn)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   {
      case MSG_IN:
         memcpy(&gp->msgIn, srcAddr, sizeof(gp->msgIn));
         break;
      case MSG_OUT:
         memcpy(&gp->msgOut, srcAddr, sizeof(gp->msgOut));
         break;
      case RESP_IN:
         memcpy(&gp->respIn, srcAddr, sizeof(gp->respIn));
         break;
      case RESP_OUT:
         memcpy(&gp->respOut, srcAddr, sizeof(gp->respOut));
         break;
      case NV_IN_ADDR:
         memcpy(&gp->nvInAddr, srcAddr, sizeof(gp->nvInAddr));
         break;
      case NV_ARRAY_INDEX:
         memcpy(&gp->nvArrayIndex, srcAddr, sizeof(gp->nvArrayIndex));
         break;
      default:
         /* Do nothing */
         ;
   }
}

/*****************************************************************************
Function:  AddNV
Returns:   Network variable index. For arrays, the base index is returned.
Reference: None
Purpose:   Adds a new network variable. This involves
           adding an entry into nvConfigTable, nvFixedTable,
           SNVT information, if present etc.
           The return value is the index assigned to the variable.
           For arrays, each element is like a separate network
           variable. So, multiple entries are added to the
           tables. However, only the base index is returned.
Comments:  Format of the snvt.sb space is as follows:
   Self-Id for each network variable (SNVTdescStruct)
      (For arrays, one entry for each element)
   Node Self-Doc string.
   Self-Doc for each network variable
      (SNVTExtension & variable part)
   Self-Id for binding and status (AliasField)
*****************************************************************************/
int16 AddNV(NVDefinition *dp)
{
   uint16          i, nvSelfIdCnt;
   uint16          nvNameLen; /* Length for name of network variable. */
   uint16          docLen;    /* Length for self-doc for network var. */
   uint16          selectorVal;
   uint16          sizeNeeded;
   uint16          dim;
   uint16          jumpBy;    /* Number of bytes by that we shift sb array. */
   uint8           remainder,quotient;
   SNVTextension  *se;
   SNVTdescStruct *sd;
   AliasField      saveAlias; /* To save old alias field. */
   char           *p;
   char           *extPtr;  /* Points to where the new ext rec can be stored.*/
   char           *endOfSb; /* Points to end of sb array. */

   /* Initialize local pointers for structure information in dp so that
      we can use field names in these structures instead of explicit
      bit operations */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
se = (SNVTextension *)  &dp->snvtExt;
/* snvtType info is not part of the snvtDesc. We only want to access the
   first byte of the structure, anyway. */
sd = (SNVTdescStruct *) &dp->snvtDesc;

/* First determine the number of entries to be added to nvConfigTable */
if (dp->arrayCnt > 0)
{
   dim = dp->arrayCnt; /* Array Variable. dim can still be 1. */
}
else
{
   dim = 1; /* Simple variable */
}

if (dp->nvName == NULL || dp->varAddr == NULL)
{
   return(-1); /* Network variable name and address is a must. */
}

if (nmp->nvTableSize + dim >  NV_TABLE_SIZE)
{
   /* Not enough space in network variable table. */
   return(-1);
}

/* Save the original alias field. */
saveAlias = *nmp->snvt.aliasPtr;

/* Make endOfSb point to last element of sb array. We never want to go
   past this element */
endOfSb = (char *)&nmp->snvt.sb[SNVT_SIZE - 1];

/* Compute the size needed for this variable in SNVT structure. */
if (dp->arrayCnt > 0 && dp->explodeArray)
{
   /* We need one entry for each network variable. */
   nvSelfIdCnt = dim;
}
else
{
   nvSelfIdCnt = 1;

}
sizeNeeded = sizeof(SNVTdescStruct) * nvSelfIdCnt;

/* Fixed extRec. One for each entry in self id desc part. */
if (sd->extRec) /* Check ext_rec bit */
{
   /* Need space for fixed extension record too */
   sizeNeeded = sizeNeeded + sizeof(SNVTextension) * nvSelfIdCnt;
}

/* Variable Part of Extension Record. */
/* One for each entry in self id desc part. */
/* See Page 9-23 of Technology Device Data Book Rev 3 */
if (sd->extRec && se->mre)
{
   sizeNeeded += nvSelfIdCnt; /* One byte for maximum rate */
}
if (sd->extRec && se->re)
{
   sizeNeeded += nvSelfIdCnt; /* One byte for average rate */
}
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   if (sd->extRec && se->nm)
   {
      nvNameLen = strlen(dp->nvName) + 1;
      if (dp->arrayCnt > 0 && dp->explodeArray)
      {
         /* Need space for [ ] and up to 3 bytes for index */
         nvNameLen += 5;
         if (nvNameLen > 22)
         {
            return(-1);
         }
      }
      else if (nvNameLen > 17)
      {
         return(-1); /* Only up to 17 bytes supported */
      }

      sizeNeeded += (nvNameLen * nvSelfIdCnt);
   }
   if (sd->extRec && se->sd)
   {
      if (dp->nvSdoc)
      {
         docLen = strlen(dp->nvSdoc) + 1;
      }
      else
      {
         docLen = 1; /* For null character */
      }
      if (docLen > MAX_NV_SELF_DOC_LENGTH)
      {
         return(-1);
      }
      sizeNeeded += (docLen * nvSelfIdCnt);
   }

   if (sd->extRec & se->nc)
   {
      /* 16 bit count for # of network variables of this type. dim? */
      /* Reference implementation uses ver 1. */
      sizeNeeded += (2 * nvSelfIdCnt);
   }

   if ((char *)nmp->snvt.aliasPtr + sizeof(AliasField) + sizeNeeded
       > endOfSb)
   {
      /* No Space for the new snvt_desc_struct and extension rec. */
      return(-1);
   }

   /* For arrays, we need space in gp->nvArrayTblSize */
   if (dp->arrayCnt > 0 && gp->nvArrayTblSize == MAX_NV_ARRAYS)
   {
      /* No more space to save array information */
      return(-1);
   }

   /*************************************************************************
      dp->bind = TRUE ==> the network variable is bindable. Bindable
      variables are automatically given selector numbers in the range
      0x3000-0x3FFF (unbound network variables). Bindable means they
      are currently unbound.

      dp->bind = FALSE ==> the network variable is non-bindable. i.e
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   the application program assigns selector numbers and these variables
   are cannot be bound by the binder tools. In this case, it is
   the responsibility of the application program to assign these
   selector numbers in a reasonable way. For array variables, the
   application program indicates the selector for the first element
   and the other elements automatically get the previous selector
   numbers. The selector numbers count down.
*************************************************************************/

/* We need dim many selectors. dim can be 1. */
/* bindable variables use unboundSelectors   */
selectorVal = gp->unboundSelector; /* initialize */
if (dp->bind && gp->unboundSelector - dim + 1 < 0x3000)
{
   /* Not enough selector numbers available for assigning */
   return(-1);
}
else if (!dp->bind)
{
   /* selectors are specified by the application program.
      For arrays, only the selector for first element is
      given. Application program has the responsibility to
      assign unique selectors for all variables */

   selectorVal = dp->selectorHi;
   selectorVal = (selectorVal << 8) + dp->selectorLo;
   /* We are using dim selectors. Make sure that all are in range */
   if (selectorVal + dim - 1 > 0x2FFF)
   {
      /* Nonbindable variables should have value in 0-0x2FFF */
      return(-1);
   }
}

/* Now selectorVal has the selector number to be assigned */

/* Everything is fine. We are now ready to add this variable */

/* Make extPtr point to where the new extension rec would go */
/* aliasPtr points to the byte following the last byte of ext records */
extPtr = (char *)nmp->snvt.aliasPtr + sizeof(SNVTdescStruct) * nvSelfIdCnt;

/* Add entry or entries into the nvConfigTable and Fixed Table */
/* Add one entry for each array element irrespective of whether it
   is exploded or not. */
for (i = nmp->nvTableSize; i < nmp->nvTableSize + dim; i++)
{
   /* nv config table */
   nmp->nvConfigTable[i].nvPriority   = dp->priority;
   nmp->nvConfigTable[i].nvDirection  = dp->direction;
   nmp->nvConfigTable[i].nvSelectorHi = selectorVal >> 8;
   nmp->nvConfigTable[i].nvSelectorLo = selectorVal & 0xFF;
   selectorVal--; /* automatic countdown - for both bind & non-bind */
   if (dp->bind)
   {
      gp->unboundSelector--; /* Update this as we just used up one */
   }
   nmp->nvConfigTable[i].nvTurnaround = dp->turnaround;
   nmp->nvConfigTable[i].nvService    = dp->service;
   nmp->nvConfigTable[i].nvAuth       = dp->auth;
   nmp->nvConfigTable[i].nvAddrIndex  = 0xF;

   /* nv fixed table */
   nmp->nvFixedTable[i].nvSync     = sd->nvSync;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   nmp->nvFixedTable[i].nvLength   = dp->nvLength;
   /* For arrays, make sure we compute the address of each item. */
   nmp->nvFixedTable[i].nvAddress  = (char *) dp->varAddr +
                                     (i - nmp->nvTableSize) * dp->nvLength;
}

/* If we are adding an array, save this info in gp->nvArrayTbl */
/* Array with dim 1 is considered like an array */
if (dp->arrayCnt > 0)
{
   gp->nvArrayTbl[gp->nvArrayTblSize].nvIndex = nmp->nvTableSize;
   gp->nvArrayTbl[gp->nvArrayTblSize++].dim   = dim;
}

/*************************************************************************
Format of the SNVT structure: (See APPReset for more info).
   snvtheader nv-self-id-desc node-self-doc nv-self-doc alias-field

   snvtheader is part of the nmp->snvt structure, its initial members.
   nmp->snvt.sb array is used for [nv-self-id-desc ... alias-field]
   alias-fields has binding and status information.
   nmp->snvt.descPtr points to beginning address of node-self-doc.
         i.e the address where the nv-self-id-desc for new network
         variable should be placed.
*************************************************************************/

/* Add data to snvt structure */

/* Update snvt descriptor count in snvt structure.
   nmp->snvt.msbNumNetvars is the most significant byte.
   nmp->snvt.numNetvars is the least significant byte.
   We want to add the value of dim to this number */
quotient  = (nmp->snvt.numNetvars + nvSelfIdCnt) / 256;
remainder = (nmp->snvt.numNetvars + nvSelfIdCnt) % 256;
nmp->snvt.numNetvars     = remainder;
nmp->snvt.msbNumNetvars += quotient;

/* Now we need to move nv-self-doc and node-self-doc to make
   for self-id-desc for the network variable being added. */

/* Make p point the last byte of nv-self-doc */
p = (char *)nmp->snvt.aliasPtr - 1;
/* Determine how many bytes everything moves by. */
jumpBy = sizeof(SNVTdescStruct) * nvSelfIdCnt;
while(p >= (char *)(nmp->snvt.descPtr))
{
   *(p + jumpBy) = *p;
   p--;
}

/* Copy dp->snvtDesc information. One for each array item.     */
/* All array items have the same information. i.e homogeneous. */
sd = nmp->snvt.descPtr; /* Initialize */
for (i = 0; i < nvSelfIdCnt; i++)
{
   memcpy(sd, &dp->snvtDesc, 1); /* can't assign without casting. */
   sd->snvtTypeIndex = dp->snvtType;
   sd++;
}
nmp->snvt.descPtr = sd;

sd = (SNVTdescStruct *)&dp->snvtDesc; /* Reinitialize */
/* We are now ready to add the extension information */
if(sd->extRec)
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
{
   /* This variable has extension record following node-self-doc */
   /* Add one for each entry in the self id desc part. */
   for (i = 0; i < nvSelfIdCnt; i++)
   {
      /* extPtr has already been set to point to the right place    */
      /* se has already been set to point to dp->snvtExt            */
      /* extPtr is of type (Byte *)                                 */
      *extPtr = dp->snvtExt;
      extPtr += sizeof(SNVTextension);
      if(se->mre)
      {
         *extPtr = dp->maxrEst;
         extPtr++;
      }

      if(se->re)
      {

         *extPtr = dp->rateEst;
         extPtr++;
      }

      if(se->nm)
      {
         /* We need to store the name of the variable. null terminated. */
         nvNameLen = strlen(dp->nvName) + 1;
         if (dp->arrayCnt > 0 && dp->explodeArray)
         {
            int index = i + 1;
            /* We support maximum of 999 array items */
            /* 999 is the maximum index we can represent with 3 digits */
            sprintf(extPtr, "%s_", dp->nvName);
            extPtr += nvNameLen; /* Should now point to null character */
            if (index < 10)
            {
               sprintf(extPtr, "%1d", index); /* Single digit index */
               extPtr++;
            }
            else if (index < 100)
            {
               sprintf(extPtr, "%2d", index); /* Double digit index */
               extPtr += 2;
            }
            else
            {
               sprintf(extPtr, "%3d", index); /* Triple digit index */
               extPtr += 3;
            }
            *extPtr = '\0';
            extPtr++;
         } /* for j loop */
         else
         {
            /* Simple variable. or Array with one entry. */
            strcpy(extPtr, dp->nvName); /* Adds Null ch too */
            extPtr += nvNameLen;
         } /* else */
      } /* if (se->nm) */

      if(se->sd)
      {
         if (dp->nvSdoc == NULL)
         {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
               *extPtr = '\0'; /* nvSdoc is missing */
               extPtr++;
            }
            else
            {
               docLen = strlen(dp->nvSdoc) + 1;
               strcpy(extPtr, dp->nvSdoc);
               extPtr += docLen;
            }
         }

         if(se->nc)
         {
            /* Store the dimension of the variable. Useful only for array
               entered in SNVT that are not exploded. */
            *extPtr = dp->arrayCnt >> 8;   /* High order byte. */
            extPtr++;
            *extPtr = dp->arrayCnt & 0xFF; /* Low order byte */
            extPtr++;
         }
      } /* for */
   } /* if (sd->extRec) */

   /* Restore Alias */
   nmp->snvt.aliasPtr = (AliasField *)extPtr;
   *nmp->snvt.aliasPtr = saveAlias;
   extPtr = extPtr + sizeof(AliasField);

   /* Now extPtr points to the byte following the alias field */

   /* Update nmp->snvt.length */
   /* 6 below refers to the size of the header in snvt structure */
   nmp->snvt.length  = (extPtr - (char *)&nmp->snvt.sb[0]) + 6;

   nmp->nvTableSize += dim;

   return(nmp->nvTableSize - dim); /* Base index for arrays. */
}

/*****************************************************************************
Function:  ProcessNV
Returns:   None
Purpose:   To process an incoming network variable message.
           The message can be
            1. NV Update Message (ACKD, UNACKD, UNACKD_RPT)
            2. NV Poll Message. (REQUEST)
Comments:  In either case, the msg should have at least 2 bytes.
*****************************************************************************/
static void ProcessNV(APPReceiveParam *appReceiveParamPtr,
                      APDU            *apduPtr)
{
   if (appReceiveParamPtr->service == REQUEST)
   {
      ProcessNVPoll(appReceiveParamPtr, apduPtr);
   }
   else
   {
      ProcessNVUpdate(appReceiveParamPtr, apduPtr);
   }
}

/*****************************************************************************
Function:  ProcessNVPoll
Returns:   None
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Purpose:   To process an incoming network variable poll message.
           The service should be a request.
Reference: The Technology Device Data Book Rev 3. Page 9-54
Comments:  The message should have 2 bytes.
           The network variable poll message is normally addressed
           to an output variable. However, we will allow either
           input or output variable to be polled. We simply respond
           with the data value of the matching variable.
           The application layer in reference implementation
           never sends network variable poll messages addressed
           to input variables. A network variable monitor tool
           might do this to get the value of input variables.
           (NVFetch is another way).

           If we are offline, send a response with no data.
           Else
               If we have one matching network variable (primary
               or alias), then send the response with the data.

               If we have two matching primary network variables
               then we ignore this message.

               If we don't have a matching network variable,
               then send a response with no data.

           The response is network variable update message
           with the direction flipped from what was received.

           If we have more than one primary variable with matching
           selector, then we have a problem. The sender is expecting
           only one response from each node receiving the poll message.
           So, it is not clear which one to send. If we have this
           situation, we will ignore this message and not respond
           at all. Note that it is not meaningful to have a primary
           and an alias of that primary to have the same selector.
           If the incoming variable is connected to two different primary
           variables on this node, then the incoming variable
           should not have been polled.
*********************************************************************************/
static void ProcessNVPoll(APPReceiveParam *appReceiveParamPtr,
                          APDU            *apduPtr)
{
   int16          i;
   uint8          nvDirection;
   uint16         selector, thisSelector;
   int16          matchingIndex;
   uint16         matchingPrimaryIndex;
   Queue         *tsaOutQPtr;
   TSASendParam  *tsaSendParamPtr;
   APDU          *apduRespPtr;
   NVStruct      *thisNVStrPtr, *matchingNVStrPtr;
   Boolean        authOK;
   int16          nvAliasTableSize;
   Boolean        noData; /* Should data go out? */

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

   if(appReceiveParamPtr->pduSize != 2)
   {
      /* The message does not have correct size */
      nmp->errorLog = NV_MSG_TOO_SHORT;
      DeQueue(&gp->appInQ);
      return;
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
tsaOutQPtr = &gp->tsaRespQ;

if(QueueFull(tsaOutQPtr))
{
   /* Can't send response yet - try later. */
   return;
}

if (NodeUnConfigured())
{
   /* Ignore this message in this state. See Tech Data Book Rev 3 p9-47. */
   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = TRUE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   EnQueue(tsaOutQPtr);
   DeQueue(&gp->appInQ);
   return;
}

/* Determine selector and nvDirection for variable in the poll message. */
selector    = (apduPtr->code.nv.nvCode << 8) | apduPtr->data[0];
nvDirection = apduPtr->code.nv.nvDir;

noData = TRUE; /* Assume that we will respond with no data */
matchingIndex = -1; /* Initialize to indicate that no match yet. */

/* If application is not running, then we should return with no data */
/* We know that the node is configured at this point */
if(AppPgmRuns())
{
   /* Search for matching network variable. Search both primary
      and alias entries */
   for(i = 0; i < nmp->nvTableSize+nvAliasTableSize; i++)
   {
      thisNVStrPtr     = GetNVStructPtr(i);
      thisSelector =
         (thisNVStrPtr->nvSelectorHi << 8) | thisNVStrPtr->nvSelectorLo;
      if(thisNVStrPtr->nvDirection == nvDirection &&
         thisSelector              == selector)
      {
         if (matchingIndex == -1)
         {
            matchingIndex = i; /* First match. */
         }
         else if (GetPrimaryIndex(matchingIndex) ==
                  GetPrimaryIndex(i))
         {
            /* We have two distinct primary variables with same
               selector. Ignore this message. */
            tsaSendParamPtr               = QueueTail(tsaOutQPtr);
            tsaSendParamPtr->altPathOverride = FALSE;
            tsaSendParamPtr->service      = RESPONSE;
            tsaSendParamPtr->nullResponse = TRUE;
            tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
            EnQueue(tsaOutQPtr);
            DeQueue(&gp->appInQ);
            return;
         }
      }
      else
      {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            continue; /* Skip this entry. Does not match. */
         }
      }
   }

   /* Send the response with either data or nodata. */
   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);
   apduRespPtr->code.nv.nvFlag   = 0x1;
   apduRespPtr->code.nv.nvDir    = 1 - nvDirection; /* Opposite */
   /* Keep the same selector as the one received */
   apduRespPtr->code.nv.nvCode   = apduPtr->code.nv.nvCode;
   apduRespPtr->data[0]          = apduPtr->data[0];

   if (matchingIndex != -1)
   {
      noData               = FALSE;
      matchingPrimaryIndex = GetPrimaryIndex(matchingIndex);
      matchingNVStrPtr     = GetNVStructPtr(matchingIndex);
   }

   if(appReceiveParamPtr->auth ||
      matchingIndex == -1      ||
      !matchingNVStrPtr->nvAuth)
   {
      authOK = TRUE;
   }
   else
   {
      authOK = FALSE;
   }

   if (noData || matchingIndex == -1 || authOK == FALSE)
   {
      /* Send a response with no data */
      tsaSendParamPtr->apduSize = 2;
   }
   else
   {
      /* Send a response with data */
      if (NV_LENGTH(matchingPrimaryIndex) + 2 <= gp->tsaRespBufSize)
      {
         memcpy(&apduRespPtr->data[1],
                NV_ADDRESS(matchingPrimaryIndex),
                NV_LENGTH(matchingPrimaryIndex));
         tsaSendParamPtr->apduSize = 2 + NV_LENGTH(matchingPrimaryIndex);
      }
      else
      {
         tsaSendParamPtr->apduSize = 2;
      }
   }
   EnQueue(tsaOutQPtr);

   /* Message processing completed - remove it from queue */
   DeQueue(&gp->appInQ);
   return;
}

/*****************************************************************************
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Function:  ProcessNVUpdate
Returns:   None
Purpose:   To process an incoming network variable update message.
           The service can be ACKD, UNACKD, UNACKD_RPT.
           The service can be RESPONSE too for poll responses.
Reference: The Technology Device Data Book Rev 3. Page 9-53
Comments:  The message should have > 2 bytes.
           The network variable update message is addressed
           to an input variable.

           If we are unconfigured then discard the message.

           There can be only one variable (primary or alias) with
           matching selector. Once a match is found, break.

           If we update one ore more network variables and we are
           not soft-offline then we don't give NVUpdateOccurs event.
           Otherwise, we do give NVUpdateOccurs to the application program.

           The prefix 'this' is used for local variables of this function
           related to information regarding network variables searched.
*****************************************************************************/
static void ProcessNVUpdate(APPReceiveParam *appReceiveParamPtr,
                            APDU            *apduPtr)
{
   int16          i;
   uint16         dataLength, matchingDataLength;
   uint8          nvDirection;
   uint16         selector, thisSelector;
   int16          matchingIndex;
   uint16         matchingPrimaryIndex;
   NVStruct      *thisNVStrPtr, *matchingNVStrPtr;
   Boolean        authOK;
   int16          nvAliasTableSize;
   uint16         thisDim;
   int16          thisBaseIndex;

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

   if(appReceiveParamPtr->pduSize <= 2)
   {
      /* The message does not have any correct size or data field. */
      nmp->errorLog = NV_MSG_TOO_SHORT;
      DeQueue(&gp->appInQ);
      return;
   }

   if (eep->readOnlyData.nodeState == APPL_UNCNFG)
   {
      /* Ignore this message in this state. See Tech Data Book Rev 3 p9-47. */
      DeQueue(&gp->appInQ);
      return;
   }

   /* Determine selector and nvDirection for variable in the update message. */
   selector    = (apduPtr->code.nv.nvCode << 8) | apduPtr->data[0];
   nvDirection = apduPtr->code.nv.nvDir;
   if (nvDirection == NV_OUTPUT)
   {
      /* Ignore this message */
      nmp->errorLog = NV_UPDATE_ON_OUTPUT_NV;
      DeQueue(&gp->appInQ);
      return;
   }
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
dataLength = appReceiveParamPtr->pduSize - 2; /* data length in message */

/* Go through network input variables looking for a match. Once
   a match is found, update it and break. */
matchingIndex = -1;
for(i = 0; i < nmp->nvTableSize + nvAliasTableSize; i++)
{
   thisNVStrPtr = GetNVStructPtr(i);
   thisSelector =
      (thisNVStrPtr->nvSelectorHi << 8) | thisNVStrPtr->nvSelectorLo;
   if (thisNVStrPtr->nvDirection == NV_OUTPUT)
   {
      continue; /* Skip network output variables */
   }
   if (thisSelector == selector)
   {
      matchingIndex = i;
      break;
   }
}

if (matchingIndex != -1)
{
   /* Need to update the network input variable   */
   /* matchingIndex can be primary or alias index */
   matchingPrimaryIndex = GetPrimaryIndex(matchingIndex);
   matchingDataLength   = NV_LENGTH(matchingPrimaryIndex);
   matchingNVStrPtr     = GetNVStructPtr(matchingPrimaryIndex);

   /* If the data size does not match, don't update. ignore. */
   if (dataLength != matchingDataLength)
   {
      nmp->errorLog = NV_LENGTH_MISMATCH;
      DeQueue(&gp->appInQ);
      return;
   }

   if(appReceiveParamPtr->auth || !matchingNVStrPtr->nvAuth)
   {
      authOK = TRUE;
   }
   else
   {
      authOK = FALSE;
   }

   if (!authOK)
   {
      DeQueue(&gp->appInQ);
      return; /* Skip the update as authentication did not succeed. */
   }

   /* Update the variable */
   if (dataLength > 0 && appReceiveParamPtr->service == RESPONSE)
   {
      /* We have a response to poll message. Update gp->nvInDataStatus flag. */
      gp->nvInDataStatus = SUCCESS;
   }
   memcpy(NV_ADDRESS(matchingPrimaryIndex),
          &apduPtr->data[1],
          dataLength);
   if (AppPgmRuns())
   {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            /* Notify application program only if it is running. */
            gp->nvInAddr.domain = appReceiveParamPtr->srcAddr.domainIndex;
            gp->nvInAddr.flexDomain =
                  (appReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN);
            memcpy(&gp->nvInAddr.srcAddr,
                   &appReceiveParamPtr->srcAddr.subnetAddr,
                   sizeof(gp->nvInAddr.srcAddr));

            switch(appReceiveParamPtr->srcAddr.addressMode)
            {
               case BROADCAST:
                  gp->nvInAddr.format  = 0;
                  break;
               case MULTICAST:
                  gp->nvInAddr.format  = 1;
                  gp->nvInAddr.destAddr.group =
                     appReceiveParamPtr->srcAddr.group;
                  break;
               case SUBNET_NODE:
                  gp->nvInAddr.format  = 2;
                  break;
               case UNIQUE_NODE_ID:
                  gp->nvInAddr.format  = 3;
                  break;
               default:
                  /* should not come here */
                  gp->nvInAddr.format  = 5; /* unknown */
            }
            IsArrayNV(matchingPrimaryIndex, &thisDim, &thisBaseIndex);
            gp->nvArrayIndex = matchingPrimaryIndex - thisBaseIndex;
            REF_IMP_TO_ALT(NV_IN_ADDR, &nv_in_addr);
            REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
            NVUpdateOccurs(thisBaseIndex, gp->nvArrayIndex);
         }
      }

      DeQueue(&gp->appInQ);
      return;
   }


   /*************************************************************************
   Function: PropagateThisIndex
   Returns:  SUCCESS if the index is scheduled.
             FAILURE if the queue is full and hence not scheduled
                     or for sync network output variables, the queue
                     buffer size is not sufficient for this variable.
             INVALID if the index does not correspond to NV_OUTPUT
   Purpose:  To schedule a specific index of a network variable
             (primary or alias), polled or not.
   Comment:
             This function is local to this file and used by API functions
             Propagate, PropagateNV, and PropagateArrayNV.
             The address table entry for NV_OUTPUT can be turnarouud.
             So, if it is UNBOUND, then we check for turnaround field too.

             primaryIndex is passed for efficiency to avoid recomputation.
             nvIndexIn is always valid.
   *************************************************************************/
   static Status PropagateThisIndex(int16 nvIndexIn, int16 primaryIndex)
   {
      int16    *indexPtr;
      Queue    *indexQPtr;
      uint16    addrIndex;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   NVStruct *nvStructPtr;
   uint16    nvAliasTableSize;
   char     *valPtr;
   uint16    bufSize;

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
   nvStructPtr      = GetNVStructPtr(nvIndexIn);
   addrIndex        = nvStructPtr->nvAddrIndex;

   /* If the variable is not output, then we can't propagate. */
   if (nvStructPtr->nvDirection != NV_OUTPUT)
   {
      return(INVALID);
   }

   indexQPtr = &gp->nvOutIndexQ;
   bufSize   = gp->nvOutIndexBufSize;

   if (QueueFull(indexQPtr))
   {
      return(FAILURE); /* Could not schedule all. */
   }

   indexPtr  = QueueTail(indexQPtr);
   *indexPtr = nvIndexIn;
   if (NV_SYNC(primaryIndex))
   {
      /* Copy current value for synchronous variables */
      /* In the queue, the 2 byte index should follow the value */
      valPtr = (char *)(indexPtr + 1);
      if (NV_LENGTH(primaryIndex) <= bufSize - 2)
      {
         memcpy(valPtr, NV_ADDRESS(primaryIndex), NV_LENGTH(primaryIndex));
         EnQueue(indexQPtr);
      }
      else
      {
         return(FAILURE);
      }
   }
   else
   {
      EnQueue(indexQPtr);
   }

   return(SUCCESS);
}

/*************************************************************************
Function: PropagateThisPrimary
Returns:  None
Purpose:  To schedule a specific primary network variable.
          If the variable is bound
          then
            this schedules the primary and any alias entries for this
            primary using PropagateThisIndex.
            If nothing was scheduled
            then
               generate failure completion event.
          else
            generate success completion event.
Comment:  After scheduling the primary and all related alias entries,
          this function will add -1 to the queue to indicate end.
*************************************************************************/
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
void  PropagateThisPrimary(int16 nvIndexIn)
{
   int16    *indexPtr;
   Queue    *indexQPtr;
   NVStruct *nvStructPtr;
   uint16    nvAliasTableSize;
   uint16    count, dim;
   int16     baseIndex;
   int16     j;
   uint16    queueSpace;

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
   nvStructPtr      = GetNVStructPtr(nvIndexIn);

   indexQPtr = &gp->nvOutIndexQ;

   queueSpace = QueueCnt(indexQPtr) - QueueSize(indexQPtr);

   /* Schedule primary network output variables for NVUpdate. */
   if (IsNVBound(nvIndexIn))
   {
      /* We need space for at least 2 entries to schedule.
         i.e we need to reserve one space for -1 at the end. */

      count = 0;
      if (queueSpace > 1 && PropagateThisIndex(nvIndexIn, nvIndexIn) == SUCCESS)
      {
         count++;
         queueSpace--;
      }
      /* Schedule all alias entries that map to this primary entry.
         If queue does not have much space, stop scheduling rest. */
      for (j = nmp->nvTableSize;
           j < nmp->nvTableSize + nvAliasTableSize && queueSpace > 1;
           j++)
      {
         if (GetPrimaryIndex(j) != nvIndexIn)
         {
            continue;
         }
         if (PropagateThisIndex(j, nvIndexIn) == SUCCESS)
         {
            count++;
            queueSpace--;
         }
      }
      if (count == 0)
      {
         IsArrayNV(nvIndexIn, &dim, &baseIndex);
         gp->nvArrayIndex = nvIndexIn - baseIndex;
         REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
         NVUpdateCompletes(FAILURE, baseIndex, gp->nvArrayIndex);
      }
      else
      {
         /* Schedule a -1 to indicate end of indices for this primary. */
         /* There should be at least one space left in queue */
         indexPtr  = QueueTail(indexQPtr);
         *indexPtr = -1;
         EnQueue(indexQPtr);
      }
   }
   else
   {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      IsArrayNV(nvIndexIn, &dim, &baseIndex);
      gp->nvArrayIndex = nvIndexIn - baseIndex;
      REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
      NVUpdateCompletes(SUCCESS, baseIndex, gp->nvArrayIndex); /* Not bound. */
   }
}


/*****************************************************************************
Function:   Propagate
Returns:    None
Purpose:    To propagate all output network variables.
Comment:    This function is called by the application program to propagate
            all output network variables (declared as polled or not).

            Schedule all NV_OUTPUT indices with a valid address table
            entry regardless of whether it is primary or alias.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call SendVar function to actually send out
            NV Updates messages. If a network output variable is not
            bound, then there is no need to schedule it.

            The addressing information is implicit. If the network
            output variables does not correspond to an address table
            entry, then it can't be scheduled as we don't know how to
            generate the destination address.

            It is possible that not all variables can be scheduled
            due to space limitation in the queues. For guranteed
            scheduling of all possible network ouput variables, the
            nv output queue should be large enough.
*****************************************************************************/
void  Propagate(void)
{
   int16     i;

   /* Schedule primary network output variables. */
   for (i = 0; i < nmp->nvTableSize;  i++)
   {
      PropagateThisPrimary(i);
   }
}

/*****************************************************************************
Function:   PropagateNV
Returns:    None
Purpose:    To propagate a specific output network variable (simple or array)
Comment:    This function is called by the application program to propagate
            a specific network output variable (declared as polled or not).
            If the index corresponds to the first item of an array,
            then all items of the array are propagated.

            If the index does not correspond to first item of an array.
            only that index is propagated.

            The variable must be a primary variable. Alias entries that
            map to this primary (for array, any of the entries), that
            are bound to some valid address table entry. are also
            scheduled.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            APPSend will call SendVar function to actually send out
            NV Update messages.

            There is no gurantee that all possible variables are scheduled
            due to space limitation in the schedule queue. To ensure that
            this does not happen, the queue size for the scheduling should
            be made larger.

            The addressing information is implicit. If the network
            output variables does not correspond to an address table
            entry, then it can't be scheduled as we don't know how to
            generate the destination address.
*******************************************************************************/
void  PropagateNV(int16 nvIndexIn)
{
   uint16    dim;
   int16     baseIndex;
   int16     i;

   if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize )
   {
      return; /* Must be a valid primary index. */
   }


   IsArrayNV(nvIndexIn, &dim, &baseIndex);
   if (nvIndexIn != baseIndex)
   {
      dim = 1; /* nvIndexIn is not the first item of array */
   }

   /* Scheduled one or more (for array) primary indices. */
   /* For array, nvIndexIn = baseIndex */
   for (i = nvIndexIn; i < nvIndexIn + dim; i++)
   {
      PropagateThisPrimary(i);
   }
}


/*******************************************************************************
Function:   PropagateArrayNV
Returns:    None
Purpose:    To propagate a specific element of an array output network
            variable or any other network variable (i.e non-array nv).
Comment:    This function is called by the application program to propagate
            a specific item of an array network output variable
            (declared as polled or not) or a simple network variable.

            Once the primary is scheduled, we also schedule alias
            entries that map to this primary to make sure that we
            reach all possible input connections.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call SendVar function to actually send out
            NV Update messages.

            The addressing information is implicit. If the network
            input variables does not correspond to an address table
            entry, then it can be scheduled as we don't know how to
            generate the destination address.

            If arrayNVIndexIn corresponds to the baseindex of an array
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            network variable then indexIn is used to compute the specific
            item of the array. Otherwise indexIn is set to 0 so that this
            function will propagate the given index only (a specific array
            item or a simple network variable).
*****************************************************************************/
void  PropagateArrayNV(int16 arrayNVIndexIn, int16 indexIn)
{
   uint16    dim;
   int16     baseIndex;
   int16     nvIndex;

   if (arrayNVIndexIn < 0 || arrayNVIndexIn >= nmp->nvTableSize)
   {
      return; /* Invalid index. */
   }

   /* if arrayNVIndexIn is not an array variable, then dim is set to 1
      and baseIndex is set to arrayNVIndexIn by IsArrayNV function */
   if (!IsArrayNV(arrayNVIndexIn, &dim, &baseIndex) )
   {
      indexIn = 0; /* Simple network variable */
   }
   else if (baseIndex != arrayNVIndexIn)
   {
      indexIn = 0; /* Any other array item. don't use indexIn passed. */
   }

   if (indexIn < 0 || indexIn >= dim)
   {
      return; /* Invalid index */
   }

   nvIndex = arrayNVIndexIn + indexIn;
   PropagateThisPrimary(nvIndex);
}

/*****************************************************************************
Function:  SendVar
Returns:   None.
Purpose:   Generates NV Update message for the given index.
           The given index can be either primary or alias.
           For sync network output variables, we are also
           given the corresponding value. For nonsync variables,
           valPtr is null and current value is used.

           If the network output variable has nvTurnaround on,
           then we need to search for a network input variable with
           matching selector number. Once found, we update it, and
           send NVUpdateOccurs event to the application program.
           Note that we search through both primary and alias
           entries for the turnaround. If a primary is connected to this
           output variables, we will find it eventually either direcly
           or indirectly (via alias).

Comments:
           We need to handle only the given index. Propagate functions
           take care of scheduling all needed alias indices properly.
           Implicit NV Updates are handleded through Propagate.

           It is possible that variables are scheduled, but by the time
           they are processed in this function, the application is
           offline or the node is unconfigured. So, check for this
           too before updating network variables for turnaround.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
          Also, it is possible that some attributes of the variable
          has been changed by a network management tool after the
          index was scheduled.

          If the index is -1, then it represents the end of scheduling for a
          primary. Generate a message to transport layer with a special tag
          that will be recognized and sent back immediately in the
          indication to application layer.
*************************************************************************/
static void SendVar()
{
   Queue          *indexQPtr;
   int16           i;              /* For loop. */
   uint16          bufSize;        /* bufSize for the target queue */
   int16           nvIndex;
   int16           primaryIndex;   /* For nvIndex. */
   uint16          nvLength;       /* For nvIndex. */
   uint16          selector;       /* For nvIndex. */
   Byte           *nvPtr;          /* For nvIndex. Points to storage. */
   uint16          dimIn;          /* For input network variable */
   int16           primaryIndexIn; /* For input network variable */
   int16           baseIndexIn;    /* For input network variable */
   uint16          nvLengthIn;     /* For input network variable */
   Byte           *nvPtrIn;        /* For input network variable */
   uint16          selectorIn;     /* For input network variable */
   Queue          *nwOutQPtr, *tsaOutQPtr;
   TSASendParam   *tsaSendParamPtr;
   NWSendParam    *nwSendParamPtr;
   APDU           *apduPtr;
   uint16          nvAliasTableSize;
   NVStruct       *nvStrPtr, *nvStrPtrIn;
   uint16          addrIndex;
   AddrTableEntry *ap;
   Boolean         turnAroundOnly;  /* does not mean turnaround for sure. Means
                                       that the variable does not have addr table
                                       entry or the address table entry is unbound
                                       or it is turnaround entry. */
   Boolean         foundMatchingInputVar;
   int16          *indexPtr;
   char           *valPtr;

   indexQPtr = &gp->nvOutIndexQ;

   if (!gp->nvOutCanSchedule || QueueEmpty(indexQPtr))
   {
      return; /* Nothing to do. */
   }
   indexPtr  = QueueHead(indexQPtr);
   nvIndex   = *indexPtr;
   valPtr    = (char *)(indexPtr + 1);

   /* If the node enters unconfigued state before processing this nv update
      we do not want to schedule these indices. We simply do nothing and wait
      for the node to go configured. */
   if (eep->readOnlyData.nodeState == APPL_UNCNFG)
   {
      return;
   }

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

   /* nvIndexIn > 0 => nv variable. nvIndex == -1 => end of scheduling */
   if (nvIndex >= 0)
   {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* Make nvStrPtr point to the right NVStruct for nvIndexIn */
   nvStrPtr     = GetNVStructPtr(nvIndex);
   primaryIndex = GetPrimaryIndex(nvIndex);
   nvPtr        = NV_ADDRESS(primaryIndex);
   nvLength     = NV_LENGTH(primaryIndex);
   selector     = (nvStrPtr->nvSelectorHi << 8) | nvStrPtr->nvSelectorLo;
   addrIndex    = nvStrPtr->nvAddrIndex;
   /* The variable is turnaround only if addrIndex is 0xF or the address
      table entry is unbound (turnaround or not). */
   /* *** START INFORMATIVE - Unbound Network Variable */
   /* It is acceptable to require that turnaround NVs have an address table
    * entry assigned.  This entry may be used for determining retry counts
    * and tx_timer values for retrying in the event of resource problems such
    * as input buffer inavailability.  Network management tools are expected
    * to assign an address table entry even if using unackd service. */
   /* *** END INFORMATIVE - Unbound Network Variable */
   turnAroundOnly  = addrIndex == 0xF ||
                     eep->addrTable[addrIndex].addrFormat == UNBOUND;

   if(nvStrPtr->nvPriority)
   {
      tsaOutQPtr = &gp->tsaOutPriQ;
      nwOutQPtr  = &gp->nwOutPriQ;
      if (nvStrPtr->nvService == UNACKD)
      {
         bufSize = gp->nwOutPriBufSize;
      }
      else
      {
         bufSize = gp->tsaOutPriBufSize;
      }
   }
   else
   {
      tsaOutQPtr = &gp->tsaOutQ;
      nwOutQPtr  = &gp->nwOutQ;
      if (nvStrPtr->nvService == UNACKD)
      {
         bufSize = gp->nwOutBufSize;
      }
      else
      {
         bufSize = gp->tsaOutBufSize;
      }
   }
}
else
{
   /* Let us use tsaOutQ for the special message */
   tsaOutQPtr = &gp->tsaOutQ;
}

if (nvIndex == -1 || !turnAroundOnly)
{
   /* We need to make sure that we have space in transport or network
      layer. If there is no space, we should return without doing any
      processing.
      NV variable messages can be ACK, UNACK, UNACKD_RPT */
   if (nvIndex == -1)
   {
      if (QueueFull(tsaOutQPtr))
      {
         return;
      }
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      }
      else if(nvStrPtr->nvService == UNACKD && QueueFull(nwOutQPtr))
      {
         return;
      }
      else if(nvStrPtr->nvService != UNACKD && QueueFull(tsaOutQPtr))
      {
         return;
      }
   }

   if (nvIndex == -1)
   {
      /* Form a message with a special tag to transport layer. */
      tsaSendParamPtr              = QueueTail(tsaOutQPtr);
      tsaSendParamPtr->altPathOverride = FALSE;
      tsaSendParamPtr->service     = ACKD;
      tsaSendParamPtr->tag         = NV_UPDATE_LAST_TAG_VALUE;
      tsaSendParamPtr->apduSize    = 0;
      EnQueue(tsaOutQPtr);
      gp->nvOutCanSchedule = FALSE; /* Only one at a time. */
      DeQueue(indexQPtr);
      return;
   }


   /* If the variable is flagged as turnaround, then we look for first
      network input variable with matching selector number. Once found,
      we update that input variable, and then send NVUpdateOccurs event
      to the application program. We break as soon as first match is found. */
   if (nvStrPtr->nvTurnaround)
   {
      foundMatchingInputVar = FALSE;
      for(i = 0; i < nmp->nvTableSize+nvAliasTableSize; i++)
      {
         nvStrPtrIn = GetNVStructPtr(i);
         selectorIn =
               (nvStrPtrIn->nvSelectorHi << 8) | nvStrPtrIn->nvSelectorLo;
         /* If this variable is not input or does not have matching selector,
            then skip this entry */
         if (nvStrPtrIn->nvDirection != NV_INPUT ||
             selector != selectorIn)
         {
            continue; /* Not a matching entry */
         }
         /* Found a matching turnaroud entry for nvIndexIn */
         primaryIndexIn = GetPrimaryIndex(i);
         nvPtrIn       = NV_ADDRESS(primaryIndexIn);
         nvLengthIn    = NV_LENGTH(primaryIndexIn);
         if (nvLength != nvLengthIn)
         {
            break; /* selector matches but length does not */
         }
         memcpy(nvPtrIn, nvPtr, nvLength);
         foundMatchingInputVar = TRUE;
         /* Notify application if it is running */
         if (AppPgmRuns())
         {
            IsArrayNV(primaryIndexIn, &dimIn, &baseIndexIn);

            gp->nvInAddr.format = 4; /* TURNAROUND */
            memset(&gp->nvInAddr.srcAddr, 0, sizeof(SubnetAddress));
            gp->nvInAddr.domain = 0; /* Not relevant */
            gp->nvArrayIndex    = primaryIndexIn - baseIndexIn;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            REF_IMP_TO_ALT(NV_IN_ADDR, &nv_in_addr);
            REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
            NVUpdateOccurs(baseIndexIn, gp->nvArrayIndex);
         }

         break; /* break after first match */
      } /* for */
   } /* if */

   if (turnAroundOnly)
   {
      /* Completion event is given in HandleMsgCompletion function */
      /* Since there is nothing actually scheduled, no need to clear
         gp->nvOutCanSchedule flag. */
      /* Since this index does not go through HandleMsgCompletion, we need
         to set gp->nvOutIndex here. This is to take care of the case when
         a variable is turndaround only with no alias. In this case this index
         is followed by -1 in the queue. Hence, the gp->nvOutIndex would never
         be initialized when HandleMsgCompletion gets NV_UPDATE_LAST_TAG_VALUE.
         Explicit initialization here will fix the problem. */
      gp->nvOutIndex = GetPrimaryIndex(nvIndex);
      DeQueue(indexQPtr);
      return;
   }

   gp->nvOutCanSchedule = FALSE; /* Only one index at a time */
   DeQueue(indexQPtr);

   /* Build and send network variable update message. */
   ap = AccessAddress(addrIndex); /* ap can't be null. */
   /* Fail if we don't have sufficient space in the target queue. */
   if (2 + nvLength > bufSize)
   {
      /* Discard this index as the space is not sufficient */
      return;
   }

   if (nvStrPtr->nvService != UNACKD)
   {
      /* The message should go to the transport layer */
      tsaSendParamPtr              = QueueTail(tsaOutQPtr);
      tsaSendParamPtr->altPathOverride = FALSE;
      tsaSendParamPtr->domainIndex = COMPUTE_DOMAIN_INDEX;
      *(AddrTableEntry *)(&tsaSendParamPtr->destAddr) = *ap;
      tsaSendParamPtr->service     = nvStrPtr->nvService;
      tsaSendParamPtr->auth        = nvStrPtr->nvAuth;
      /* See app.h for information on tag usage. */
      tsaSendParamPtr->tag         = GET_NV_UPDATE_TAG(primaryIndex);
      apduPtr                      = (APDU *)(tsaSendParamPtr + 1);
      apduPtr->code.nv.nvFlag      = 0x1;
      apduPtr->code.nv.nvDir       = NV_INPUT;
      apduPtr->code.nv.nvCode      = nvStrPtr->nvSelectorHi;
      apduPtr->data[0]             = nvStrPtr->nvSelectorLo;
      tsaSendParamPtr->apduSize    = 2 + nvLength;
      if (NV_SYNC(primaryIndex))
      {
         /* Send value given */
         memcpy(&apduPtr->data[1], valPtr, nvLength);
      }
      else
      {
         /* Send the most current value */
         memcpy(&apduPtr->data[1], nvPtr, nvLength);
      }
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      EnQueue(tsaOutQPtr);
      return;
   }

   /* The message is for the network layer */
   nwSendParamPtr = QueueTail(nwOutQPtr);
   nwSendParamPtr->dropIfUnconfigured = TRUE;
   nwSendParamPtr->tag = GET_NV_UPDATE_TAG(primaryIndex);

   switch(ap->addrFormat)
   {
      case SUBNET_NODE:
         nwSendParamPtr->destAddr.addressMode = SUBNET_NODE;
         nwSendParamPtr->destAddr.domainIndex =
                     ap->snodeEntry.domainIndex;
         nwSendParamPtr->destAddr.addr.addr2a.subnet =
                     ap->snodeEntry.subnetID;
         nwSendParamPtr->destAddr.addr.addr2a.node   =
                     ap->snodeEntry.node;
         break;
      case BROADCAST:
         nwSendParamPtr->destAddr.addressMode = BROADCAST;
         nwSendParamPtr->destAddr.domainIndex =
                  ap->bcastEntry.domainIndex;
         nwSendParamPtr->destAddr.addr.addr0 =
                  ap->bcastEntry.subnetID;
         break;

      default:
         /* Since the address table entry can't be unbound or turnaround,
            we must have group entry */
         nwSendParamPtr->destAddr.addressMode = MULTICAST;
         nwSendParamPtr->destAddr.domainIndex =
                  ap->groupEntry.domainIndex;
         nwSendParamPtr->destAddr.addr.addr1 =
                  ap->groupEntry.groupID;
   } /* switch */

   nwSendParamPtr->pduType     = APDU_TYPE;
   nwSendParamPtr->deltaBL     = 0; /* No ack generated */
   nwSendParamPtr->altPath     = FALSE;
   nwSendParamPtr->pduSize     = 2 + nvLength;
   apduPtr                     = (APDU *)(nwSendParamPtr + 1);
   apduPtr->code.nv.nvFlag     = 0x1;
   apduPtr->code.nv.nvDir      = NV_INPUT;
   apduPtr->code.nv.nvCode     = nvStrPtr->nvSelectorHi;
   apduPtr->data[0]            = nvStrPtr->nvSelectorLo;
   if (NV_SYNC(primaryIndex))
   {
      /* Send value given. */
      memcpy(&apduPtr->data[1], valPtr, nvLength);
   }
   else
   {
      /* Send the most current value. */
      memcpy(&apduPtr->data[1], NV_ADDRESS(primaryIndex), nvLength);
   }
   EnQueue(nwOutQPtr);

   return;
}

/*****************************************************************************
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Function: PollThisIndex
Returns:  SUCCESS if the index is scheduled.
          FAILURE if the queue is full and hence not scheduled.
          INVALID if the index does not correspond to NV_INPUT.
Purpose:  To schedule a specific index of a network variable
          (primary or alias).
Comment:
          This function is local to this file and used by API functions
          Poll, PollNV, and PollArrayNV.
          nvIndexIn is always valid.
*************************************************************************/
static Status PollThisIndex(int16 nvIndexIn)
{
   int16    *indexPtr;
   Queue    *indexQPtr;
   uint16    addrIndex;
   NVStruct *nvStructPtr;
   uint16    nvAliasTableSize;

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
   nvStructPtr      = GetNVStructPtr(nvIndexIn);
   addrIndex        = nvStructPtr->nvAddrIndex;
   if (nvStructPtr->nvDirection != NV_INPUT)
   {
      return(INVALID);
   }

   indexQPtr = &gp->nvInIndexQ;

   if (QueueFull(indexQPtr))
   {
      return(FAILURE); /* Could not schedule all. */
   }
   indexPtr  = QueueTail(indexQPtr);
   *indexPtr = nvIndexIn;
   EnQueue(indexQPtr);
   return(SUCCESS);
}

/*************************************************************************
Function: PollThisPrimary
Returns:  None
Purpose:  To schedule a specific primary network variable.
          If the variable is bound
          then
            this schedules the primary and any alias entries for this
            primary using PollThisIndex.
            If nothing was scheduled
            then
               generate failure completion event.
          else
            generate success completion event.
Comment:  None.
*************************************************************************/
void  PollThisPrimary(int16 nvIndexIn)
{
   int16    *indexPtr;
   Queue    *indexQPtr;
   NVStruct *nvStructPtr;
   uint16    nvAliasTableSize;
   uint16    count, dim;
   int16     baseIndex;
   int16     j;
   uint16    queueSpace;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
   nvStructPtr      = GetNVStructPtr(nvIndexIn);

   indexQPtr = &gp->nvInIndexQ;

   queueSpace = QueueCnt(indexQPtr) - QueueSize(indexQPtr);

   /* Schedule primary network input variables for poll */
   if (IsNVBound(nvIndexIn))
   {
      /* We need space for at least 2 entries to schedule.
      i.e we need to reserve one space for -1 at the end. */
      count = 0;
      if (queueSpace > 1 && PollThisIndex(nvIndexIn) == SUCCESS)
      {
         count++;
         queueSpace--;
      }
      /* Schedule all alias entries that map to this primary entry.
         If queue does not have much space, stop scheduling rest. */
      for (j = nmp->nvTableSize;
           j < nmp->nvTableSize + nvAliasTableSize && queueSpace > 1;
           j++)
      {
         if (GetPrimaryIndex(j) != nvIndexIn)
         {
            continue;
         }
         if (PollThisIndex(j) == SUCCESS)
         {
            count++;
            queueSpace--;
         }
      }
      if (count == 0)
      {
         IsArrayNV(nvIndexIn, &dim, &baseIndex);
         gp->nvArrayIndex = nvIndexIn - baseIndex;
         REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
         NVUpdateCompletes(FAILURE, baseIndex, gp->nvArrayIndex);
      }
      else
      {
         /* Schedule -1 to indicate end of indices for this primary. */
         /* There should be at least one space left in queue */
         indexPtr  = QueueTail(indexQPtr);
         *indexPtr = -1;
         EnQueue(indexQPtr);
      }
   }
   else
   {
      IsArrayNV(nvIndexIn, &dim, &baseIndex);
      gp->nvArrayIndex = nvIndexIn - baseIndex;
      REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
      NVUpdateCompletes(SUCCESS, baseIndex, gp->nvArrayIndex); /* Not bound. */
   }
}

/*****************************************************************************
Function:   Poll
Returns:    None
Purpose:    To poll all input network variables.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Comment:    This function is called by the application program to poll
            all network input variables.

            Schedule all NV_INPUT indices with a valid address table
            entry regardless of whether it is primary or alias.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call PollVar function to actually send out
            polling requests.

            It is possible that not all variables can be scheduled
            due to space limitation in the queues. For guranteed
            scheduling of all possible network input variables, the
            polling queue should be large enough.

            NVUpdateCompletes for input variables means that they
            are completion indication of corresponding poll requests.
*************************************************************************/
void  Poll(void)
{
   int16     i;

   /* Schedule primary network input variables for poll */
   for (i = 0; i < nmp->nvTableSize;  i++)
   {
      PollThisPrimary(i);
   }
}

/*************************************************************************
Function:   PollNV
Returns:    None
Purpose:    To poll a specific input network variable (simple or array)
Comment:    This function is called by the application program to poll
            a specific network input variable.
            If the index corresponds to the first item of an array,
            then all items of the array are polled.

            If the index does not correspond to first item of an array.
            only that index is polled.

            The variable must be a primary variable. All alias entries
            for this primary are also scheduled.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call PollVar function to actually send out
            polling requests.

            There is no gurantee that all possible variables are scheduled
            due to space limitation in the schedule queue. To ensure that
            this does not happen, the queue size for the scheduling should
            be made larger.

            The addressing information is implicit. If the network
            input variables does not correspond to an address table
            entry, then it can't be scheduled as we don't know how to
            generate the destination address.

            NVUpdateCompletes for input variables means that they
            are completion indication of corresponding poll requests.
*************************************************************************/
void  PollNV(int16 nvIndexIn)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
{
   uint16    dim;
   int16     baseIndex;
   int16     i;

   if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize )
   {
      return; /* Must be a valid primary index. */
   }


   IsArrayNV(nvIndexIn, &dim, &baseIndex);
   if (nvIndexIn != baseIndex)
   {
      dim = 1; /* nvIndexIn is not the first item of array */
   }

   /* Scheduled one or more (for array) primary indices. */
   /* For array, nvIndexIn = baseIndex */
   for (i = nvIndexIn; i < nvIndexIn + dim; i++)
   {
      PollThisPrimary(i);
   }
}

/*************************************************************************
Function:   PollArrayNV
Returns:    None
Purpose:    To poll a specific element of an array input network variable
            or any other network variable (i.e non-array).
Comment:    This function is called by the application program to poll
            a specific item of an array network input variable or any
            other network variable (non-array).

            Once the primary is schedules, we also schedule alias
            entries that map to this primary to make sure that we
            reach all possible output connections.

            This function only schedules these variables by placing
            them in a queue for later processing by APPSend function.
            APPSend will call PollVar function to actually send out
            polling requests.

            The addressing information is implicit. If the network
            input variables does not correspond to an address table
            entry, then it can be scheduled as we don't know how to
            generate the destination address.

            If arrayNVIndexIn corresponds to the base index of an array,
            the indexIn is used to get the specific item of the array to
            be polled. Othewise, indexIn is set to 0 so that any other
            array item or any other network variable can be passed.

            NVUpdateCompletes for input variables means that they
            are completion indication of corresponding poll requests.
*************************************************************************/
void  PollArrayNV(int16 arrayNVIndexIn, int16 indexIn)
{
   uint16    dim;
   int16     baseIndex;
   int16     nvIndex;


   if (arrayNVIndexIn < 0 || arrayNVIndexIn >= nmp->nvTableSize)
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   {
      return; /* Invalid index. */
   }

   /* if arrayNVIndexIn is not an array variable, then dim is set to 1
      and baseIndex is set to arrayNVIndexIn by IsArrayNV function */
   if (!IsArrayNV(arrayNVIndexIn, &dim, &baseIndex))
   {
      indexIn = 0; /* Simple network variable */
   }
   else if (baseIndex != arrayNVIndexIn)
   {
      indexIn = 0; /* Any other array item. don't use indexIn passed. */
   }

   if (indexIn < 0 || indexIn >= dim)
   {
      return; /* Invalid index */
   }

   nvIndex = arrayNVIndexIn + indexIn;
   PollThisPrimary(nvIndex);
}

/*****************************************************************************
Function:  PollVar
Returns:   None
Reference: Technology Device Data Book Rev 3. page 9-54.
Purpose:   Issues a poll message for a single network input variable.
           Sends NV Poll message for the connected output variables.

           If there exists a unique turnaround output, we update
           the network variable directly and send NVUpdateOccurs event
           to the application program. Thus, updates from externally
           connected output variables will arrive later.

Comments:  Note that this return value does not mean that the
           poll transaction is complete. It only means that the
           polling request has been delivered to the session layer.

           nvIndexIn must be good as it was previously scheduled.
           nvIndexIn can be either primary or alias.

           Do not check turnaround entries unless the session layer
           has space in the request queue for scheduling the poll.
           Otherwise, we may do the turnaround but stuck with not
           being able send the poll request. If we keep the entry
           in the queue after turnaround entries are processed, we
           may end up with double processing of turnaround entries.

           The NV Variable Poll message must be Request. So, use
           session layer.
*****************************************************************************/
static void PollVar(void)
{
   Queue          *indexQPtr;
   int16           nvIndex;
   int16          *indexPtr;
   int16           i;              /* For loop. */
   int16           primaryIndex;   /* For nvIndex. */
   uint16          nvLength;       /* For nvIndex. */
   uint16          dim;            /* For nvIndex, if it is array */
   int16           baseIndex;      /* For nvIndex, if it is array */
   uint16          selector;       /* For nvIndex. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Byte           *nvPtr;           /* For nvIndex. Points to storage. */
int16           primaryIndexOut; /* For output network variable */
uint16          nvLengthOut;     /* For output network variable */
Byte           *nvPtrOut;        /* For output network variable */
uint16          selectorOut;     /* For output network variable */
Queue          *tsaOutQPtr;
TSASendParam   *tsaSendParamPtr;
APDU           *apduPtr;
uint16          nvAliasTableSize;
NVStruct       *nvStrPtr, *nvStrPtrOut;
uint16          addrIndex, addrIndexOut;
AddrTableEntry *ap;
Boolean         turnAroundOnly;  /* does not mean turnaround for sure. Means
                                    that the variable does not have addr table
                                    entry or the address table entry is unbound
                                    or it is turnaround entry. */
Boolean         foundMatchingOutputVar;
int16           matchingIndexOut;

indexQPtr = &gp->nvInIndexQ;

if (!gp->nvInCanSchedule || QueueEmpty(indexQPtr))
{
   return;
}
indexPtr  = QueueHead(indexQPtr);
nvIndex = *indexPtr;

/* If the node enters unconfigued state before processing this nv poll
   we do not want to schedule these indices. We simply do nothing and wait
   for the node to go configured. */
if (eep->readOnlyData.nodeState == APPL_UNCNFG)
{
   return;
}

nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

if (nvIndex >= 0)
{
   /* Make nvStrPtr point to the right NVStruct for nvIndex */
   nvStrPtr     = GetNVStructPtr(nvIndex);
   /* START INFORMATIVE - Network Variable Alias Priority */
   /* The bracketed code in conjunction with other code in this implementation
    * is structured in a way that allows network variable aliases
    * to have a different priority attribute than the primary and for that
    * difference in priority to be honored.  It is acceptable, however, to send
    * all aliases using the same priority attribute as that of the primary. */
   if(nvStrPtr->nvPriority)
   {
      tsaOutQPtr = &gp->tsaOutPriQ;
   }
   else
   {
      tsaOutQPtr = &gp->tsaOutQ;
   }
   /* END INFORMATIVE - Network Variable Alias Priority */

   primaryIndex = GetPrimaryIndex(nvIndex);
   nvPtr        = NV_ADDRESS(primaryIndex);
   nvLength     = NV_LENGTH(primaryIndex);
   selector     = (nvStrPtr->nvSelectorHi << 8) | nvStrPtr->nvSelectorLo;
   addrIndex    = nvStrPtr->nvAddrIndex;
   /* The variable is turnaround only if addrIndex is 0xF or it is unbound
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      (turnaround or not) */
   turnAroundOnly  = (addrIndex == 0xF) ||
                     (eep->addrTable[addrIndex].addrFormat == UNBOUND);
}
else
{
   tsaOutQPtr = &gp->tsaOutQ;
}
/* If the address table entry is turnaround only, then we don't send out
   any nv poll messages and hence we don't need to check the queue
   for space availability */
if(nvIndex == -1 || !turnAroundOnly)
{
   if (QueueFull(tsaOutQPtr))
   {
      /* Can't send request yet - try later. Don't even try turnaround. */
      return;
   }
}

if (nvIndex == -1)
{
   /* Form a message with a special tag to transport layer. */
   tsaSendParamPtr              = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service     = ACKD;
   tsaSendParamPtr->tag         = NV_POLL_LAST_TAG_VALUE;
   tsaSendParamPtr->apduSize    = 0;
   EnQueue(tsaOutQPtr);
   gp->nvInCanSchedule = FALSE; /* Only one at a time. */
   DeQueue(indexQPtr);
   return;
}

/* If nvIndex is part of an array, we need to get the dimension and
   the baseIndex so that we can report nvInAddr and nvArrayIndex
   properly, in case there is a turn around connected. Alias entries
   can't be used to determine whether it is array. Use primaryIndex.
   We will use baseIndex for NVUpdateOccurs event */
IsArrayNV(primaryIndex, &dim, &baseIndex);

/* If the variable is flagged as turnaround, then we look for first
   network output variables with matching selector number. Once found,
   we update this input variable, and then send NVUpdateOccurs event
   to the application program. */
if (nvStrPtr->nvTurnaround)
{
   foundMatchingOutputVar = FALSE;
   matchingIndexOut = -1;
   for(i = 0; i < nmp->nvTableSize+nvAliasTableSize; i++)
   {
      nvStrPtrOut  = GetNVStructPtr(i);
      addrIndexOut = nvStrPtrOut->nvAddrIndex;
      /* Skip input network variables */
      if (nvStrPtrOut->nvDirection == NV_INPUT)
      {
         continue;
      }
      /* It is a network output variable. Match selector */
      selectorOut =
         (nvStrPtrOut->nvSelectorHi << 8) | nvStrPtrOut->nvSelectorLo;
      if (selector != selectorOut)
      {
         continue; /* Selector does not match */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
         }
         /* Found a matching output variable for nvIndex */
         matchingIndexOut = i;
         break;
      }

      if (matchingIndexOut != -1)
      {
         primaryIndexOut = GetPrimaryIndex(matchingIndexOut);
         nvPtrOut        = NV_ADDRESS(primaryIndexOut);
         nvLengthOut     = NV_LENGTH(primaryIndexOut);

         /* Update the input variable provided the length matches */
         if (nvLength == nvLengthOut)
         {
            /* We will skip authentication check as it is turnaround */
            memcpy(nvPtr, nvPtrOut, nvLength);
            foundMatchingOutputVar = TRUE;

            /* turnaround updates are ignored for valid data check for polls.
               The only time it is ignored is if the nv is turnaround only.
               So, do not update gp->nvInDataStatus flag here. */

            /* Notify application program if it is running. */
            if (AppPgmRuns())
            {
               gp->nvInAddr.format = 4; /* TURNAROUND */
               memset(&gp->nvInAddr.srcAddr, 0, sizeof(SubnetAddress));
               gp->nvInAddr.domain = 0; /* Not relevant */
               /* gp->nvArrayIndex is 0 for simple var */
               gp->nvArrayIndex    = primaryIndex - baseIndex;
               REF_IMP_TO_ALT(NV_IN_ADDR, &nv_in_addr);
               REF_IMP_TO_ALT(NV_ARRAY_INDEX, &nv_array_index);
               /* same as primaryIndex for simple var */
               NVUpdateOccurs(baseIndex, gp->nvArrayIndex);
            }
         }
       }
   }

   if (turnAroundOnly)
   {
      /* NV completion event is given in HandleMsgCompletion. */
      /* Don't clear the nvInCanSchedule flag as we can continue */
      /* Since this index does not go through HandleMsgCompletion, we need
         to set gp->nvInIndex here. This is to take care of the case when
         a variable is turndaround only with no alias. In this case, this index
         is followed by -1 in the queue. Hence, the gp->nvInIndex would never
         be initialized when HandleMsgCompletion gets NV_POLL_LAST_TAG_VALUE.
         Explicit initialization here will fix the problem. */
      gp->nvInIndex = GetPrimaryIndex(nvIndex);
      if (nvStrPtr->nvTurnaround && matchingIndexOut != -1)
      {
         /* We did find a matching output variable and updated the polled */
         /* variable */
         /* Note that even if one of the indices (primary or alias) is turnaround
            only, this flag is set to true. */
         gp->nvInDataStatus = SUCCESS; /* to enable poll to succeed */
      }
      DeQueue(indexQPtr);
      return;
   }

   gp->nvInCanSchedule = FALSE;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   DeQueue(indexQPtr);

   /* Build and send netvar poll message. It is a REQUEST message. */
   ap = AccessAddress(addrIndex); /* ap can't be null */

   tsaSendParamPtr              = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->domainIndex = COMPUTE_DOMAIN_INDEX;
   *(AddrTableEntry *)(&tsaSendParamPtr->destAddr) = *ap;
   tsaSendParamPtr->service     = REQUEST; /* Poll Message */
   tsaSendParamPtr->auth        = nvStrPtr->nvAuth;
   /* See app.h for details on tag usage. */
   tsaSendParamPtr->tag         = GET_NV_POLL_TAG(primaryIndex);
   apduPtr                      = (APDU *)(tsaSendParamPtr + 1);
   apduPtr->code.nv.nvFlag      = 0x1;
   apduPtr->code.nv.nvDir       = NV_OUTPUT;
   apduPtr->code.nv.nvCode      = nvStrPtr->nvSelectorHi;
   apduPtr->data[0]             = nvStrPtr->nvSelectorLo;
   tsaSendParamPtr->apduSize    = 2;

   EnQueue(tsaOutQPtr);
   return;
}

/*****************************************************************************
Function: NewMsgTag
Returns:  A new message tag of the requested type (bindable or non-bindable).
Purpose:  To allocate message tags for use by application program.
Comment:  Reference implementation does not support rate information
          for tags.
*****************************************************************************/
MsgTag NewMsgTag(BindNoBind bindStatusIn)
{
   if (bindStatusIn == BIND)
   {
      if (gp->nextBindableMsgTag < NUM_ADDR_TBL_ENTRIES &&
          nmp->snvt.mtagCount < 0xFF)
      {
         nmp->snvt.mtagCount++;
         return(gp->nextBindableMsgTag++);
      }
      else
      {
         /* Ran out of addr table entries or no room in mtagCount. */
         return(-1);
      }
   }

   if (gp->nextNonbindableMsgTag < 0xFFFF)
   {
      return(gp->nextNonbindableMsgTag++); /* We have lots of this */
   }
   else
   {
      return(-1); /* Extremely unlikely to happen. */
   }
}

/*****************************************************************************
Function:  IsArrayNV
Returns:   Returns TRUE if a given primary index corresponds to a
           network array variable.
           The index can be any of the indices for the array
           elements.
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Reference: None
Purpose:   To determine if an index corresponds to a network
           array variable. This is needed as this info is not part
           of the nv config or fixed tables.
           For simple variables, dimOut is set to 1 and baseIndexOut
           is set to the given index. (By default).
Comments:  gp->nvArrayTbl is an array of structures with two
           fields: nvIndex and dim. The table contains nvIndex
           and dim only for array variables. The table is populated
           during AddNV.
           For array variables, dim (dimension of array) and array
           base index passed back if provided with non-null addresses.
*************************************************************************/
static Boolean IsArrayNV(int16 nvIndexIn,
                         uint16 *dimOut, int16 *baseIndexOut)
{
   int16 i;

   if (dimOut)
   {
      *dimOut = 1; /* Default for simple variables */
   }
   if (baseIndexOut)
   {
      *baseIndexOut = nvIndexIn;
   }
   /* Sequential search. OK if there are not too many arrays. */
   if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize)
   {
      return(FALSE); /* Only primary variables can be arrays */
   }
   for (i = 0; i < gp->nvArrayTblSize; i++)
   {
      if (nvIndexIn >= gp->nvArrayTbl[i].nvIndex &&
          nvIndexIn <  gp->nvArrayTbl[i].nvIndex + gp->nvArrayTbl[i].dim)
      {
         /* Lies in the range for this array variable */
         if (dimOut)
         {
            *dimOut   = gp->nvArrayTbl[i].dim;
         }
         if (baseIndexOut)
         {
            *baseIndexOut = gp->nvArrayTbl[i].nvIndex;
         }
         return(TRUE);
      }
   }
   return(FALSE); /* Not found. Must be a simple variable */
}

/* Application can call this fn to put itself offline */
void GoOffline(void)
{
   OfflineEvent();
   gp->appPgmMode = OFF_LINE;
}

/* Appplication can call this fn to put itself unconfigured */
void GoUnconfigured(void)
{
   int i, numDomains;

   eep->readOnlyData.nodeState = APPL_UNCNFG;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* Set appPgmMode to OFF_LINE so that when we configured again, the
      application program will be soft-off-line.
      See P9-47 of Tech Device Data Book Rev 3 */
   gp->appPgmMode              = OFF_LINE;
   if (eep->readOnlyData.twoDomains)
   {
      numDomains = 2;
   }
   else
   {
      numDomains = 1;
   }

   /* Overwrite all domain information. Destroys auth key */
   for (i = 0; i < numDomains; i++)
   {
      memcpy(eep->domainTable[i].domainId, "gmrdwf", DOMAIN_ID_LEN);
      eep->domainTable[i].subnet = 0;
      eep->domainTable[i].cloneDomain = 0;
      eep->domainTable[i].node = 0;
      eep->domainTable[i].len = 0xFF;
      memcpy(eep->domainTable[i].key, "NO_KEY", AUTH_KEY_LEN);
   }
}

/*----------------------------------End of app.c-----------------------------*/
_
```

A.11 Network Management Commands

```
/*****************************************************************************
          File:        netmgmt.c

       Version:        1.7

     Reference:CNP, part 1, sub-clause 13.7.

       Purpose:        App Layer/Network Management

                       The functions in this file handle the network
                       management messages (HandleNM()) and network
                       diagnostic messages (HandleND()).

          Note:
                           Discard if    Honor even if     Never
                           Not Request   read/write prot   Authcated
                           -----------   ---------------   -----------
Network Management Messages:
NM_QUERY_ID           0x61  YES           YES               YES
NM_RESPOND_TO_QUERY   0x62  no            YES               YES
NM_UPDATE_DOMAIN      0x63  no            YES               no
NM_LEAVE_DOMAIN       0x64  no            YES               no
NM_UPDATE_KEY         0x65  no            YES               no
NM_UPDATE_ADDR        0x66  no            YES               no
NM_QUERY_ADDR         0x67  YES           YES               no
NM_QUERY_NV_CNFG      0x68  YES           YES               no
NM_UPDATE_GROUP_ADDR  0x69  no            YES               no
NM_QUERY_DOMAIN       0x6A  YES           YES               no
NM_UPDATE_NV_CNFG     0x6B  no            YES               no
NM_SET_NODE_MODE      0x6C  no            YES               no
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
NM_READ_MEMORY          0x6D  YES           Limited           no
NM_WRITE_MEMORY         0x6E  no            Limited           no
NM_CHECKSUM_RECALC      0x6F  no            YES               no
NM_WINK                 0x70  no            YES               no
NM_MEMORY_REFRESH       0x71  no            YES               no
NM_QUERY_SNVT           0x72  YES           YES               no
NM_NV_FETCH             0x73  YES           YES               no

Network Diagnostic Messages:
ND_QUERY_STATUS         0x51  YES           YES               YES
ND_PROXY_COMMAND        0x52  YES           YES               YES
ND_CLEAR_STATUS         0x53  NO            YES               no
ND_QUERY_XCVR           0x54  YES           YES               no

Manual Service Request Message:
NM_MANUAL_SERVICE_REQUEST
                        0x1F -na-          -na-              -na-

The HandleNM() function is called for each network management
message, and does the appropriate processing.  Some messages
are simple enough to be processed right in HandleNM(), others
have their own function that is called by HandleNM().

The HandleND() function is called for each network diagnostic
message.  Like HandleNM(), the HandleND() function takes care
of simple messages, and more complicated messages are handled
by their own function.

*************************************************************************/
/*-------------------------------------------------------------------------
Section: Includes
-------------------------------------------------------------------------*/
#include <string.h>

#include <cnp_1.h>
#include <node.h>
#include <app.h>
#include <api.h>
#include <netmgmt.h>

/*-------------------------------------------------------------------------
Section: Constant Definitions
-------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------------
Section: Type Definitions
-------------------------------------------------------------------------*/
#pragma maxalign(1)

typedef struct
{
   uint16  transmissionErrors;
   uint16  transmitTXFailures;
   uint16  receiveTXFull;
   uint16  lostMessages;
   uint16  missedMessages;
   uint8   resetCause;
   uint8   nodeState;
   uint8   versionNumber;
   uint8   errorLog;
   uint8   modelNumber;
} NDQueryStat;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
#pragma maxalign()

/*-------------------------------------------------------------------------
Section: Globals
-------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------------
Section: Function Prototypes.
-------------------------------------------------------------------------*/
/* Local Functions */
static void RecomputeChecksum(void);
static void NMNDRespond(NtwkMgmtMsgType msgType,
                        Status success,
                        APPReceiveParam *appReceiveParamPtr,
                        APDU *apduPtr);

/* Network management related functions */
void HandleNMQueryId(APPReceiveParam *appReceiveParamPtr,
                     APDU *apduPtr);

void HandleNMWink(APPReceiveParam *appReceiveParamPtr,
                  APDU *apduPtr);

void HandleNMLeaveDomain(APPReceiveParam *appReceiveParamPtr,
                         APDU *apduPtr);

void HandleNMQueryAddr(APPReceiveParam *appReceiveParamPtr,
                       APDU *apduPtr);

void HandleNMQueryNvCnfg(APPReceiveParam *appReceiveParamPtr,
                         APDU *apduPtr);

void HandleNMQuerySIData(APPReceiveParam *appReceiveParamPtr,
                          APDU *apduPtr);

void HandleNMNVFetch(APPReceiveParam *appReceiveParamPtr,
                     APDU *apduPtr);

void HandleNMReadMemory(APPReceiveParam *appReceiveParamPtr,
                        APDU *apduPtr);

void HandleNMWriteMemory(APPReceiveParam *appReceiveParamPtr,
                         APDU *apduPtr);

void HandleNMQueryDomain(APPReceiveParam *appReceiveParamPtr,
                          APDU *apduPtr);

/* Network diagnostics related functions */
void HandleNDQueryStatus(APPReceiveParam *appReceiveParamPtr,
                          APDU *apduPtr);

void HandleNDProxyCommand(APPReceiveParam *appReceiveParamPtr,
                           APDU            *apduPtr);

void HandleNDTransceiverStatus(APPReceiveParam *appReceiveParamPtr,
                               APDU            *apduPtr);

void HandleNDClearStatus(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr);

/* This function is for use by HandleNDProxyCommand */
void HandleNDQueryUnconfig(APPReceiveParam *appReceiveParamPtr,
                           APDU *apduPtr);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* External function from physical layer to get the transceiver status. */
extern void GetTransceiverStatus(Byte transceiverStatusOut[]);

/*-----------------------------------------------------------------------------
Section: Function Definitions
-----------------------------------------------------------------------------*/
/*****************************************************************************
Function:  RecomputeChecksum
Returns:   None.
Reference: None
Purpose:   To compute the configuration checksum and store.
Comments:  None.
*****************************************************************************/
static void RecomputeChecksum(void)
{
   eep->configCheckSum = ComputeConfigCheckSum();
}


/*****************************************************************************
Function:  ManualServiceRequestMessage
Returns:   TRUE if the message is sent, FALSE otherwise.
Reference: None
Purpose:   Produces a manual service request message.
Comments:  Prototype in api.h so that application program can use this
           function too. Returns TRUE or FALSE so that application
           program can determine whether the message was sent or not.
*****************************************************************************/
Boolean ManualServiceRequestMessage(void)
{
   NWSendParam       *nwSendParamPtr;
   APDU              *apduRespPtr;

   if(QueueFull(&gp->nwOutQ))
   {
      return(FALSE); /* Can't send it now. Try later. */
   }

   /* Send unack domain wide broadcast message. */
   nwSendParamPtr          = QueueTail(&gp->nwOutQ);
   nwSendParamPtr->pduSize = 1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN;
   if (nwSendParamPtr->pduSize > gp->nwOutBufSize)
   {
      return(FALSE); /* Do not have sufficient space to send the message. */
   }
   apduRespPtr                            = (APDU *)(nwSendParamPtr + 1);
   apduRespPtr->code.allBits              = 0x7F;  /* Manual Service Request. */
   nwSendParamPtr->destAddr.domainIndex   = FLEX_DOMAIN;
   nwSendParamPtr->destAddr.flexDomainLen = 0;
   nwSendParamPtr->pduType                = APDU_TYPE;
   nwSendParamPtr->destAddr.addressMode   = BROADCAST;
   nwSendParamPtr->destAddr.addr.addr0    = 0; /* Domain wide broadcast. */
   nwSendParamPtr->deltaBL                = 0;
   nwSendParamPtr->altPath                = 0; /* don't use alternate path. */
   nwSendParamPtr->tag                    = MANUAL_SERVICE_REQ_TAG_VALUE;

   memcpy(apduRespPtr->data, eep->readOnlyData.uniqueNodeId,
          UNIQUE_NODE_ID_LEN);
   memcpy(&(apduRespPtr->data[UNIQUE_NODE_ID_LEN]),
          eep->readOnlyData.progId, ID_STR_LEN);
   EnQueue(&gp->nwOutQ);
   gp->manualServiceRequest = FALSE;
   return(TRUE);
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
}


/*****************************************************************************
Function:  NMNDRespond
Returns:   None
Reference: None
Purpose:   Respond with success or failure code to current message.
           Can be called either for ND or NM messages.
Comments:  None
*****************************************************************************/
static void NMNDRespond(NtwkMgmtMsgType msgType,
                        Status success,
                        APPReceiveParam *appReceiveParamPtr,
                        APDU            *apduPtr)
{
   Queue             *tsaOutQPtr;
   TSASendParam      *tsaSendParamPtr;
   APDU              *apduRespPtr;

   /* If service type is not request, nothing to do. */
   if(appReceiveParamPtr->service != REQUEST)
   {
      /* Does not make sense to respond if the original is not a request. */
      return;
   }

   if (QueueFull(&gp->tsaRespQ))
   {
      return; /* Can't send the response. Do nothing. */
   }

   /* Send response. */
   tsaOutQPtr                   = &gp->tsaRespQ;
   tsaSendParamPtr              = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service     = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId       = appReceiveParamPtr->reqId;
   tsaSendParamPtr->apduSize    = 1;
   apduRespPtr                  = (APDU *)(tsaSendParamPtr + 1);
   if (msgType == NM_MESSAGE)
   {
      apduRespPtr->code.allBits     =
         (success == SUCCESS?NM_resp_success:NM_resp_failure)
            | (apduPtr->code.allBits & 0x1F);
   }
   else
   {
      apduRespPtr->code.allBits     =
         (success == SUCCESS?ND_resp_success:ND_resp_failure)
            | (apduPtr->code.allBits & 0x0F);

   }
   EnQueue(tsaOutQPtr);
}

/*****************************************************************************
Function:  HandleNMLeaveDomain
Returns:   None
Reference: None
Purpose:   Handle incoming NM LeaveDomain message.
              - Delete the indicated domain table entry.
              - Recompute the configuration checksum.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
              - If message was received on the indicated domain,
                do not respond
              - If node no longer belongs to any Domain:
                + become unconfigured
                + reset
Comments:  None.
*******************************************************************************/
void HandleNMLeaveDomain(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
   /* Fail if message is not 2 bytes long */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* It is not required that the processing of network management commands
    * include parameter validation.  In general, it is the responsibility of the
    * configuration tool to ensure that requests are well formed. */
   if(appReceiveParamPtr->pduSize != 2)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* If the domain index is bad, fail */
   if (apduPtr->data[0] != 0 && apduPtr->data[0] != 1)
   {
      nmp->errorLog = INVALID_DOMAIN;
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   /* Leave the domain */
   memcpy(eep->domainTable[apduPtr->data[0]].domainId,
          "gmrdwf",
          DOMAIN_ID_LEN);
   eep->domainTable[apduPtr->data[0]].subnet      = 0;
   eep->domainTable[apduPtr->data[0]].cloneDomain = 1;
   eep->domainTable[apduPtr->data[0]].node        = 0;
   eep->domainTable[apduPtr->data[0]].len         = 0xFF; /* not in use */
   memcpy(eep->domainTable[apduPtr->data[0]].key,
          "NO_KEY",
          AUTH_KEY_LEN);

   /* Recompute the configuration checksum */
   RecomputeChecksum();

   /* If message not received on domain just left, then respond */
   if(apduPtr->data[0] != appReceiveParamPtr->srcAddr.domainIndex)
   {
      NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr,apduPtr);
   }

   /* If not a member of any domain, go unconfigured and reset. */
   if ((!eep->readOnlyData.twoDomains &&
         eep->domainTable[0].len == 0xFF) ||
       (eep->readOnlyData.twoDomains    &&
        eep->domainTable[0].len == 0xFF &&
        eep->domainTable[1].len == 0xFF) )
   {
      eep->readOnlyData.nodeState = APPL_UNCNFG;
      nmp->resetCause             = SOFTWARE_RESET;
      gp->resetNode               = TRUE; /* Scheduler will reset the node */
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   DeQueue(&gp->appInQ);
}

/*****************************************************************************
Function:  HandleNMQueryAddr
Returns:   None
Reference: None
Purpose:   Handle incoming NM Query Address message.
Comments:  None.
*****************************************************************************/
void HandleNMQueryAddr(APPReceiveParam *appReceiveParamPtr,
                       APDU            *apduPtr)
{
   Queue             *tsaOutQPtr;
   TSASendParam      *tsaSendParamPtr;
   APDU              *apduRespPtr;
   AddrTableEntry    *ap;

   tsaOutQPtr = &gp->tsaRespQ;
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   /* Fail if message is not 2 bytes long. */
   if(appReceiveParamPtr->pduSize != 2)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */
   /* Fail if the address table index is bad and set statistics. */
   if (apduPtr->data[0] >= NUM_ADDR_TBL_ENTRIES)
   {
      nmp->errorLog = INVALID_ADDR_TABLE_INDEX;
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* Fail if the buffer is insufficient for the response. */
   if (1 + sizeof(AddrTableEntry) > gp->tsaRespBufSize )
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }

   ap = AccessAddress(apduPtr->data[0]); /* ap should not NULL. */

   /* Send response */
   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);
   apduRespPtr->code.allBits     = NM_resp_success | NM_QUERY_ADDR;
   tsaSendParamPtr->apduSize     = 1 + sizeof(AddrTableEntry);
   memcpy(apduRespPtr->data, ap, sizeof(AddrTableEntry));
   EnQueue(tsaOutQPtr);
   DeQueue(&gp->appInQ);
}

/*****************************************************************************
Function:  HandleNMQueryNvCnfg
Returns:   None
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Reference: None
Purpose:   Handle incoming NM Query Netvar Config message.
Comments:  None.
********************************************************************************/
void HandleNMQueryNvCnfg(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
   Queue             *tsaOutQPtr;
   TSASendParam      *tsaSendParamPtr;
   APDU              *apduRespPtr;
   uint16            n;
   uint16            nvAliasTableSize;

   /* Fail if the request does not have correct size */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (appReceiveParamPtr->pduSize != 2 && appReceiveParamPtr->pduSize != 4)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   n = apduPtr->data[0];
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (n == 255 && appReceiveParamPtr->pduSize != 4)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   /* Calculate current alias table size */
   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

   /* Decode index */

   /* *** START INFORMATIVE - Escaped NV Index *** */
   /* In implementations that handle a maximum number of network variables that is
   /* less than 255, it is not necessary to check for network variable escapes of
   /* 255.
   */
   if(n == 255)
   {
      n = (uint16) apduPtr->data[1];
      n = (n << 8) | apduPtr->data[2];
   }
   /* *** END INFORMATIVE - Escaped NV Index *** */

   /* Fail if there is insufficient space to send the response */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (
        (n < nmp->nvTableSize && (1 + sizeof(NVStruct)) > gp->tsaRespBufSize ) ||
        (n >= nmp->nvTableSize && n < nmp->nvTableSize + nvAliasTableSize &&
          (1 + sizeof(AliasStruct)) > gp->tsaRespBufSize )
      )
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   /* Send response */
   tsaOutQPtr                   = &gp->tsaRespQ;
   tsaSendParamPtr              = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service     = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId       = appReceiveParamPtr->reqId;
   apduRespPtr                  = (APDU *)(tsaSendParamPtr + 1);
   apduRespPtr->code.allBits    = NM_resp_success | NM_QUERY_NV_CNFG;
   if(n < nmp->nvTableSize)
   {
      /* Copy the NVStruct entry */
      tsaSendParamPtr->apduSize = 1 + sizeof(NVStruct);
      memcpy(apduRespPtr->data, &(nmp->nvConfigTable[n]),
         sizeof(NVStruct));
   }
   else if(n < nmp->nvTableSize + nvAliasTableSize)
   {
      /* Copy the alias table entry. */
      tsaSendParamPtr->apduSize = 1 + sizeof(AliasStruct);
      n = n - nmp->nvTableSize;
      memcpy(apduRespPtr->data, &(nmp->nvAliasTable[n]),
         sizeof(AliasStruct));
   }
   else
   {
      nmp->errorLog             = INVALID_NV_INDEX;
      apduRespPtr->code.allBits = NM_resp_failure | NM_QUERY_NV_CNFG;
      tsaSendParamPtr->apduSize = 1;
   }

   EnQueue(tsaOutQPtr);
   DeQueue(&gp->appInQ);
}

/*****************************************************************************
Function:  HandleNMNVFetch
Returns:   None
Reference: None
Purpose:   Handle incoming NM NV Fetch message.
Comments:  None.
*****************************************************************************/
void HandleNMNVFetch(APPReceiveParam *appReceiveParamPtr,
                     APDU            *apduPtr)
{
   Queue             *tsaOutQPtr;
   TSASendParam      *tsaSendParamPtr;
   APDU              *apduRespPtr;
   uint16            n;
   uint8             i;

   /* Check if the message has correct size. If not, fail. */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (appReceiveParamPtr->pduSize != 2 &&
       appReceiveParamPtr->pduSize != 4)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* *** END INFORMATIVE - Parameter Validation *** */
   n = apduPtr->data[0];
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (n == 255 && appReceiveParamPtr->pduSize != 4)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */
   tsaOutQPtr = &gp->tsaRespQ;
   /* Send response */
   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);

   /* *** START INFORMATIVE - Escaped NV Index *** */
   /* See "START INFORMATIVE - Escaped NV Index" above. */
   if(n == 255)
   {
      n = apduPtr->data[1];
      n = (n << 8) | apduPtr->data[2];
      memcpy(apduRespPtr->data, apduPtr->data, 3); /* copy the index */
      i = 3; /* index where value of nv is copied */
   }
   /* *** END INFORMATIVE - Escaped NV Index *** */
   else
   {
      apduRespPtr->data[0] = (char) n;
      i = 1;
   }
   if (n < nmp->nvTableSize)
   {
      /* Make sure there is sufficient space for the response. Else, fail. */
      /* *** START INFORMATIVE - Parameter Validation *** */
      /* See "START INFORMATIVE - Parameter Validation" above. */
      if (nmp->nvFixedTable[n].nvLength + i + 1 > gp->tsaRespBufSize)
      {
         NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
         DeQueue(&gp->appInQ);
         return;
      }
      /* *** END INFORMATIVE - Parameter Validation *** */
      memcpy(&apduRespPtr->data[i],
             nmp->nvFixedTable[n].nvAddress,
             nmp->nvFixedTable[n].nvLength);
      tsaSendParamPtr->apduSize = nmp->nvFixedTable[n].nvLength + i + 1;
      apduRespPtr->code.allBits = NM_resp_success | NM_NV_FETCH;
      EnQueue(tsaOutQPtr);

   }
   else
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
   }
   DeQueue(&gp->appInQ);
}

/*****************************************************************************
Function:  HandleNMQuerySIData
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
Returns:   None
Reference: None
Purpose:   Handle incoming NM Query SI Data message.
Comments:  None.
*******************************************************************************/
void HandleNMQuerySIData(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
   Queue             *tsaOutQPtr;
   TSASendParam      *tsaSendParamPtr;
   APDU              *apduRespPtr;
   uint16             offset;
   uint8              count;

   tsaOutQPtr = &gp->tsaRespQ;

   /* *** START INFORMATIVE - Query SI Data *** */
   /* It is only necessary to respond to Query SI Data messages (AKA Query SNVT)
    * if the implementation does not support direct memory read/[write] of the
    * device self identification data.  Only a device with a
    * "ReadOnlyDataStruct.snvtStruct"field with value 0xffff must implement this
    * function. */

   /* Fail if message is not 4 bytes long */
   if(appReceiveParamPtr->pduSize != 4)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }

   /* Decode offset and count */
   offset = apduPtr->data[0];
   offset = (offset << 8) | apduPtr->data[1];
   count  = apduPtr->data[2];
   /* Check if we have enough space to respond for this message */
   if (count + 1 > gp->tsaRespBufSize)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }

   /* Send response */
   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);
   apduRespPtr->code.allBits     = NM_resp_success | NM_QUERY_SNVT;
   tsaSendParamPtr->apduSize     = 1 + count;
   memcpy(apduRespPtr->data,  offset + (char *)&(nmp->snvt), count);
   EnQueue(tsaOutQPtr);
   /* *** END INFORMATIVE - Query SI Data *** */
   DeQueue(&gp->appInQ);
}

/*******************************************************************************
Function:  HandleNMWink
Returns:   None
Reference: None
Purpose:   Handle incoming NM Wink message.
Comments:  None.
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
*****************************************************************************/
void HandleNMWink(APPReceiveParam *appReceiveParamPtr,
                  APDU            *apduPtr)
{
   Queue             *tsaQueuePtr;
   TSASendParam      *tsaSendParamPtr;
   APDU              *apduRespPtr;
   int8               subcmd;
   int8               niIndex;

   subcmd = 0;

   if (appReceiveParamPtr->pduSize > 1)
   {
      subcmd  = apduPtr->data[0];
   }

   if (appReceiveParamPtr->pduSize > 2)
   {
      niIndex = apduPtr->data[1];
   }

   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (appReceiveParamPtr->pduSize > 3)
   {
      /* Incorrect size */
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   /* *** START INFORMATIVE - Wink Handling *** */
   /* Handling of wink subcommands is not required.  That is, it is acceptable
    * to treat all requests with this code as if they were simple wink requests.
    * Wink subcommands are a set of backward compatible extensions to the wink
    * command code. */
   if(appReceiveParamPtr->pduSize <= 1 || subcmd == 0)
   {
      if (appReceiveParamPtr->service != REQUEST)
      {
         /* Any service except request/response */
         Wink();  /* Simple Wink */
      }
      else
      {
        NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      }
      DeQueue(&gp->appInQ);
      return;
   }

   /* must be requesting SEND_ID_INFO. */
   if(appReceiveParamPtr->service != REQUEST)
   {
      DeQueue(&gp->appInQ); /* Discard */
      return; /* The message should be a request. */
   }

   tsaQueuePtr = &gp->tsaRespQ;
   if (QueueFull(tsaQueuePtr))
   {
      return;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }
   tsaSendParamPtr                = QueueTail(tsaQueuePtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   /* Note: This implementation only has one NI Interface. i.e 0 */
   /* Send response. */
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   tsaSendParamPtr->nullResponse = FALSE;
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);

   if (niIndex == 0 &&
       UNIQUE_NODE_ID_LEN + ID_STR_LEN + 2 <= gp->tsaRespBufSize )
   {
      tsaSendParamPtr->apduSize      = ID_STR_LEN + UNIQUE_NODE_ID_LEN + 2;
      apduRespPtr->data[0]           = 0; /* Interface not down. */
      apduRespPtr->code.allBits      = NM_resp_success | NM_WINK;
      memcpy(&apduRespPtr->data[1], eep->readOnlyData.uniqueNodeId,
             UNIQUE_NODE_ID_LEN);
      memcpy(&(apduRespPtr->data[1+UNIQUE_NODE_ID_LEN]),
             eep->readOnlyData.progId, ID_STR_LEN);
   }
   else
   {
      tsaSendParamPtr->apduSize  = 1;
      apduRespPtr->code.allBits  = NM_resp_failure | NM_WINK;
   }
   EnQueue(tsaQueuePtr);
   /* *** END INFORMATIVE - Wink Handling *** */
   DeQueue(&gp->appInQ);
}

/************************************************************************
Function:  HandleNMQueryId
Returns:   None
Reference: None
Purpose:   Handle incoming NM Query ID message.
Comments:  The message must be a request. There must be space in
           response queue. See HandleNM (We do these checks first).
************************************************************************/
void HandleNMQueryId(APPReceiveParam *appReceiveParamPtr,
                     APDU            *apduPtr)
{
   NMQueryIdRequest  *pid;
   char              *memp;
   Queue             *tsaOutQPtr;
   TSASendParam      *tsaSendParamPtr;
   APDU              *apduRespPtr;
   Boolean            allowed;
   uint16             offset;

   /* Fail if message does not have the correct size. Should be 2 or 6+n */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if(appReceiveParamPtr->pduSize != 2 && appReceiveParamPtr->pduSize < 6)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   tsaOutQPtr = &gp->tsaRespQ;

   /* Init some fields here. Assume we may fail. */
```

```
tsaSendParamPtr               = QueueTail(tsaOutQPtr);
tsaSendParamPtr->altPathOverride = FALSE;
apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);
tsaSendParamPtr->service      = RESPONSE;
tsaSendParamPtr->nullResponse = FALSE;
tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
tsaSendParamPtr->apduSize     = 1;
/* Since there are a lot of fail cases, init code to indicate failed
   response. */
apduRespPtr->code.allBits     = NM_resp_failure | NM_QUERY_ID;

pid = (NMQueryIdRequest *)&apduPtr->data;
offset = pid->offset;

/* if optional fields are present, check that the data field has
   sufficient bytes. */
/* *** START INFORMATIVE - Parameter Validation *** */
/* See "START INFORMATIVE - Parameter Validation" above. */
if (appReceiveParamPtr->pduSize > 2 &&
    appReceiveParamPtr->pduSize != (6 + pid->count))
{
   /* The message does not have sufficient data or it has too much data. */
   NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
   return;
}
/* *** END INFORMATIVE - Parameter Validation *** */

switch(pid->selector)
{
   case UNCONFIGURED:
      if(!NodeUnConfigured())
      {
         /* Not unconfigured - don't respond. */
         tsaSendParamPtr->nullResponse = TRUE;
         EnQueue(tsaOutQPtr);
         DeQueue(&gp->appInQ);
         return;
      }
      break;
   case SELECTED:
      if(!gp->selectQueryFlag)
      {
         /* Not selected - don't respond. */
         tsaSendParamPtr->nullResponse = TRUE;
         EnQueue(tsaOutQPtr);
         DeQueue(&gp->appInQ);
         return;
      }
      break;
   case SELECTED_UNCFG:   /* must be selected and unconfigured */
      if(!gp->selectQueryFlag)
      {
         /* Not selected - don't respond. */
         tsaSendParamPtr->nullResponse = TRUE;
         EnQueue(tsaOutQPtr);
         DeQueue(&gp->appInQ);
         return;
      }
      if(!NodeUnConfigured())
      {
         /* Not unconfigured - don't respond */
         tsaSendParamPtr->nullResponse = TRUE;
         EnQueue(tsaOutQPtr);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            DeQueue(&gp->appInQ);
            return;
         }
         break;
      default:
         EnQueue(tsaOutQPtr); /* Failed response. */
         DeQueue(&gp->appInQ);
         return;
   }

   /* If memory matching is present, check memory match */
   if(appReceiveParamPtr->pduSize > 2)
   {
      switch(pid->mode)
      {
         case ABSOLUTE_MEM_ADDR:
            memp = (char *)nmp;
            if(offset >= 0xF000)
            {
               memp = (char *)eep - 0xF000;
            }
            else if (offset >= 0xFC00)
            {
               memp = (char *)nmp->memMapSpace - 0xFC00;
            }
            break;
         case CONFIG_RELATIVE:
            memp = (char *)&(eep->configData);
            break;
         case READ_ONLY_RELATIVE:
            memp = (char *)&(eep->readOnlyData);
            break;
         default:
            EnQueue(tsaOutQPtr);
            DeQueue(&gp->appInQ);
            return;   /* Failed response. */
      }

      memp += offset;

      /* Absolute addressing to read snvt is not possible */
      allowed = (memp >= (char *)&eep->readOnlyData &&
                 memp + apduPtr->data[3] < (char *)&eep->domainTable[0]);
      if (!allowed)
      {
         EnQueue(tsaOutQPtr);
         DeQueue(&gp->appInQ);
         return; /* Failed response. */
      }

      if(memcmp(pid->data, memp, pid->count) != 0)
      {
         /* Compare failed - don't reply. */
         tsaSendParamPtr->nullResponse = TRUE;
         EnQueue(tsaOutQPtr);
         DeQueue(&gp->appInQ);
         return;
      }
   }

   /* Send response */
   if (1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN <= gp->tsaRespBufSize )
   {
      tsaSendParamPtr->apduSize = 1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      apduRespPtr->code.allBits = NM_resp_success | NM_QUERY_ID;
      memcpy(apduRespPtr->data, eep->readOnlyData.uniqueNodeId,
             UNIQUE_NODE_ID_LEN);
      memcpy(&(apduRespPtr->data[UNIQUE_NODE_ID_LEN]),
             eep->readOnlyData.progId, ID_STR_LEN);
   }

   EnQueue(tsaOutQPtr);
   DeQueue(&gp->appInQ);
}


/*************************************************************************
Function:  HandleNMQueryDomain
Returns:   None
Reference: None
Purpose:   Handle incoming NM QueryDomain message.
Comments:  Must be a Request.
*************************************************************************/
void HandleNMQueryDomain(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
   Queue             *tsaOutQPtr;
   TSASendParam      *tsaSendParamPtr;
   APDU              *apduRespPtr;
   int                n;  /* Domain Index */


   /* Fail if message does not have the correct size. */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if(appReceiveParamPtr->pduSize != 2)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   tsaOutQPtr = &gp->tsaRespQ;
   n = apduPtr->data[0];  /* Domain Index */
   /* If domain index is other than 0 or 1 or if the node is in only one
      domain and the index is 1, then fail. */
   if ((n != 0 && n != 1) || (eep->readOnlyData.twoDomains == 0 && n == 1))
   {
      /* Domain index is bad. */
      nmp->errorLog = INVALID_DOMAIN;
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* If there is not enough space, fail too. */
   if (1 + sizeof(DomainStruct) > gp->tsaRespBufSize)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }

   /* Send response */
   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   tsaSendParamPtr->apduSize     = 1 + sizeof(DomainStruct);
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);
   apduRespPtr->code.allBits     = NM_resp_success | NM_QUERY_DOMAIN;
   memcpy(apduRespPtr->data, &eep->domainTable[n], sizeof(DomainStruct));

   EnQueue(tsaOutQPtr);
   DeQueue(&gp->appInQ);
}


/*****************************************************************************
Function:  HandleNMReadMemory
Returns:   None
Reference: None
Purpose:   Handle incomming NM ReadMemory message
Comments:  None.
*****************************************************************************/
void HandleNMReadMemory(APPReceiveParam *appReceiveParamPtr,
                        APDU            *apduPtr)
{
   Queue               *tsaOutQPtr;
   TSASendParam        *tsaSendParamPtr;
   APDU                *apduRespPtr;
   char                *memp;
   uint16               offset;
   Boolean              allowed;

   tsaOutQPtr = &gp->tsaRespQ;

   /* Fail if message is not 5 bytes long. */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if(appReceiveParamPtr->pduSize != 5)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   offset = (apduPtr->data[1] << 8) | apduPtr->data[2];

   /* Assemlbe response */
   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   tsaSendParamPtr->apduSize     = 1 + apduPtr->data[3];

   apduRespPtr = (APDU *)(tsaSendParamPtr + 1);

   apduRespPtr->code.allBits = NM_resp_success | NM_READ_MEMORY;

   switch(apduPtr->data[0])
   {
      case ABSOLUTE_MEM_ADDR:
         memp = (char *)nmp;
         if(offset >= 0xF000)
         {
            memp = (char *)eep - 0xF000;
         }
         else if (offset >= 0xFC00)
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        {
           memp = (char *)nmp->memMapSpace - 0xFC00;
        }
        break;
     case READ_ONLY_RELATIVE:
     default:
        memp = (char *)&(eep->readOnlyData);
        break;
     case CONFIG_RELATIVE:
        memp = (char *)&(eep->configData);
        break;
     case STAT_RELATIVE:
        memp = (char *)&(nmp->stats);
        break;
  }

  memp += offset;

  /* If readWriteProtect flag is on, then only readonly data structure,
     snvt structures, and configuration structure can be read.
     The one byte read of location 0 (firmware number) is also allowed.
     Reference implementation does not support snvt reading through
     absolute addressing. readOnlyData.snvtStruct is 0xFFFF */


  if(eep->readOnlyData.readWriteProtect == TRUE)
  {
     allowed = (memp >= (char *)&eep->readOnlyData &&
                memp + apduPtr->data[3] < (char *)&eep->domainTable[0]) ||
                (memp == (char *)&nmp && apduPtr->data[3] == 1);

     if(!allowed)
     {
        tsaSendParamPtr->apduSize = 1;
        apduRespPtr->code.allBits = NM_resp_failure | NM_READ_MEMORY;
        EnQueue(tsaOutQPtr);
        DeQueue(&gp->appInQ);
        return;
     }
  }

  if (apduPtr->data[3] <= gp->tsaRespBufSize)
  {
     memcpy(apduRespPtr->data, memp, apduPtr->data[3]);
  }
  else
  {
     tsaSendParamPtr->apduSize = 1;
  }

  if (memp == (char *)nmp && apduPtr->data[3] == 1)
  {
     /* trap for absolute read of location 0 and write
        version number (hard code it as 11) */
     apduRespPtr->data[0] = 11;
  }
  EnQueue(tsaOutQPtr);
  DeQueue(&gp->appInQ);
}

/*************************************************************************
Function:  HandleNMWriteMemory
Returns:   None
Reference: None
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Purpose:   Handle incoming NM WriteMemory message.
Comments:  None.
********************************************************************************/
void HandleNMWriteMemory(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
   Queue                *tsaOutQPtr;
   TSASendParam         *tsaSendParamPtr;
   APDU                 *apduRespPtr;
   NMWriteMemoryRequest *pr;
   char                 *memp;
   uint16                offset;
   Boolean               allowed;

   tsaOutQPtr = &gp->tsaRespQ;

   /* Fail if message is not at least 6 bytes long */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if(appReceiveParamPtr->pduSize < 6)
   {
      NMNDRespond(NM_MESSAGE, FAILURE,appReceiveParamPtr,apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   /* Pointer to struct describing memory request */
   pr = (NMWriteMemoryRequest *)&apduPtr->data[0];

   offset = pr->offset;

   /* Fail if message length doesn't match count. Poke can use 16 data bytes.
      Allow that. Note that the code takes one byte. */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if(appReceiveParamPtr->pduSize != 6 + pr->count &&
      appReceiveParamPtr->pduSize != 17)
   {
      NMNDRespond(NM_MESSAGE, FAILURE,appReceiveParamPtr,apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   if (appReceiveParamPtr->service == REQUEST)
   {
      /* Assemble response */
      tsaSendParamPtr               = QueueTail(tsaOutQPtr);
      tsaSendParamPtr->altPathOverride = FALSE;
      tsaSendParamPtr->service      = RESPONSE;
      tsaSendParamPtr->nullResponse = FALSE;
      tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
      tsaSendParamPtr->apduSize     = 1;
      apduRespPtr = (APDU *)(tsaSendParamPtr + 1);
   }

   switch(pr->mode)
   {
      case ABSOLUTE_MEM_ADDR:
         memp = (char *)nmp;
         if(offset >= 0xF000)
         {
            memp = (char *)eep - 0xF000;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      }
      else if (offset >= 0xFC00)
      {
         memp = (char *)nmp->memMapSpace - 0xFC00;
      }
      break;
   case CONFIG_RELATIVE:
      memp = (char *)&(eep->configData);
      break;
   case STAT_RELATIVE:
      memp = (char *)&(nmp->stats);
      break;
   case READ_ONLY_RELATIVE:
      memp = (char *)&(eep->readOnlyData);
      break;
   default:
      /* Invalid Mode */
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
}

memp += offset;

/* Check if the range of memory cells written is good. */
allowed = (memp >= (char *)nmp &&
           (memp + pr->count) <= ((char *)nmp + 64 * 1024)) ||
           (memp >= (char *)eep &&
                  (memp + pr->count) <= ((char *)eep + sizeof(EEPROM)));
if (! allowed)
{

   /* Send failure response if the message was a request */
   NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
   return;
}

/* If readwrite flag is on, then only config structure can be written. */
if(eep->readOnlyData.readWriteProtect == TRUE)
{
   allowed = (memp >= (char *)&eep->configData &&
              memp + pr->count < (char *)&eep->domainTable[0]);

   if(! allowed)
   {
      /* Send failure response if the message was a request */
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
}

/* Need to copy only the data array in NM_Req structure.
   The header is 5 bytes long */
/* We have to assume that pr->count is good. Max is 255 */
/* Reference implementation has no application check sum.
   Only config checksum */
memcpy(memp, apduPtr->data+5, pr->count);

if (pr->form & CNFG_CS_RECALC) {
   RecomputeChecksum();
}
if (pr->form & ACTION_RESET) {
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      gp->resetNode   = TRUE;
      nmp->resetCause = SOFTWARE_RESET;
   }

   /* There is no harm in responding even when the node is reset. It will
      be lost anyway */
   if(appReceiveParamPtr->service == REQUEST)
   {
      apduRespPtr->code.allBits = NM_resp_success | NM_WRITE_MEMORY;
      EnQueue(tsaOutQPtr);
   }
   DeQueue(&gp->appInQ);
}



/*****************************************************************************
Function:  HandleProxyResponse
Returns:   None
Reference: None
Purpose:   Handle Proxy Response Message.
Comments:  None.
*****************************************************************************/
void HandleProxyResponse(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{
   Queue         *tsaOutQPtr;
   TSASendParam  *tsaSendParamPtr;
   APDU          *apduRespPtr;

   /* Send response */
   tsaOutQPtr = &gp->tsaRespQ;

   if(QueueFull(tsaOutQPtr))
   {
      return;  /* Return without processing message */
   }

   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId        = nmp->pxyData.reqId;
   tsaSendParamPtr->apduSize     = appReceiveParamPtr->pduSize;
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);
   if (appReceiveParamPtr->pduSize <= gp->tsaRespBufSize)
   {
      memcpy(apduRespPtr, apduPtr, appReceiveParamPtr->pduSize);
   }
   else
   {
      tsaSendParamPtr->apduSize = 1;
   }

   EnQueue(tsaOutQPtr);

   /* Done processing */
   DeQueue(&gp->appInQ);

   /* Reset the flag as we have now responded */
   nmp->pxyData.pxyType = -1;
}

/*****************************************************************************
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Function:  HandleNDQueryStatus
Returns:   none
Reference: None
Purpose:   Handle Network Diagnostics Query Status Message.
Comments:  None.
*******************************************************************************/
void HandleNDQueryStatus(APPReceiveParam *appReceiveParamPtr,
                         APDU *apduPtr)
{
   NDQueryStat    ndq;
   Queue         *tsaOutQPtr;
   TSASendParam  *tsaSendParamPtr;
   APDU          *apduRespPtr;

   tsaOutQPtr = &gp->tsaRespQ;
   if (((apduPtr->code.allBits & 0x0F) == ND_QUERY_STATUS) &&
        appReceiveParamPtr->pduSize != 1)
   {
      /* Incorrect size. Fail. */
      NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }

   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (((apduPtr->code.allBits & 0x0F) == ND_PROXY_COMMAND) &&
        appReceiveParamPtr->pduSize != 2)
   {
      /* Incorrect size. Fail. */
      NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   ndq.transmissionErrors = nmp->stats.transmissionErrors;
   ndq.transmitTXFailures = nmp->stats.transmitTXFailures;
   ndq.receiveTXFull      = nmp->stats.receiveTXFull;
   ndq.lostMessages       = nmp->stats.lostMessages;
   ndq.missedMessages     = nmp->stats.missedMessages;
   ndq.resetCause         = nmp->resetCause;
   if (eep->readOnlyData.nodeState == CNFG_ONLINE &&
       gp->appPgmMode == OFF_LINE)
   {
      ndq.nodeState = SOFT_OFFLINE;
   }
   else
   {
      ndq.nodeState = eep->readOnlyData.nodeState;
   }
   ndq.versionNumber = 128;
   ndq.errorLog      = nmp->errorLog;
   ndq.modelNumber   = 128;
    /* Send response */
   tsaSendParamPtr              = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   tsaSendParamPtr->apduSize     = 1 + sizeof(NDQueryStat);
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);
   /* The response code is formed using apduPtr->code.allBits as the
      actual could be a proxy command and the response code will be
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      different. Thus, using apduPtr->code.allBits works for both
      native query status command as well as proxy based query status */
   if (tsaSendParamPtr->apduSize <= gp->tsaRespBufSize)
   {
      apduRespPtr->code.allBits     =
         ND_resp_success | (apduPtr->code.allBits & 0x0F);
      memcpy(apduRespPtr->data, &ndq,  sizeof(NDQueryStat));
   }
   else
   {
      apduRespPtr->code.allBits     =
         ND_resp_failure | (apduPtr->code.allBits & 0x0F);
      tsaSendParamPtr->apduSize     = 1;
   }
   EnQueue(tsaOutQPtr);
   DeQueue(&gp->appInQ);

}

/*****************************************************************************
Function:  HandleNDProxyCommand
Returns:   none
Reference: None
Purpose:   Handle network diagnostics proxy request message.
Comments:  None.
*****************************************************************************/
void HandleNDProxyCommand(APPReceiveParam *appReceiveParamPtr,
                          APDU            *apduPtr)
{
   Queue         *tsaOutQPtr;
   TSASendParam  *tsaSendParamPtr;
   APDU          *apduSendPtr;


   /* See if the proxy command is to be forwarded or it is for us
      to process. If the address information in the packet is missing,
      then there is no forwarding information and hence we respond directly */
   if (appReceiveParamPtr->pduSize == 2)
   {
       /* Address portion is missing. This node is the proxy target. */
      tsaOutQPtr = &gp->tsaRespQ;
      /* Send the response directly back to the sender */
      switch(apduPtr->data[0])
      {
         case 0:
            HandleNDQueryUnconfig(appReceiveParamPtr, apduPtr);
            break;
         case 1:
            HandleNDQueryStatus(appReceiveParamPtr, apduPtr);
            break;
         case 2:
            HandleNDTransceiverStatus(appReceiveParamPtr, apduPtr);
            break;
         default:
            /* Invalid sub_command. Send failure response */
            NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
      }
      return;
   }

   /* Send failure response if a proxy command is already in progress
      or we are proxy agent and the message received is on flex domain
      or it does not have correct size */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
if (
    nmp->pxyData.pxyType != -1 ||
    appReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN ||
    (apduPtr->data[1] == UNIQUE_NODE_ID &&
         appReceiveParamPtr->pduSize !=
            (2 + sizeof(AddrTableEntry) + UNIQUE_NODE_ID_LEN) ) ||
    (apduPtr->data[1] != UNIQUE_NODE_ID &&
         appReceiveParamPtr->pduSize != (2 + sizeof(AddrTableEntry)) )
   )
{
   NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
   return;
}


/* We need to forward the proxy now. Set the target queue ptrs */
if(appReceiveParamPtr->priority)
{
   tsaOutQPtr = &gp->tsaOutPriQ;
}
else
{
   tsaOutQPtr = &gp->tsaOutQ;
}

/* Check if the target queue has space for forwarding this request. */
if(QueueFull(tsaOutQPtr))
{
   return;  /* Return without processing message */
}

/* Save the proxy data for processing the response later.
   Also, pxyType will no longer be equal to -1 that serves
   as a kind of lock on this record. Future proxy commands
   will be ignored until this proxy command is done. The proxy
   command is done as soon as we get the response and we forward
   it to the original sender or when the transaction forwarding
   the proxy times out. Session layer will not give us duplicate
   requests and hence we don't need to handle that here. Also,
   session layet will take care of sending response again for
   duplicate requests. */
nmp->pxyData.pxyType = apduPtr->data[0];
nmp->pxyData.reqId   = appReceiveParamPtr->reqId;

/* Generate request message */
tsaSendParamPtr                = QueueTail(tsaOutQPtr);
/* First copy the msg_out_addr info in the proxy command */
if (apduPtr->data[1] == UNIQUE_NODE_ID)
{
   memcpy(&tsaSendParamPtr->destAddr,&apduPtr->data[1],
          sizeof(MsgOutAddr) + UNIQUE_NODE_ID_LEN);
}
else
{
   memcpy(&tsaSendParamPtr->destAddr,&apduPtr->data[1],
          sizeof(MsgOutAddr));
}
/* Set the domain index to be the one in which it was received.
   Note that this cannot be flex domain. Transport or session
   will use this instead of the one in the destAddr (i.e msg_out_addr) */
tsaSendParamPtr->domainIndex   = appReceiveParamPtr->srcAddr.domainIndex;
tsaSendParamPtr->service  = REQUEST;
tsaSendParamPtr->auth     = FALSE;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   tsaSendParamPtr->tag      = (MsgTag) 0xFFFF;
   /* Proxy relays the message, the altpath bit should be same as the one
      used for the proxy message received */
   tsaSendParamPtr->altPathOverride = TRUE;
   tsaSendParamPtr->altPath = appReceiveParamPtr->altPath;
   tsaSendParamPtr->apduSize = 2; /* Always two bytes: code + sub_command */
   apduSendPtr  = (APDU *)(tsaSendParamPtr + 1);
   /* Copy the code as it is */
   apduSendPtr->code.allBits = apduPtr->code.allBits;
   apduSendPtr->data[0] = apduPtr->data[0];
   EnQueue(tsaOutQPtr);
   DeQueue(&gp->appInQ);
}

/*****************************************************************************
Function:  HandleNDTransceiverStatus
Returns:   none
Reference: None
Purpose:   Handle network diagnostics transceiver status message.
Comments:  None.
*****************************************************************************/
void HandleNDTransceiverStatus(APPReceiveParam *appReceiveParamPtr,
                               APDU            *apduPtr)
{
   Queue         *tsaOutQPtr;
   TSASendParam  *tsaSendParamPtr;
   APDU          *apduRespPtr;
   Byte           transceiverStatus[NUM_COMM_PARAMS];
   uint8          n;

   /* Check for proper size of the message. Regular pdusize is 1 byte.
      If this fn is called due to proxy request, then it is 2 bytes. */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (appReceiveParamPtr->pduSize > 2)
   {
      NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */
   else if (appReceiveParamPtr->pduSize == 2)
   {
      /* Make sure it is a proxy request. Or else, length is wrong. */
      if (apduPtr->code.nd.ndFlag == 0x5 &&
          apduPtr->code.nd.ndCode == ND_PROXY_COMMAND &&
          apduPtr->data[0] == 2)
      {
         ; /* It is indeed a proxy command. Length ok. Proceed. */
      }
      else
      {
         NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
         DeQueue(&gp->appInQ);
         return;
      }
   }

   if (eep->configData.commType == SPECIAL_PURPOSE)
   {
      GetTransceiverStatus(transceiverStatus);
      n = NUM_COMM_PARAMS;
   }
   else
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   {
      n = 0;
   }

   /* Send response */
   tsaOutQPtr = &gp->tsaRespQ;
   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   tsaSendParamPtr->apduSize     = 1 + n;
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);

   /* If a node does not have transceiver status registers, fail */
   if ( n > 0 &&
       tsaSendParamPtr->apduSize <= gp->tsaRespBufSize)
   {
      /* No registers or not enough space */
      apduRespPtr->code.allBits     = ND_resp_success |
                                      (apduPtr->code.allBits & 0x0F);

      memcpy(apduRespPtr->data, transceiverStatus, n);
   }
   else
   {
      apduRespPtr->code.allBits     = ND_resp_failure |
                                      (apduPtr->code.allBits & 0x0F);

      tsaSendParamPtr->apduSize     = 1;
   }
   EnQueue(tsaOutQPtr);
   DeQueue(&gp->appInQ);
}

/************************************************************************
Function:  HandleNDClearStatus
Returns:   none
Reference: None
Purpose:   Handle network diagnostics clear status message.
Comments:  None.
************************************************************************/
void HandleNDClearStatus(APPReceiveParam *appReceiveParamPtr,
                         APDU            *apduPtr)
{

   /* Check for proper size of the message */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (appReceiveParamPtr->pduSize != 1)
   {
      NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   /* Clear Status */
   nmp->stats.transmissionErrors   = 0;
   nmp->stats.transmitTXFailures   = 0;
   nmp->stats.receiveTXFull        = 0;
   nmp->stats.lostMessages         = 0;
   nmp->stats.missedMessages       = 0;
   nmp->stats.layer2Received       = 0;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   nmp->stats.layer3Received       = 0;
   nmp->stats.layer3Transmitted    = 0;
   nmp->stats.transmitTXRetries    = 0;
   nmp->stats.backlogOverflow      = 0;
   nmp->stats.lateAcknowledgements = 0;
   nmp->stats.collisions           = 0;
   nmp->resetCause                 = CLEARED;
   nmp->errorLog                   = NO_ERRORS;  /* Cleared */

   nmp->stats.layer6_7MsgsSent     = 0;
   nmp->stats.layer6_7RespSent     = 0;
   nmp->stats.layer6_7MsgsRcvd     = 0;
   nmp->stats.layer6_7RespRcvd     = 0;
   nmp->stats.lateResponses        = 0;
   nmp->stats.lateChallenges       = 0;
   nmp->stats.lateReplies          = 0;

   /* NMNDRespond will send response only if the msg is REQUEST */
   NMNDRespond(ND_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
}

/*****************************************************************************
Function:  HandleNDQueryUnconfig
Returns:   None
Reference: None
Purpose:   Handle Query Unconfig through Proxy message.
Comments:  This is a simplified version of HandleNMQueryId.
           Only unconfig version of Query ID is supported.
*****************************************************************************/
void HandleNDQueryUnconfig(APPReceiveParam *appReceiveParamPtr,
                           APDU            *apduPtr)
{
   Queue             *tsaOutQPtr;
   TSASendParam      *tsaSendParamPtr;
   APDU              *apduRespPtr;


   /* Check for proper size of the message */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (appReceiveParamPtr->pduSize != 2)
   {
      NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */

   tsaOutQPtr = &gp->tsaRespQ;

   /* Init some fields here. */
   tsaSendParamPtr               = QueueTail(tsaOutQPtr);
   tsaSendParamPtr->altPathOverride = FALSE;
   apduRespPtr                   = (APDU *)(tsaSendParamPtr + 1);
   tsaSendParamPtr->service      = RESPONSE;
   tsaSendParamPtr->nullResponse = FALSE;
   tsaSendParamPtr->reqId        = appReceiveParamPtr->reqId;
   tsaSendParamPtr->apduSize     = 1;
   /* apduPtr->code.allBits should be proxy code */
   apduRespPtr->code.allBits     = ND_resp_failure |
                                   (apduPtr->code.allBits & 0x0F);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   if(!NodeUnConfigured())
   {
      /* Not unconfigured - don't respond. */
      tsaSendParamPtr->nullResponse = TRUE;
      EnQueue(tsaOutQPtr);
      DeQueue(&gp->appInQ);
      return;
   }

   /* Send response */
   if (1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN <= gp->tsaRespBufSize )
   {
      tsaSendParamPtr->apduSize = 1 + UNIQUE_NODE_ID_LEN + ID_STR_LEN;
      apduRespPtr->code.allBits = ND_resp_success |
                                  (apduPtr->code.allBits & 0x0F);
      memcpy(apduRespPtr->data, eep->readOnlyData.uniqueNodeId,
             UNIQUE_NODE_ID_LEN);
      memcpy(&(apduRespPtr->data[UNIQUE_NODE_ID_LEN]),
             eep->readOnlyData.progId, ID_STR_LEN);
   }

   EnQueue(tsaOutQPtr);
   DeQueue(&gp->appInQ);
}


/*****************************************************************************
Function:  HandleND
Returns:   None
Reference: None
Purpose:   Handle incomming network diagnostic message.
Comments:  None.
*****************************************************************************/
void HandleND(APPReceiveParam *appReceiveParamPtr,
              APDU            *apduPtr)
{
   if(appReceiveParamPtr->service == RESPONSE)
   {
      /* It is not legal for a response to be an ND command */
      DeQueue(&gp->appInQ);
      return;
   }

   /* If network diagnostics messages need authentication
      and the message did not pass authentication and
      the node is not unconfigured
      then discard those messages that should be discarded and return. */
   if(!NodeUnConfigured() && eep->configData.nmAuth &&
      !appReceiveParamPtr->auth)
   {
      /* Only two messages are allowed. Others should be discarded */
      if (apduPtr->code.nd.ndCode != ND_QUERY_STATUS &&
          apduPtr->code.nd.ndCode != ND_PROXY_COMMAND)
      {
         if(QueueFull(&gp->tsaRespQ))
         {
            return;  /* Can't send response now. Try later. */
         }
         nmp->errorLog = AUTHENTICATION_MISMATCH;
         NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
         DeQueue(&gp->appInQ);
         return;
      }
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   /* Only clear status command need not be request message. Others
      must be request/response */
   if (apduPtr->code.nd.ndCode != ND_CLEAR_STATUS &&
       appReceiveParamPtr->service != REQUEST)
   {
      /* Discard. We can't respond anyway as it is not a request */
      DeQueue(&gp->appInQ);
      return;
   }

   if (appReceiveParamPtr->service == REQUEST &&
       QueueFull(&gp->tsaRespQ))
   {
      return; /* Can't send a response. Try later. */
   }

   /* Handle various network diagnostic message codes */
   switch(apduPtr->code.nd.ndCode)
   {
      case ND_QUERY_STATUS:
         HandleNDQueryStatus(appReceiveParamPtr, apduPtr);
         return;
      case ND_PROXY_COMMAND:
         HandleNDProxyCommand(appReceiveParamPtr, apduPtr);
         return;
      case ND_CLEAR_STATUS:
         HandleNDClearStatus(appReceiveParamPtr, apduPtr);
         return;
      case ND_QUERY_XCVR:
         HandleNDTransceiverStatus(appReceiveParamPtr, apduPtr);
         return;
      default:
         /* Discard unrecognized diagnostic command */
         NMNDRespond(ND_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
         DeQueue(&gp->appInQ);
   }
}

/*****************************************************************************
Function:  HandleNM
Returns:   None
Reference: None
Purpose:   Handle incomming network management messages.
Comments:  None.
*****************************************************************************/
void HandleNM(APPReceiveParam *appReceiveParamPtr,
              APDU            *apduPtr)
{
   int16            i;
   uint16           n; /* for nv index */
   AddrTableEntry  *ap;
   GroupAddrMode   *groupStrPtr;
   uint16           addrIndex;
   NVStruct        *np;
   uint16           nvAliasTableSize;
   uint8            pduSize;

   if(appReceiveParamPtr->service == RESPONSE)
   {
      /* It is not legal for a response to be an NM command. */
      DeQueue(&gp->appInQ);
      return;
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* If network management messages need authentication
   and the message did not pass authentication and
   the node is not unconfigured
   then discard those messages that should be discarded and return. */
if(!NodeUnConfigured() && eep->configData.nmAuth && !appReceiveParamPtr->auth)
{
   /* Only two messages are allowed. Others should be discarded */
   if (apduPtr->code.nm.nmCode != NM_QUERY_ID &&
       apduPtr->code.nm.nmCode != NM_RESPOND_TO_QUERY)
   {
      if(QueueFull(&gp->tsaRespQ))
      {
         return;  /* Can't send response now. Try later. */
      }
      nmp->errorLog = AUTHENTICATION_MISMATCH;
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
}

/* If the service type is not request, discard certain messages. */
if(appReceiveParamPtr->service != REQUEST)
{
   switch(apduPtr->code.nm.nmCode)
   {
      /* messages that need request/resp. */
      case NM_QUERY_ID:
      case NM_QUERY_ADDR:
      case NM_QUERY_NV_CNFG:
      case NM_QUERY_DOMAIN:
      case NM_READ_MEMORY:
      case NM_QUERY_SNVT:
      case NM_NV_FETCH:
         DeQueue(&gp->appInQ);
         return;
      default:
         break;
   }
}

/* If service type is request, and there is no room for response
   return without processing - so message will be processed later. */
if(appReceiveParamPtr->service == REQUEST)
{
   if(QueueFull(&gp->tsaRespQ))
   {
      return; /* Wait until there is room for response. */
   }
}

/* Handle various network mgmt message codes */
switch(apduPtr->code.nm.nmCode)
{
   case NM_QUERY_ID:
      HandleNMQueryId(appReceiveParamPtr,apduPtr);
      return;
   case NM_RESPOND_TO_QUERY:
      /* Fail if message is not 2 bytes long or the byte is bad. */
      if(appReceiveParamPtr->pduSize != 2 ||
         (apduPtr->data[0] != 0 && apduPtr->data[0] != 1) )
      {
         NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      DeQueue(&gp->appInQ);
      return;
   }
   gp->selectQueryFlag = apduPtr->data[0];
   NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr,apduPtr);
   DeQueue(&gp->appInQ);
   return;
case NM_UPDATE_DOMAIN:
   /* Fail if message is not the right size or the domain_index is bad. */
   if(appReceiveParamPtr->pduSize != 2 + sizeof(DomainStruct))
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   if (apduPtr->data[0] != 0 && apduPtr->data[0] != 1)
   {
      nmp->errorLog = INVALID_DOMAIN;
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* If the node is in only one domain and the request is to update
      domain index 1, then we send a failure response. */
   if (eep->readOnlyData.twoDomains == 0 &&
       apduPtr->data[0] == 1)
   {
      nmp->errorLog = INVALID_DOMAIN;
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* Reference implementation allows even clone domain update. */
   if (((DomainStruct *)&apduPtr->data[1])->cloneDomain == 1)
   {
      UpdateDomain((DomainStruct *)&apduPtr->data[1], apduPtr->data[0]);
   }
   else
   {
      UpdateCloneDomain((DomainStruct *)&apduPtr->data[1],
                        apduPtr->data[0]);
   }
   NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
   RecomputeChecksum();
   DeQueue(&gp->appInQ);
   return;
case NM_LEAVE_DOMAIN:
   HandleNMLeaveDomain(appReceiveParamPtr, apduPtr);
   return;
case NM_UPDATE_KEY:
   /* Fail if message is not of correct length or domain index is bad. */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if(appReceiveParamPtr->pduSize != 2 + AUTH_KEY_LEN)
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   if (apduPtr->data[0] != 0 && apduPtr->data[0] != 1)
   {
      nmp->errorLog = INVALID_DOMAIN;
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            return;
         }
         /* *** END INFORMATIVE - Parameter Validation *** */

         for(i = 0; i < AUTH_KEY_LEN; i++)
         {
            eep->domainTable[apduPtr->data[0]].key[i] +=
               apduPtr->data[i+1];
         }
         RecomputeChecksum();
         NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr,apduPtr);
         DeQueue(&gp->appInQ);
         return;
      case NM_UPDATE_ADDR:
         /* Check for incorrect size. Allow for padding. */
         /* *** START INFORMATIVE - Parameter Validation *** */
         /* See "START INFORMATIVE - Parameter Validation" above. */
         if (  appReceiveParamPtr->pduSize < 6 ||
               appReceiveParamPtr->pduSize > 7)
         {
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
            return;
         }
         if (apduPtr->data[1] != UNBOUND)
         {
            /* Not turnaround. Must be 7 bytes. */
            if (appReceiveParamPtr->pduSize != 7)
            {
               NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
               DeQueue(&gp->appInQ);
               return;
            }
         }
         /* *** END INFORMATIVE - Parameter Validation *** */
         /* Fail if the address table index is bad. */
         if (apduPtr->data[0] >= NUM_ADDR_TBL_ENTRIES)
         {
            nmp->errorLog = INVALID_ADDR_TABLE_INDEX;
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
            return;
         }

         UpdateAddress((AddrTableEntry *)&apduPtr->data[1], apduPtr->data[0]);
         RecomputeChecksum();
         NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
         DeQueue(&gp->appInQ);
         return;
      case NM_QUERY_ADDR:
         HandleNMQueryAddr(appReceiveParamPtr, apduPtr);
         return;
      case NM_QUERY_NV_CNFG:
         HandleNMQueryNvCnfg(appReceiveParamPtr, apduPtr);
         return;
      case NM_UPDATE_GROUP_ADDR:
         /* This message must be delivered with group addressing and is
            updated based on the domain in which it was received. Hence,
            flex domain is not allowed. */
         if (appReceiveParamPtr->srcAddr.addressMode != MULTICAST ||
             appReceiveParamPtr->srcAddr.domainIndex == FLEX_DOMAIN)
         {
            /* This message should be sent in MULTICAST. Fail */
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            DeQueue(&gp->appInQ);
            return;
         }
         /* *** START INFORMATIVE - Parameter Validation *** */
         /* See "START INFORMATIVE - Parameter Validation" above. */
         if (appReceiveParamPtr->pduSize != 1 + sizeof(AddrTableEntry))
         {
            /* Incorrect size */
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
            return;
         }
         /* *** END INFORMATIVE - Parameter Validation *** */
         /* For accessing the corresponding address table entry,
            let us use the domainIndex in which it was received and
            the group in which it was received. It makes sense to use
            the domainIndex in which the message was received rather than
            the domain index in the packet(that will be same for all
            recipients) as it may be different for different nodes. */
         groupStrPtr = (GroupAddrMode *)&apduPtr->data[0];
         if (groupStrPtr->groupFlag == 1)
         {
            addrIndex = AddrTableIndex(appReceiveParamPtr->srcAddr.domainIndex,
                                       appReceiveParamPtr->srcAddr.group);
         }

         /* Make sure we got a good index. */
         if (groupStrPtr->groupFlag != 1 || addrIndex == 0xFF)
         {
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
            return;
         }
         ap = AccessAddress(addrIndex); /* ap cannot be NULL */
         /* Only group size and timer values should be changed */
         ap->groupEntry.groupSize  = groupStrPtr->groupSize;
         ap->groupEntry.rptTimer   = groupStrPtr->rptTimer;
         ap->groupEntry.retryCount = groupStrPtr->retryCount;
         ap->groupEntry.rcvTimer   = groupStrPtr->rcvTimer;
         ap->groupEntry.txTimer    = groupStrPtr->txTimer;
         RecomputeChecksum();
         NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
         DeQueue(&gp->appInQ);
         return;
      case NM_QUERY_DOMAIN:
         HandleNMQueryDomain(appReceiveParamPtr,apduPtr);
         return;
      case NM_UPDATE_NV_CNFG:
         /* Check for pdu size. Fail if the size is too small or too big */
         /* *** START INFORMATIVE - Parameter Validation *** */
         /* See "START INFORMATIVE - Parameter Validation" above. */
         if (appReceiveParamPtr->pduSize < 5 ||
             appReceiveParamPtr->pduSize > 10)
         {
            NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
            DeQueue(&gp->appInQ);
            return;
         }
         /* *** END INFORMATIVE - Parameter Validation *** */
         nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;
         /* Decode index */

         n = apduPtr->data[0];
         /* *** START INFORMATIVE - Escaped NV Index *** */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* See "START INFORMATIVE - Escaped NV Index" above. */
if(n == 255)
{
   n = (uint16)apduPtr->data[1];
   n = (n << 8) | apduPtr->data[2];
   if (n < nmp->nvTableSize)
   {
      pduSize = sizeof(NVStruct) + 4; /* escaped regular update */
   }
   else
   {
      /* Escaped alias update. Assume that host_primary is
         absent for now. */
      pduSize = (sizeof(AliasStruct) - 2) + 4;
   }
   np = (NVStruct *)(&apduPtr->data[3]);
}
/* *** END INFORMATIVE - Escaped NV Index *** */
else
{
   if (n < nmp->nvTableSize)
   {
      pduSize = sizeof(NVStruct) + 2; /* regular update */
   }
   else
   {
      /* Alias update. Assume that host_primary is absent for now. */
      /* last 2 is for index + code */
      pduSize = (sizeof(AliasStruct) - 2) + 2;
   }
   np = (NVStruct *)(&apduPtr->data[1]);
}

/* Update nv config or alias table */
if(n < nmp->nvTableSize)
{
   if (appReceiveParamPtr->pduSize >= pduSize)
   {
      memcpy(&nmp->nvConfigTable[n], np, sizeof(NVStruct));
   }
   else
   {
      /* Incorrect size */
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
}
else if(n < nmp->nvTableSize + nvAliasTableSize)
{
   n = n - nmp->nvTableSize; /* Alias table index */
   /* Check for various forms of alias update */
   if (((AliasStruct *)np)->primary == 0xFF &&
       appReceiveParamPtr->pduSize == pduSize)
   {
      /* host_primary missing. default to 0xffff. Null alias update. */
      ((AliasStruct *)np)->hostPrimary = 0xffff;
   }
   else if (((AliasStruct *)np)->primary == 0xFF)
   {
      /* escaped alias. hostPrimary is present */
      pduSize += 2;
   }
   /* Update the nv alias table */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      if (appReceiveParamPtr->pduSize >= pduSize)
      {
         memcpy(&nmp->nvAliasTable[n], np, sizeof(AliasStruct));
      }
      else
      {
         /* Incorrect size */
         NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
         DeQueue(&gp->appInQ);
         return;
      }
   }
   else
   {
      /* Invalid nv table index */
      nmp->errorLog = INVALID_NV_INDEX;
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }

   /* Recompute checksum and send response */
   RecomputeChecksum();
   NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
   return;

/* NM_SET_NODE_MODE:
 *                                                          Possible
 * Description                   State  Mode   Service LED  in ref imp?
 * ------------------------------------------------------------------
 * Applicationless, unconfigured   3     -     On          NO
 * Unconfigured (w/application)    2     -     Flashing    YES
 * Configured, Hard Offline        6     -     Off         NO
 * Configured                      4     1     Off         YES
 * Configured, Soft offline        4     0     Off         YES
 *
 * The NM_SET_NODE_MODE message encompasses a lot of functionality,
 * and impacts some other areas of the implementation.
 * 1) Mode is not maintained in EEPROM
 * 2) A node that is soft-offline will go on-line when it is reset
 * 3) The hard-offline state is preserved across reset
 * 4) For either hard or soft offline, the scheduler is disabled
 * 5) When soft-offline:
 *    A) Polling an NV will return NULL data
 *    B) Incomming network variable updates are handled normally
 *    C) But nv_update_occurs events will be lost
 * 6) In all other states except configured:
 *    A) No response is returned on NV polls
 *    B) Incomming NV updates are discarded
 * 7) If a node is in a non-configured state, is reset and then issued
 *    a command to go configured, it will come up soft offline
 * 8) If a set node mode message changes the mode to offline or online
 *    the approprite task (if any) is executed
 * 9) Changing the node state recomputes the configuration checksum
 */
case NM_SET_NODE_MODE:
   /* Fail if message is not 2 or 3 bytes long. */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if(appReceiveParamPtr->pduSize < 2 || appReceiveParamPtr->pduSize > 3)
   {
      NMNDRespond(NM_MESSAGE, FAILURE,appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   return;
}
if ( (apduPtr->data[0] != 3 && (appReceiveParamPtr->pduSize < 2 ||
      appReceiveParamPtr->pduSize > 3)) ||
     (apduPtr->data[0] == 3 && appReceiveParamPtr->pduSize != 3)
   )
{
   /* Incorrect size */
   NMNDRespond(NM_MESSAGE, FAILURE,appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
   return;
}
/* *** END INFORMATIVE - Parameter Validation *** */

/* Mode on-line and mode off-line messages must not be request
   messages. See Page 9-47 Technology Device Data Book. Rev 3 */
if (
      (apduPtr->data[0] == 0 || apduPtr->data[0] == 1)
      &&
      appReceiveParamPtr->service == REQUEST
   )
{
   /* Fail. */
   NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
   return;
}

switch(apduPtr->data[0])
{
   case 0:  /* Go to soft offline state */
      if (AppPgmRuns())
      {
         OfflineEvent();  /* Indicate to application program. */
      }
      eep->readOnlyData.nodeState = CNFG_ONLINE;
      gp->appPgmMode   = OFF_LINE;
      gp->ioOutputPin1 = FALSE; /* LED off */
      /* No response given as the message is not a request. */
      break;
   case 1:  /* Go on-line */
      OnlineEvent(); /* Indicate to application program. */
      eep->readOnlyData.nodeState = CNFG_ONLINE;
      gp->appPgmMode   = ON_LINE;
      gp->ioOutputPin1 = FALSE; /* LED off */
      /* No response given as the message is not a request. */
      break;
   case 2:  /* Application reset */
      gp->resetNode   = TRUE;
      nmp->resetCause = SOFTWARE_RESET; /* Software reset. */
      /* No response since the node is being reset. */
      break;
   case 3:  /* Change State */
      /* Fail if message is not 3 bytes long. */
      if(appReceiveParamPtr->pduSize != 3)
      {
         NMNDRespond(NM_MESSAGE, FAILURE,
                     appReceiveParamPtr, apduPtr);
         break;
      }
      eep->readOnlyData.nodeState = apduPtr->data[1];
      /* Preserve the state of appPgmMode except for
         NO_APPL_UNCNFG. */
      if (eep->readOnlyData.nodeState == NO_APPL_UNCNFG)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
         {
            gp->appPgmMode = NOT_RUNNING;
         }
         RecomputeChecksum();
         /* Respond with success if the message was a request. */
         NMNDRespond(NM_MESSAGE, SUCCESS,
                     appReceiveParamPtr, apduPtr);
         break;
      default:
         /* Let us reset the node for this case */
         gp->resetNode   = TRUE;
         nmp->resetCause = SOFTWARE_RESET;
         break;
   }
   DeQueue(&gp->appInQ);
   return;
case NM_READ_MEMORY:
   HandleNMReadMemory(appReceiveParamPtr, apduPtr);
   return;
case NM_WRITE_MEMORY:
   HandleNMWriteMemory(appReceiveParamPtr, apduPtr);
   return;
case NM_CHECKSUM_RECALC:
   /* Fail if the message does not have correct size or has bad value. */
   /* *** START INFORMATIVE - Parameter Validation *** */
   /* See "START INFORMATIVE - Parameter Validation" above. */
   if (appReceiveParamPtr->pduSize != 2 ||
       (apduPtr->data[0] != 1 && apduPtr->data[0] != 4))
   {
      NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
      DeQueue(&gp->appInQ);
      return;
   }
   /* *** END INFORMATIVE - Parameter Validation *** */
   /* We don't have checksum for application. Just config checksum */
   RecomputeChecksum();
   NMNDRespond(NM_MESSAGE, SUCCESS, appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
   return;
case NM_WINK:   /* Same as NM_INSTALL */
   HandleNMWink(appReceiveParamPtr,apduPtr);
   return;
case NM_MEMORY_REFRESH:
   /* *** START INFORMATIVE - Memory Refresh *** */
   /* It is acceptable to also respond with a failure to the memory
    * refresh request if the implementation does not include memory
    * that requires refreshing.  An example of responding to memory
    * refresh with non failure responses is not provided here. */
   NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
   DeQueue(&gp->appInQ);
   /* *** END INFORMATIVE - Memory Refresh *** */
   return;
case NM_QUERY_SNVT:
   HandleNMQuerySIData(appReceiveParamPtr, apduPtr);
   return;
case NM_NV_FETCH:
   HandleNMNVFetch(appReceiveParamPtr, apduPtr);
   return;
case NM_MANUAL_SERVICE_REQUEST:
   /* This is unsolicited message from a node. Reference
      implementation ignores manual service request message from other nodes
   */
   DeQueue(&gp->appInQ);
   return;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      default:
         /* This is where any message that is not taken care of should be
            handled. An example is product query command. For now, we treat
            everything else as unrecognized network management message. */
         NMNDRespond(NM_MESSAGE, FAILURE, appReceiveParamPtr, apduPtr);
         DeQueue(&gp->appInQ); /* Simply discard it */
   }
}


/*************************************************************************/
_
```

A.12 Configuration Data Structures

```
/*************************************************************************
          File:  node.c

       Version:  1.7

     Reference:  Annex B of this standard

       Purpose:  Configuration Data Strutures that contain
                 information about this node.
                 Also, define all type defintions needed.

          Note:  RefImp supports any number of stacks.
                 A global structure called ProtocolStackData
                 is defined in node.h. An array of such
                 structures is used so that each stack has
                 its own data that it works on. A global
                 pointer gp points to the right structure
                 before the stack code is executed. This
                 is done by the scheduler.

                 The support for multiple-stacks does not include support
                 for mac layer to handle multiple stacks or multiple application
                 programs. A true multi-stack system needs some extra coding.
         To Do:  None
*************************************************************************/
/*-------------------------------------------------------------------------
Section: Includes
-------------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <cnp_1.h>
#include <custom.h>
#include <node.h>
#include <physical.h>

/*-------------------------------------------------------------------
Section: Constant Definitions
-------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------
Section: Type Definitions
-------------------------------------------------------------------*/
/* None */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/*-------------------------------------------------------------------
Section: Globals
-------------------------------------------------------------------*/
EEPROM            *eep; /* actual structure is in eeprom.c */
NmMap             *nmp;
NmMap              nm[NUM_STACKS] = {{0}};
ProtocolStackData *gp;
ProtocolStackData  protocolStackDataGbl[NUM_STACKS];

#if defined(ALTERNATE_STRUCTURES_NEEDED)
msg_in_type        msg_in;
msg_out_type       msg_out;
resp_in_type       resp_in;
resp_out_type      resp_out;
nv_in_addr_type    nv_in_addr;
int16              nv_array_index;
#endif

/*-------------------------------------------------------------------
Section: Local Globals
-------------------------------------------------------------------*/
/* See Tech Device Data Book for the tables with these values */

/* Table on page 9-9. Field Value 1 is missing.
   We assume the value 20 for 1 */
static uint16 bufSizeCodeLGbl[16] =
{
   255,20,20,21,22,24,26,30,34,42,50,66,82,114,146,210
};
/* Table is on page 9-10. Field value 1 is missing.
   We assume the value 1 for 1 */
static uint16 bufCntCodeLGbl[16] =
{
   0,1,1,2,3,5,7,11,15,23,31,47,63,95,127,191
};
static uint16 rptTimerCodeLGbl[16] =
{
   16,24,32,48,64,96,128,192,256,384,512,768,1024,1536,2048,3072
};
static uint16 rcvTimerCodeLGbl[16] =
{
   128,192,256,384,512,768,1024,1536,2048,3072,4096,6144,8192,12288,
   16384,24576
};
static uint16 txTimerCodeLGbl[16] =
{
   16,24,32,48,64,96,128,192,256,384,512,768,1024,1536,2048,3072
};
static uint16 nonGroupTimerCodeLGbl[16] =
{
   128,192,256,384,512,768,1024,1536,2048,3072,4096,6144,8192,
   12288,16384,24576
};

/*-------------------------------------------------------------------
Section: Local Function Prototypes
-------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------
Section: Function Definitions
-------------------------------------------------------------------*/
/*****************************************************************
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Function:  AccessDomain
Returns:   Address of structure corresponding to given index
Purpose:   To return the address of the structure that has domain
           information for this node
Comments:  If an invalid index is given, log error message.
*****************************************************************/

DomainStruct *AccessDomain(uint8 indexIn)
{
   if (indexIn < MAX_DOMAINS)
   {
      return(&eep->domainTable[indexIn]);
   }
   return(NULL);
}


/*****************************************************************
Function:  UpdateDomain
Returns:   None
Purpose:   To Change the domain table entry with given structure.
Comments:  If an invalid index is given, log error message.
*****************************************************************/
void UpdateDomain(DomainStruct *domainInp, uint8 indexIn)
{
   if (domainInp && (indexIn < MAX_DOMAINS))
   {
      eep->domainTable[indexIn] = *domainInp;
      eep->domainTable[indexIn].cloneDomain = 1;
      return;

   }
   if (domainInp)
   {
      ErrorMsg("UpdateDomain: Invalid index.\n");

   }
   else
   {
      ErrorMsg("UpdateDomain: NULL domainInp passed.\n");
   }
}

/*****************************************************************
Function:  UpdateCloneDomain
Returns:   None
Purpose:   To Change the domain table entry with given structure.
Comments:  If an invalid index is given, log error message.
           If this fn is called, then the node can only receive
           messages addressed with Unique Node ID, group unack or
           broadcast addressing mode. CloneDomain is set to 0.
           Clone Domain allows a node to have same subnet and node ID
           as another node and yet receive messages of above types.
           All messages in network have this bit set to 1.
*****************************************************************/
void UpdateCloneDomain(DomainStruct *domainInp, uint8 indexIn)
{
   if (domainInp && (indexIn < MAX_DOMAINS))
   {
      eep->domainTable[indexIn] = *domainInp;
      eep->domainTable[indexIn].cloneDomain = 0;
      return;

   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   if (domainInp)
   {
      ErrorMsg("UpdateDomain: Invalid index.\n");

   }
   else
   {
      ErrorMsg("UpdateDomain: NULL domainInp passed.\n");
   }
}

/*****************************************************************
Function:  AccessAddress
Returns:   Address of structure at given index
Reference: Tech Device Data Book Rev 1 p.9-12
Purpose:   To access address table entry
Comments:  None
*****************************************************************/
AddrTableEntry *AccessAddress(uint16 indexIn)
{
   if (indexIn < NUM_ADDR_TBL_ENTRIES)
   {
      return(&eep->addrTable[indexIn]);
   }
   return(NULL);
}

/*****************************************************************
Function:  UpdateAddress
Returns:   None
Reference: Tech Device Data Rev 1 p.9-12
Purpose:   To update an address table entry.
Comments:  None
*****************************************************************/
void UpdateAddress(AddrTableEntry *addrEntryInp, uint16 indexIn)
{
   if (addrEntryInp && indexIn < NUM_ADDR_TBL_ENTRIES)
   {
      eep->addrTable[indexIn] = *addrEntryInp;
      return;
   }
   if (addrEntryInp)
   {
      nmp->errorLog = INVALID_ADDR_TABLE_INDEX;
      ErrorMsg("UpdateAddress: Invalid index.\n");
   }
   else
   {
      ErrorMsg("UpdateAddress: NULL addrEntryInp passed.\n");
   }
}

/*****************************************************************
Function:  IsGroupMember
Returns:   TRUE if this node belongs to given group. FALSE, else.
Reference: None
Purpose:   To check if a node belongs to a given group in the
           given domain. If it does, also get the member number.
Comments:  If groupMemberOut is NULL, then it is not used.
*****************************************************************/
Boolean IsGroupMember(Byte domainIndexIn, uint8 groupIn,
                      uint8 *groupMemberOut)
{
   uint16 i;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   for (i = 0; i < NUM_ADDR_TBL_ENTRIES; i++)
   {
      if (eep->addrTable[i].addrFormat >= 128)
      {
         /* Group Format */
         if (eep->addrTable[i].groupEntry.groupID == groupIn &&
           eep->addrTable[i].groupEntry.domainIndex == domainIndexIn)
         {
            break;
         }
      }
   }
   if (i == NUM_ADDR_TBL_ENTRIES)
   {
      return(FALSE); /* Not Found */
   }
   if (groupMemberOut)
   {
      *groupMemberOut = eep->addrTable[i].groupEntry.member;
   }
   return(TRUE); /* Found */
}

/*****************************************************************
Function:  AddrTableIndex
Returns:   The index of the address table for the given domain
           and group. 0xFF if not found.
Reference: None
Purpose:   To get the addr table index for a given group and domain.
           If there is no such entry in the addr table, return 0xff.
Comments:  None
*****************************************************************/
uint16 AddrTableIndex(uint8 domainIndexIn, uint8 groupIn)
{
   uint16 i;

   for (i = 0; i < NUM_ADDR_TBL_ENTRIES; i++)
   {
      if (eep->addrTable[i].addrFormat >= 128)
      {
         /* Group Format */
         if (eep->addrTable[i].groupEntry.groupID == groupIn &&
           eep->addrTable[i].groupEntry.domainIndex == domainIndexIn)
         {
            return(i);
         }
      }
   }

   return(0xFF); /* Not Found */
}

/*****************************************************************
Function:  DecodeBufferSize
Returns:   Actual Buffer Size
Reference: Tech Device Data Rev 1 p9-9
Purpose:   To compute the actual buffer size from code
Comments:  None
*****************************************************************/
uint16  DecodeBufferSize(uint8 bufSizeIn)
{
   if (bufSizeIn <= 15)
   {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      return(bufSizeCodeLGbl[bufSizeIn]);
   }
   ErrorMsg("DecodeBufferSize: Invalid code.\n");
   return(0);
}

/*****************************************************************
Function:  DecodeBufferCnt
Returns:   Actual Buffer Count
Reference: Tech Device Data Rev 1 p.9-10
Purpose:   To compute the actual buffer count from code
Comments:  None
*****************************************************************/
uint16  DecodeBufferCnt(uint8 bufCntIn)
{
   if (bufCntIn <= 15)
   {
      return(bufCntCodeLGbl[bufCntIn]);
   }
   ErrorMsg("DecodeBufferCnt: Invalid code.\n");
   return(0);
}

/*****************************************************************
Function:  DecodeRptTimer
Returns:   Actual timer value in ms
Reference: Tech Device Data Rev 1 p.9-17
Purpose:   To compute the actual rpt timer value from code
Comments:  None
*****************************************************************/
uint16 DecodeRptTimer(uint8 rptTimerIn)
{
   if (rptTimerIn <= 15)
   {
      return(rptTimerCodeLGbl[rptTimerIn]);
   }
   ErrorMsg("DecodeRptTimer: Invalid code.\n");
   return(0);
}

/*****************************************************************
Function:  DecodeRcvTimer
Returns:   Actual Receive Timer value in ms
Reference: Tech Device Data Rev 1 p.9-17
Purpose:   To compute the actual rcv timer value in ms from code
Comments:  None
*****************************************************************/
uint16 DecodeRcvTimer(uint8 rcvTimerIn)
{
   if (rcvTimerIn <= 15)
   {
      return(rcvTimerCodeLGbl[rcvTimerIn]);
   }
   ErrorMsg("DecodeRcvTimer: Invalid code.\n");
   return(0);
}

/*****************************************************************
Function:  DecodeTxTimer
Returns:   Actual Transmit Timer Value in ms
Reference: Tech Device Data Rev 1 p.9-17
Purpose:   To compute the actual transmit timer value from code
Comments:  None
*****************************************************************/
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
uint16 DecodeTxTimer(uint8  txTimerIn)
{
   if (txTimerIn <= 15)
   {
      return(txTimerCodeLGbl[txTimerIn]);
   }
   ErrorMsg("DecodeTxTimer: Invalid code.\n");
   return(0);
}

/*****************************************************************
Function:  DecodeNonGroupTimer
Returns:   Actual non-group timer value in ms
Reference: Tech Device Data Rev 1 p.9-11
Purpose:   To Compute the actual non-group timer value from code
Comments:  None
*****************************************************************/
uint16 DecodeNonGroupTimer(uint8 nonGroupTimerIn)
{
   if (nonGroupTimerIn <= 15)
   {
      return(nonGroupTimerCodeLGbl[nonGroupTimerIn]);
   }
   ErrorMsg("DecodeNonGroupTimer: Invalid code.\n");
   return(0);
}

/*****************************************************************
Function:  AccessNV
Returns:   Address of NV conf table entry
Reference: Tech Device Data Rev 1 p.9-18
Purpose:   To Access the NV Config Table Entry given the index
Comments:  None
*****************************************************************/
NVStruct *AccessNV(uint16 indexIn)
{
   if (indexIn < nmp->nvTableSize)
   {
      return(&nmp->nvConfigTable[indexIn]);
   }
   ErrorMsg("AccessNV: Invalid index.\n");
   return(NULL);
}

/*****************************************************************
Function:  UpdateNV
Returns:   None
Reference: Tech Device Data Rev 1 p.9-18
Purpose:   To update an entry in NV Config Table
Comments:  None
*****************************************************************/
void UpdateNV(NVStruct *nvStructInp, uint16 indexIn)
{
   if (nvStructInp && indexIn < nmp->nvTableSize)
   {
      nmp->nvConfigTable[indexIn] = *nvStructInp;
      return;
   }
   if (nvStructInp)
   {
      ErrorMsg("UpdateNV: Invalid index.\n");
   }
   else
   {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      ErrorMsg("UpdateNV: NULL nvStructInp.\n");
   }
}

/*************************************************************
Function:  NVTableIndex
Returns:   index of NV Config Table
Reference: Tech Device Data Rev 1 p.9-18.
Purpose:   To retrieve the index corresponding to a network varname.
Comments:  We don't need this fn for Ref Imp. as the app pgm already
           has the index for all variables.
*************************************************************/
uint16 NVTableIndex(char varNameIn[])
{
   return(0);
}

/*************************************************************
Function:  ErrorMsg
Returns:   None
Reference: None
Purpose:   To store error msgs produced by these functions.
Comments:  It is just a sequence of Bytes that is large.
           If there is no more space, it wraps around and
           logs. Thus, if there are too many error logs,
           we will only have the latest ones.
           Each Log is automatically given a number.
           The output has log number followed by message.
*************************************************************/
void ErrorMsg(char errMessageIn[])
{
   uint16 j, msgNumber;
   uint16 spaceNeeded;

   /* Set the starting index for copy */
   /* If there is insufficient space, wrap around */
   /* spaceNeeded includes space for the null character */
   /* header(xxx. ) + msg + \0 */
   spaceNeeded = 5 + strlen(errMessageIn) + 1;
   if (gp->errorMsgIndex + spaceNeeded > ERROR_MSG_SIZE)
   {
      gp->errorMsgIndex = 0; /* Wrap around */
   }

   /* Compute the Log Number. Make sure it is no more
      than 3 digits */
   gp->errorMsgNumber++;
   if (gp->errorMsgNumber > 999)
   {
      gp->errorMsgNumber = 1;
   }
   msgNumber = gp->errorMsgNumber;

   j = gp->errorMsgIndex;
   sprintf(gp->errorMsg + j, "%3d. ", msgNumber);
   j = j + 5;

   /* Write the message */
   strcpy(gp->errorMsg + j, errMessageIn);

   /* Update errorMsgIndex. Don't count the space for null char */
   gp->errorMsgIndex = gp->errorMsgIndex + spaceNeeded - 1;
}
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/*****************************************************************
Function:  DebugMsg
Returns:   None
Reference: None
Purpose:   To Print Debugging Messages for stacks.
Comments:  Actually not recorded anywhere. One needs to set
           breakpoint at the end of this fn and print temp
           to see the msg.
*****************************************************************/
void DebugMsg(char debugMsgIn[])
{
   char temp[200];
   uint8 i;

   if (strlen(debugMsgIn) > 170)
   {
      return; /* Too large for temp. Ignore it. */
   }

   /* Find our Stack # */
   for (i = 0; i < NUM_STACKS; i++)
   {
      if (gp == &protocolStackDataGbl[i])
      {
         break;
      }
   }
   sprintf(temp, "Time:%5u: Stack %2d: ", GetCurrentMsTime(), i+1);
   strcpy(temp + strlen(temp), debugMsgIn);
}

/*****************************************************************
Function:  AllocateStorage
Returns:   Pointer to data storage allocated or NULL
Reference: None
Purpose:   A Simple version of storage allocator similar to malloc.
           A Global array is used to allocate the srorage.
           If no more space, NULL is retured.
Comments:  There is no function similar to free. There is no need
           for such a funcion in the Reference Implementation.
*****************************************************************/
void *AllocateStorage(uint16 sizeIn)
{
   Byte *ptr;

   if (gp->mallocUsedSize + sizeIn > MALLOC_SIZE)
   {
      nmp->errorLog = MEMORY_ALLOC_FAILURE;
      return(NULL); /* No space for requested size */
   }

   ptr = gp->mallocStorage + gp->mallocUsedSize;
   gp->mallocUsedSize += sizeIn;

   return(ptr);
}

/*****************************************************************
Function:  GetCurrentTime
Returns:   Current Time (not necessarily real time)
Reference:
Purpose:   To get the current time.
Comments:  None
*****************************************************************/
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
uint32 GetCurrentTime(void)
{
   return(*(gp->currentTime));
}

/*****************************************************************
Function:  GetCurrentMsTime
Returns:   Current Time (not necessarily real time)
Reference:
Purpose:   To get the current time.
Comments:  None
*****************************************************************/
uint32 GetCurrentMsTime(void)
{
   return(*(gp->currentTime)/CLOCK_TICKS_PER_MS);
}

/*****************************************************************
Function:  SetMsTimer
Returns:   None
Reference: None
Purpose:   To set a timer value to a given value in ms.
Comments:  The timer is set to given value. The lastUpdateTime is
           set to current time. UpdateTimer is called later to
           update the timer value.
*****************************************************************/
void SetMsTimer(MsTimer *timerOut, uint16 initValueIn)
{
   timerOut->curTimerValue  = initValueIn * CLOCK_TICKS_PER_MS;
   timerOut->lastUpdatedTime = GetCurrentTime();
   /* If the initial value is 0, then we assume that the timer is
      actually disabled. */
   if (initValueIn > 0)
   {
      timerOut->expired = FALSE;
   }
   else
   {
      timerOut->expired = TRUE;
   }
}

/*****************************************************************
Function:  UpdateMsTimer
Returns:   None
Reference: None
Purpose:   To Update a timer value.
Comments:  Get CurrentTime. See the difference between cur time
           and last updated time. Subtract that qty from timer
           value. Make sure that the timer value is not
           reduced below 0 (that will make it a big positive).
*****************************************************************/
void UpdateMsTimer(MsTimer *timerInOut)
{
   uint32 curTime, elapsedTime;

   if (timerInOut->curTimerValue == 0)
   {
      return; /* Already expired. */
   }

   /* Since the timer values are unsigned, C gurantees that simple
      subtraction will give the right value even if the timer
      has already wrapped. We just have to make sure that we
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      update the timer within one cycle of the clock. For 32
      bit clock, this is long enough and hence no problem. */
   curTime     = GetCurrentTime();
   elapsedTime = curTime - timerInOut->lastUpdatedTime;
   if (elapsedTime >= timerInOut->curTimerValue)
   {
      timerInOut->curTimerValue = 0;
   }
   else
   {
      timerInOut->curTimerValue -= elapsedTime;
   }

   timerInOut->lastUpdatedTime = curTime;
}

/*****************************************************************
Function:  MsTimerExpired
Returns:   TRUE if the timer has expired. FALSE if the timer has
           not expired or it has expired but has already been
           reported as expired.
Reference: None
Purpose:   To update given timer and test if it expired.
Comments:  None
*****************************************************************/
Boolean MsTimerExpired(MsTimer *timerInOut)
{
   if (timerInOut->curTimerValue > 0)
   {
      UpdateMsTimer(timerInOut); /* Not expired. First update it. */
   }
   if (timerInOut->expired)
   {
      return(FALSE); /* Already expired. */
   }
   if (timerInOut->curTimerValue == 0)
   {
      /* First time expiry. */
      timerInOut->expired = TRUE; /* Remember this expiry */
      return(TRUE);
   }
   return(FALSE); /* Not expired yet */
}


/*****************************************************************
Function:  NodeReset
Returns:   None
Reference:
Purpose:   Initialization of node data structures.
Comments:
*****************************************************************/
void NodeReset(Boolean firstReset)
{
   void APPReset(void), TCSReset(void), TSAReset(void),
        NWReset(void),  LKReset(void),  PHYReset(void);
   void AppReset(void);
   uint32 seed;
#ifndef SIMULATION
   uint32 *StackTimerInit(void);
#endif
   int i;

#ifndef SIMULATION
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   if (!firstReset)
   {
      PHYDisableSPMIsr();
   }
#endif

   /* Init variables that are not in EEPROM */
   memset(&nmp->stats, 0, sizeof(StatsStruct));
   if (firstReset)
   {
      nmp->errorLog        = NO_ERRORS;
   }
   nmp->pxyData.pxyType  = -1;
   gp->errorMsgIndex    = 0;
   gp->errorMsgNumber   = 0;
   gp->softwareTime     = 0;
   gp->prevPinState[0]  = 0;

#ifdef SIMULATION
   gp->currentTime      = &gp->softwareTime;
#else
   if (firstReset)
   {
      gp->currentTime      = StackTimerInit();
   }
#endif
   /* A node in soft off-line state should go on-line state */
   if (eep->readOnlyData.nodeState == CNFG_ONLINE && gp->appPgmMode == OFF_LINE)
   {
      gp->appPgmMode = ON_LINE; /* Normal state. on-line. */
   }

   /* If a node is reset while in unconfigured state, it will come back in
      offline mode when asked to go configured later. */
   if (NodeUnConfigured())
   {
      gp->appPgmMode = OFF_LINE;
   }

   /* First, Let each layer determine the address of all its
      data strcutures */
   gp->mallocUsedSize = 0;


   APPReset(); /* Application Layer */
   if (!gp->resetOk)
   {
      return;
   }
   TCSReset();
   if (!gp->resetOk)
   {
      return;
   }
   TSAReset();
   if (!gp->resetOk)
   {
      return;
   }
   NWReset();
   if (!gp->resetOk)
   {
      return;
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   LKReset();
   if (!gp->resetOk)
   {
      return;
   }
   PHYReset();

   if (!gp->resetOk)
   {
      return;
   }

   /* Call this function last as this function can call functions
      that will use data structures that should already exist. */
   AppReset(); /* Application Program Reset */
   if (!gp->resetOk)
   {
      return;
   }

#ifndef SIMULATION
   if (firstReset)
   {
      PHYInitSPM();
   }
   else
   {
      PHYEnableSPMIsr();
   }
#endif

   if (firstReset)
   {
      gp->prevChallenge[i] = 0; /* Init prevRand. Don't change during */
                                /* other reset. */
   }

   /* Init Seed to some unpredictable value at start */
   seed = 0;
#ifndef SIMULATION
   seed = GetCurrentTime(); /* Current Value of Timer */
#endif
   srand((unsigned int) seed);

   if (nmp->resetCause == EXTERNAL_RESET || nmp->resetCause == POWER_UP_RESET)
   {
      SetMsTimer(&gp->tsDelayTimer, TS_RESET_DELAY_TIME);
   }
   else
   {
      SetMsTimer(&gp->tsDelayTimer, 0); /* Disable */
   }
   gp->resetNode        = FALSE;
}

/*****************************************************************
Function:  InitEEPROM
Returns:   None
Reference: None
Purpose:   To initialize the EEPROM data items based on constants
           in custom.h and values set in custom.c
Comments:  Incomplete Initialization. Make sure it has the var you
           want or else add it here or in custom.h or custom.c
                depending on where it fits.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
*****************************************************************/
void    InitEEPROM(void)
{
   int i;
   char *p;

   /* Init the entire readOnlyData to 0 first. */
   memset(&eep->readOnlyData, 0, sizeof(eep->readOnlyData));
   /* Init the entire configData to 0 first. */
   memset(&eep->configData, 0, sizeof(eep->configData));

   /* Init Based on custom.h and default values */
   eep->readOnlyData.modelNum         = MODEL_NUM;
   eep->readOnlyData.minorModelNum    = MINOR_MODEL_NUM;
   eep->readOnlyData.checkSum         = 0;

   eep->readOnlyData.nvFixed[0] = 0xFF; /* not useful */
   eep->readOnlyData.nvFixed[1] = 0xFF;

   eep->readOnlyData.runWhenUnconf    = RUN_WHEN_UNCONF;
   eep->readOnlyData.nvCount          = 0;
   /* MIP uses 0xFFFF for snvtStruct field. p 9-8 */
   eep->readOnlyData.snvtStruct[0]    = 0xFF;
   eep->readOnlyData.snvtStruct[1]    = 0xFF;
   eep->readOnlyData.nodeState        = CNFG_ONLINE;
   /* NUM_ADDR_TBL_ENTRIES can be larger than 15, but
      addressCnt is set to min(15, NUM_ADDR_TBL_ENTRIES).
      The remaining entries are not seen by the lonbuilder tool */
   eep->readOnlyData.addressCnt       =
      (NUM_ADDR_TBL_ENTRIES <= 15)?NUM_ADDR_TBL_ENTRIES:15;
   eep->readOnlyData.receiveTransCnt  =
      (RECEIVE_TRANS_COUNT < 16)?RECEIVE_TRANS_COUNT-1:15;
   eep->readOnlyData.appOutBufSize    = APP_OUT_BUF_SIZE;
   eep->readOnlyData.appInBufSize     = APP_IN_BUF_SIZE;
   eep->readOnlyData.nwOutBufSize     = NW_OUT_BUF_SIZE;
   eep->readOnlyData.nwInBufSize      = NW_IN_BUF_SIZE;
   eep->readOnlyData.nwOutBufPriCnt   = NW_OUT_PRI_Q_CNT;
   eep->readOnlyData.appOutBufPriCnt  = APP_OUT_PRI_Q_CNT;
   eep->readOnlyData.appOutBufCnt     = APP_OUT_Q_CNT;
   eep->readOnlyData.appInBufCnt      = APP_IN_Q_CNT;
   eep->readOnlyData.nwOutBufCnt      = NW_OUT_Q_CNT;
   eep->readOnlyData.nwInBufCnt       = NW_IN_Q_CNT;
   eep->readOnlyData.msgTagCnt        = 0;

   eep->readOnlyData.readWriteProtect = READ_WRITE_PROTECT;
   eep->readOnlyData.txByAddress      = 0;
   eep->readOnlyData.aliasCnt         = 0; /* Host based node */


   /* Initialize configData */
   eep->configData.channelId           = 0;
   eep->configData.commClock           = 3;
   eep->configData.inputClock          = 5;
   eep->configData.commType            = SPECIAL_PURPOSE;
   eep->configData.commPinDir          = 0x1E; /* 0x17 if wake-up pin is input */
   eep->configData.reserved[0]         = 0x00; /* for special purpose mode. */
   eep->configData.reserved[1]         = 0x3F; /* packet_cycle */
   eep->configData.reserved[2]         = 0xA6; /* beta2 control */
   eep->configData.reserved[3]         = 0x77; /* xmit_interpacket */
   eep->configData.reserved[4]         = 0x67; /* recv_interpacket */
   eep->configData.nodePriority        = 1; /* 0-255. 0 => no priority slot. */
   eep->configData.channelPriorities   = 8; /* 0-255 */
   eep->configData.param.xcvrParams[0] = 0x0e;
   eep->configData.param.xcvrParams[1] = 0x01;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   eep->configData.param.xcvrParams[2] = 0;
   eep->configData.param.xcvrParams[3] = 0;
   eep->configData.param.xcvrParams[4] = 0;
   eep->configData.param.xcvrParams[5] = 0;
   eep->configData.param.xcvrParams[6] = 0;
   /* dirParams only used for direct mode not special purpose mode */
   /* eep->configData.param.dirParams.bitSyncThreshHold = 1; */
   eep->configData.nonGroupTimer       = NON_GROUP_TIMER;
   eep->configData.nmAuth              = NM_AUTH;
   eep->configData.preemptionTimeout   = 0;

   /* Initializaiton based on custom.c */
   memcpy(eep->readOnlyData.uniqueNodeId, cp->uniqueNodeId, UNIQUE_NODE_ID_LEN);
   eep->readOnlyData.twoDomains = cp->twoDomains;
   memcpy(eep->readOnlyData.progId, cp->progId, ID_STR_LEN);
   memcpy(eep->configData.location, cp->location, LOCATION_LEN);
   for (i = 0; i <= cp->twoDomains; i++)
   {
      eep->domainTable[i].len = cp->len[i];
      memcpy(eep->domainTable[i].domainId, cp->domainId[i],
                  cp->len[i]);
      eep->domainTable[i].subnet = cp->subnet[i];
      eep->domainTable[i].node   = cp->node[i];
      eep->domainTable[i].cloneDomain = 1;
      memcpy(eep->domainTable[i].key, cp->key[i], AUTH_KEY_LEN);
   }
   /* Init Address Table based on custom.c */
   for (i = 0; i < NUM_ADDR_TBL_ENTRIES; i++)
   {
      memcpy(&eep->addrTable[i], &cp->addrTbl[i], 5);
   }

   /* Init Alias Table based in custom.c */
   /* Since C initializes missing elements with 0 we use
      any non-zero value for hostPrimary field to indicate that
      we did initialize an entry. We don't need 0 anyway for
      hostPrimary as we can use primary for such entries. */

   for (i = 0; i < NV_ALIAS_TABLE_SIZE; i++)
   {
      memcpy(&nmp->nvAliasTable[i], &cp->aliasTbl[i], 6);
   }

   nmp->nvTableSize  = 0;

   /* Initialize Alias Tables that are not initialized in custom.h */
   for (i = 0; i < NV_ALIAS_TABLE_SIZE; i++)
   {
      /* Init only those that are not given meaningful values
         in custom.h */
      if (nmp->nvAliasTable[i].hostPrimary != 0)
      {
         continue; /* Skip this as it was initialized in custom.c */
      }

      p = (char *)&nmp->nvAliasTable[i];
      *p        = (char) 0x70;
      *(p + 1)  = (char) 0x00;
      *(p + 2)  = (char) 0x0F;
      *(p + 3)  = (char) 0xFF;
      *(p + 4)  = (char) 0xFF;
      *(p + 5)  = (char) 0xFF;
   }
}
```


--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

```
/*****************************************************************
Function:  GetPrimaryIndex
Returns:   The primary index of the given variable.
Reference: None
Purpose:   To compute the primary index
Comments:  Given index can be either primary or alias.
*****************************************************************/
int16 GetPrimaryIndex(int16 nvIndex)
{
   int16   primaryIndex;
   uint16  nvAliasTableSize;

   /* Fetch the alias table size */
   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

   if (nvIndex < 0 || nvIndex >= nmp->nvTableSize + nvAliasTableSize)
   {
      return(-1); /* Bad index value. */
   }

   if (nvIndex < nmp->nvTableSize)
   {
      primaryIndex = nvIndex; /* Primary index itself. */
   }
   else
   {
      nvIndex = nvIndex - nmp->nvTableSize; /* Get alias table index. */
      /* Compute the primary index. */
      primaryIndex = nmp->nvAliasTable[nvIndex].primary;
      if (primaryIndex == 0xFF)
      {
         primaryIndex = nmp->nvAliasTable[nvIndex].hostPrimary;
      }
      if (primaryIndex >= nmp->nvTableSize)
      {
         return(-1); /* Bad index in alias structure. */
      }
   }
   return(primaryIndex);
}

/*****************************************************************
Function:  GetNVStructPtr
Returns:   Pointer to the network variable structure.
Reference: None
Purpose:   To compute the pointer to the network variable structure.
Comments:  The given index can be either primary or alias.
*****************************************************************/
NVStruct *GetNVStructPtr(int16 nvIndexIn)
{
   uint16  nvAliasTableSize;

   /* Fetch the alias table size */
   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

   if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize + nvAliasTableSize)
   {
      return(NULL); /* Bad index value. */
   }

   if (nvIndexIn < nmp->nvTableSize)
   {
      return(&nmp->nvConfigTable[nvIndexIn]);
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   }

   return(&nmp->nvAliasTable[nvIndexIn - nmp->nvTableSize].nvConfig);
}

/*****************************************************************
Function:  CheckSum4
Returns:   4 bit checksum of a given data.
Reference: None
Purpose:   To Compute the checksum of an array of bytes of
           a given length. The check sum is the successive
           application of exclusive or of successive 4 bits.
Comments:  None
*****************************************************************/
uint8   CheckSum4(void *dataIn, uint16 lengthIn)
{
   unsigned char *p;
   uint16 i;
   uint8 result = 0; /* Final checksum in low order 4 bits. */

   p = dataIn;
   for (i = 0; i < lengthIn; i++)
   {
      result = result ^ (*p >> 4); /* exclusive or with high order
                                      4 bits */
      result = result ^ (*p & 0x0F); /* With low order 4 bits */
      p++;
   }
   return(result);
}

/*****************************************************************
Function:  CheckSum8
Returns:   8 bit checksum of a given data
Reference: None
Purpose:   To compute the checksum of an array of bytes of
           a given length. The check sum is the successive
           application of exclusive or of successive 4 bits.
Comments:  None
*****************************************************************/
uint8   CheckSum8(void *dataIn, uint16 lengthIn)
{
   unsigned char *p;
   uint16 i;
   uint8 result = 0; /* Final checksum */

   p = dataIn;
   for (i = 0; i < lengthIn; i++)
   {
      result = result ^ *p;
      p++;
   }
   return(result);
}

/*****************************************************************
Function:  ComputeConfigCheckSum
Returns:   The configuration checksum.
Reference: None
Purpose:   To compute the configuration checksum.
Comments:  Do not include reserved portion as it does not have
           any configuration data. Since, reference implementation
           has nv tables in nmp structure rather than EEPROM,
           we should include them in the computation.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
*****************************************************************/
uint8 ComputeConfigCheckSum(void)
{
   uint8 checkSum;
   uint16 size;

   size = (char *)&eep->addrTable[NUM_ADDR_TBL_ENTRIES] - (char *)eep;
   checkSum = CheckSum8(eep, size);
   size = (char *)&nmp->nvFixedTable[NV_TABLE_SIZE] -
          (char *)&nmp->nvConfigTable[0] + 2;
   checkSum = checkSum ^ CheckSum8(&nmp->nvConfigTable, size);
   return(checkSum);
}

/*****************************************************************
Function:  IOChanges
Returns:   TRUE if the state of input pin changed.
Reference: None
Purpose:   To determine whether there is a state change in input
           Pin 0.
Comments:  None
*****************************************************************/
Boolean IOChanges(uint8 pinNumberIn)
{
   if (pinNumberIn != 0)
   {
      return(FALSE); /* Only Input Pin 0 is supported for now */
   }

   if (gp->prevPinState[0] == 0 && gp->ioInputPin0)
   {
      /* Prev state = released  curstate = pressed */
      gp->prevPinState[0] = 1;
      return(TRUE);
   }
   if (gp->prevPinState[0] == 1 && !gp->ioInputPin0)
   {
      /* prevstate = pressed and curstate = released */
      gp->prevPinState[0] = 0;
      return(TRUE);
   }
   return(FALSE);
}

Boolean IsTagBound(uint8 tagIn)
{
   return(tagIn < nmp->snvt.mtagCount &&
          tagIn < NUM_ADDR_TBL_ENTRIES &&
          eep->addrTable[tagIn].addrFormat != UNBOUND);
}

/*****************************************************************
Function: IsNvBound
Returns:  TRUE if the variable is bound. FALSE otherwise.
Purpose:  To determine if a primary variable is bound or not.
Comment:  A variable is bound if its address index is not 0xF
          or there is an alias attached to it whose address
          index is not 0xF.
*****************************************************************/
Boolean IsNVBound(int16 nvIndexIn)
{
   uint16 i;
   int16 primaryIndex;
   uint16 addrIndex;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   uint16 nvAliasTableSize;

   if (nvIndexIn < 0 || nvIndexIn >= nmp->nvTableSize)
   {
      return(FALSE); /* not an index of a primary network variable */
   }

   /* If the primary has a valid address table index and the address
      table entry is not unbound, then the variable is bound */
   addrIndex = nmp->nvConfigTable[nvIndexIn].nvAddrIndex;
   if ( addrIndex != 0x0F &&
        (eep->addrTable[addrIndex].addrFormat != UNBOUND ||
         eep->addrTable[addrIndex].turnaEntry.turnaround == 1) )
   {
      return(TRUE);
   }

   nvAliasTableSize = nmp->snvt.aliasPtr->hostAlias;

   /* Primary is not bound. See if there is an alias for this variable
      that is bound. */
   for (i = 0; i < nvAliasTableSize; i++)
   {
      primaryIndex = GetPrimaryIndex(i + nmp->nvTableSize);
      addrIndex    = nmp->nvAliasTable[i].nvConfig.nvAddrIndex;
      /* If the alias matches the primary, has a valid address table
         index and the address table entry is not UNBOUND, then
         the primary variable is bound */
      if (primaryIndex == nvIndexIn &&
          addrIndex != 0x0F         &&
          (eep->addrTable[addrIndex].addrFormat != UNBOUND ||
           eep->addrTable[addrIndex].turnaEntry.turnaround == 1) )
      {
         return(TRUE);
      }
   }
   return(FALSE); /* No alias entry for this primary that is bound. */
}

/*****************************************************************************
Function: AppPgmRuns
Returns:  TRUE if the application program is running on the node.
          FALSE otherwise.
Purpose:  To determine whether the application program is running or not.
          This is used to determine whether to deliver messages, responses,
          and events to the application. Also used to determine whether
          to call DoApp or not.
*****************************************************************************/
Boolean AppPgmRuns(void)
{
   /* Normal Mode. Configured and running. */
   if (eep->readOnlyData.nodeState == CNFG_ONLINE &&
       gp->appPgmMode == ON_LINE)
   {
      return(TRUE);
   }

   /* Unconfigured and running. */
   if (eep->readOnlyData.nodeState == APPL_UNCNFG &&
       eep->readOnlyData.runWhenUnconf            &&
       gp->appPgmMode == ON_LINE)
   {
      return(TRUE);
   }
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   return(FALSE);
}

/************************************************************************
Function: NodeConfigured
Returns:  TRUE if the node configured is valid.
Purpose:  To determine whether currently the node is configured.
************************************************************************/
Boolean NodeConfigured(void)
{
   return(eep->readOnlyData.nodeState == CNFG_ONLINE  ||
          eep->readOnlyData.nodeState == CNFG_OFFLINE);
}

/************************************************************************
Function: NodeUnConfigured
Returns:  TRUE if the node is configured.
Purpose:  To determine whether currently node is unconfigured.
************************************************************************/
Boolean NodeUnConfigured(void)
{
   return(eep->readOnlyData.nodeState == APPL_UNCNFG  ||
          eep->readOnlyData.nodeState == NO_APPL_UNCNFG);
}

/*************************End of node.c*************************/
```

A.13 Include Files for the Reference Implementation

```
/*********************************************************************
  File:           cnp_1.h

  Version:        1.7

  Purpose:        To define constants and types that are needed
                  by all files. Most .c files will include
                  cnp_1.h either directly or indirectly.

  Note:           Reference implementation does not support
                  special purpose nodes such as routers, repeaters
                  etc. Additional code is required to implement
                  these nodes.

  To Do:          None.
*********************************************************************/
#ifndef _CNP_1_H
#define _CNP_1_H

/*-------------------------------------------------------------------------
Section: Includes
-------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------------
Section: Macro Definitions
-------------------------------------------------------------------------*/
#define MIN(x,y) (((x)<(y))?(x):(y))
#define MAX(x,y) (((x)>(y))?(x):(y))
#define INCR_STATS(x) ((x < (uint16)0xFFFF)?x++:x)

/*-------------------------------------------------------------------------
Section: Constant Definitions
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
------------------------------------------------------------------------------*/
/* Reference implementation supports the use of alternate structures such
   as msg_in, msg_out etc for ease of porting application
   programs. However, the disadvantage is the extra copy needed to
   use these structures. Uncomment this constant if you do not want
   to use the alternate structures (i.e you prefer to use reference implementation
   structures that use a different naming convention.) */
#define ALTERNATE_STRUCTURES_NEEDED

#define UNIQUE_NODE_ID_LEN 6 /* Length of the Unique Node Id.         */
#define ID_STR_LEN       8   /* Length of the program id string.      */
#define AUTH_KEY_LEN     6   /* Length of the authentication key.     */
#define DOMAIN_ID_LEN    6   /* Maximum length for a domain id.       */
#define LOCATION_LEN     6   /* Maximum length for location string.   */
#define NUM_COMM_PARAMS  7   /* Max # of parameters for a tranceiver. */
#define PROTOCOL_VERSION 0   /* 0 for reference implementaion.        */
#define MAX_DOMAINS      2   /* Maximum # of domains allowed.         */

/* Set the size of the array to log error messages from the protocol stack.
   The error messages wrap around, if there are too many errors.
   Errors seldom happen. So, there is no need for this to be too large. */
#define ERROR_MSG_SIZE  1000     /* 20 messages each with 50 chars */

#define FLEX_DOMAIN         2    /* Indicates that the message was received
                                    in flex domain when domain index is 2 */
#define COMPUTE_DOMAIN_INDEX 3   /* When application layer communicates
                                    with transport or session layer,
                                    the domainIndex for the outgoing message
                                    can be either set by the application layer
                                    or computed by transport or session layer
                                    based on the destination address.
                                    This value is used only in TSASenParam
                                    structure. */
#define MAX_GROUP_NUMBER 63    /* Maximum number of a node in a group */

/*------------------------------------------------------------------------------
Section: Type Definitions
------------------------------------------------------------------------------*/
/* The following type definitions need to be changed based on the
   compiler used. The rest of the files should use only int8,
   int16, uint8 etc. Application programs should use
   nint (8-bit int), nlong etc as much as possible. */
typedef char                int8;
typedef short int           int16;
typedef long int            int32;
typedef unsigned char       uint8;
typedef unsigned short int  uint16;
typedef unsigned long int   uint32;

/* Typical definitions for int long etc. */
typedef int8                nshort;
typedef int8                nint;
typedef uint8               nuint;
typedef uint8               nushort;
typedef int16               nlong;
typedef uint16              nulong;

typedef unsigned char Byte;
typedef unsigned char Bits;
typedef int16         MsgTag; /* Lots of Tags!!! */

/* This type definition may need #ifndef macro wrapped
   around when porting to other platforms. */
typedef enum
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
{
   FALSE = 0,
   TRUE  = 1
} Boolean;

typedef Boolean boolean; /* For nc compatibility. */

typedef enum
{
   NOBIND = 0,
   NON_BINDABLE = 0,   /* Same as NOBIND. */
   BIND   = 1,
   BINDABLE = 1        /* Same as BIND.   */
} BindNoBind;

typedef enum
{
   NV_INPUT  = 0,
   NV_OUTPUT = 1
} NVDirection;

/* Address Types. */
typedef enum
{
   UNBOUND          = 0,

   SUBNET_NODE      = 1,

   UNIQUE_NODE_ID   = 2,

   BROADCAST        = 3,

   MULTICAST        = 4,

   MULTICAST_ACK    = 5,

   BROADCAST_GROUP  = 0x23 /* Used for broadcast req/resp requiring
                              up to N responses to be delivered. */
} AddrMode;

/* Reference: page 9-49 of Tech Device Data Book. Revision 2.
   In reference implementation, the application is always loaded
   when downloading the code into the system. There is no provision
   to download application program from management tools. However,
   application can be placed in offline by calling GoOffline fn.
   Thus the node state NO_APPL_UNCNFG is not possible.
   gp->appPgmMode indicates the state of the application program. */
typedef enum
{
   /* For nodeState */
   APPL_UNCNFG     = 2, /* Application is loaded but conf is not */
   NO_APPL_UNCNFG  = 3, /* Application is not loaded yet or bad  */
   CNFG_ONLINE     = 4, /* Normal operation mode                 */
   CNFG_OFFLINE    = 6, /* same as hard offline state            */
   HARD_OFFLINE    = 6,

   SOFT_OFFLINE    = 0xC,   /* For reporting purpose             */
   CNFG_BYPASS     = 0x8C   /* Not supported in reference imp.   */
} NodeState;

/* These constants are used to represent the mode when the node is
   configured. */
typedef enum
{
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   OFF_LINE    = 0,  /* For soft off-line */
   ON_LINE     = 1,  /* For normal mode   */
   NOT_RUNNING = 2   /* For hard-offline. Not used in reference impl.
                        unless commanded to enter this state. */
} ConfigMode;

/* The order of the first 4 items is important as it is used by
   network layer to determine the type of PDU. */
typedef enum
{
   /* Do not change the order. The values are sent across the network */
   TPDU_TYPE,
   SPDU_TYPE,
   AUTHPDU_TYPE,
   APDU_TYPE,

   /* Something extra for internal use of the protocol stack */
   NPDU_TYPE,
   LPDU_TYPE,
} PDUType;

/* Services offered to application program. These are not sent over
   the network */
typedef enum
{
   ACKD,       /* Transport Layer */
   UNACK_RPT,  /* Transport Layer */
   UNACKD,     /* Network   Layer */
   REQUEST,    /* Session   Layer */

   RESPONSE    /* Session   Layer. Used by resp_send function */
} ServiceType;

/* Tranceiver types. Only the constants are used */
typedef enum
{
   BLANK            = 0,
   SINGLE_ENDED     = 1,
   SPECIAL_PURPOSE  = 2,
   DIFFERENTIAL     = 5
} TranceiverType;

/* Return status for all functions. */
typedef enum
{
   SUCCESS               = 0,
   FAILURE               = 1,
   INVALID               = 2
} Status;

/* Reset cause */
typedef enum
{
   POWER_UP_RESET       = 0x01,
   EXTERNAL_RESET       = 0x02,
   WATCHDOG_RESET       = 0x0C,
   SOFTWARE_RESET       = 0x14,
   CLEARED              = 0x00
} ResetCause;

/* Turn alignment on so that structures packed tightly */
#pragma maxalign(1)

/* Software Timer definition. */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
typedef struct
{
   uint32 curTimerValue;    /* Number of clock ticks left in timer.      */
   uint32 lastUpdatedTime;  /* The time when the timer was last updated. */
   Boolean expired;         /* TRUE => it has already expired.           */
} MsTimer;


typedef uint8  TransNum;    /* For transaction numbers.                 */
typedef uint16 RequestId;   /* For matching responses with requests.    */

typedef Byte BroadcastAddress;
typedef Byte MulticastAddress;

typedef struct
{
   Byte  subnet;
   Bits  selField   :1;  /* Used by network layer for 2b. */
   Bits  node       :7;
} SubnetAddress;

typedef struct
{
   MulticastAddress group;
   Byte member;
} GroupAddress;

typedef struct
{
   SubnetAddress subnetAddr;
   GroupAddress  groupAddr;   /* Acknowledging group member. */
} MulticastAckAddress;

typedef struct
{
   Byte   subnet;  /* For routing purpose. */
   Byte   uniqueId[UNIQUE_NODE_ID_LEN];
} UniqueNodeIdAddress;


/*****************************************************************************
   DestinationAddress is used to indicate network layer that address
   mode is used to send the message. The destination address
   is always one of the five possibilities.
   domainIndex indicates the domain table to use to determine
   domain length and domain id.
   If domainIndex = FLEX_DOMAIN, then it is flexdomain.
   In this case, src subnet/node is 0/0.
*****************************************************************************/
typedef struct
{
   Byte  domainIndex;
   uint8 flexDomainLen; /* Use if domainIndex == FLEX_DOMAIN */
   Byte  flexDomainId[DOMAIN_ID_LEN]; /* Use if domainIndex == FLEX_DOMAIN */
   AddrMode addressMode;
   union {
      BroadcastAddress       addr0;
      MulticastAddress       addr1;
      SubnetAddress          addr2a;
      MulticastAckAddress    addr2b;
      UniqueNodeIdAddress    addr3;
   } addr;
} DestinationAddress;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/*****************************************************************************
   SourceAddress is used by network layer to indicate to upper levels
   who sent the message and what mode was used.
   Thus this structure is used only for receiving messages.
   domainIndex is used to respond back in the domain in which
   the message was received.
*****************************************************************************/
typedef struct
{
   SubnetAddress        subnetAddr;   /* Subnet of source node. */
   AddrMode             addressMode;  /* What mode used. */
   Byte                 domainIndex;  /* 0 or 1 or FLEX_DOMAIN (i.e 2) */
   uint8                flexDomainLen;/* Used if domainIndex == FLEX_DOMAIN. */
   Byte  flexDomainId[DOMAIN_ID_LEN]; /* Used if domainIndex == FLEX_DOMAIN. */
   /* group is used only if addressMode is MULTICAST.
      It is the group of the source node sending this message. */
   MulticastAddress     group;
   /* ackNode is used only if addressMode is MULTICAST_ACK.
      It is the destSubnet used and the group of the node sending the ack
      or response. */
   MulticastAckAddress  ackNode;
   /* Destination subnet for broadcast messages. */
   Byte broadcastSubnet;
} SourceAddress;

#pragma maxalign()

#endif   /* #ifndef _CNP_1 */
/*********************************cnp_1.h*********************************/
/*****************************************************************
  File:           node.h  (node.h)

  Version:        1.7

  Purpose:        To define all the type definitions needed by all
                  the upper layers of the stack.
                  To also define all function to interface with
                  some of these data structures as specified in
                  the Device Data Book.

  Note:           None

  To Do:          None.
*****************************************************************/
#ifndef _NODE_H
#define _NODE_H

/*---------------------------------------------------------------------------
Section: Includes
---------------------------------------------------------------------------*/
#include <cnp_1.h>
#include <api.h>
#include <queue.h>

/*---------------------------------------------------------------
Section: Constant Definitions
---------------------------------------------------------------*/
/* Define the size of the table maintained by the transaction control
   sublayer that keeps track of each possible destination address
   in packets sent to make sure that we don't assign the same tid as in the
   last transaction to that destination. This constant is needed here
   so that we can use it in the type definition of protocol stack data. */
#define TID_TABLE_SIZE 100
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* Init this constant to reflect the number of clock ticks in
   the 32 bit clock for every 1 ms. Some computation may be
   needed to determine this number. For simulation, we consider
   each clock tick as equivalent to 1 ms. */
#ifdef SIMULATION
#define CLOCK_TICKS_PER_MS 1
#else
#define CLOCK_TICKS_PER_MS 0x61
#endif

/* Given a valid primary index of a network variable, get its address */
#define NV_ADDRESS(i) (nmp->nvFixedTable[i].nvAddress)

/* Given a valid primary index of a network variable, get its length */
#define NV_LENGTH(i)  (nmp->nvFixedTable[i].nvLength)

/* Given a valid primary index of a network variable, check if it is sync */
#define NV_SYNC(i)    (nmp->nvFixedTable[i].nvSync)

/*-------------------------------------------------------------------
Section: Type Definitions
-------------------------------------------------------------------*/
/* Reference: Tech Device Data: Page 9-6. */

/* Turn off alignment by compiler to make sure that the structure sizes
   are as we intend them to be with no padding */
#pragma maxalign(1)
typedef struct
{
   Byte  uniqueNodeId[UNIQUE_NODE_ID_LEN]; /* 48-bit unique ID.             */
   Byte  modelNum;                  /* Model Number for Ref. Impl.          */
   Bits  checkSum         : 4;      /* For Unique Node ID.                  */
   Bits  minorModelNum    : 4;      /* 0-128.                               */
   Byte  nvFixed[2];                /* Location of nv fixed table.          */
   Bits  readWriteProtect : 1;      /* read+write protect flag.             */
   Bits  runWhenUnconf    : 1;      /* 1=> Application runs.                */
   Bits  nvCount          : 6;      /* 0 for reference implementation.      */
   Byte  snvtStruct[2];             /* 0xFFFF for reference implementation. */
   char  progId[ID_STR_LEN];        /* Program Id string.                   */
   Bits                   : 1;      /* nvProcessingOff not used in Ref. Impl.*/
   Bits  twoDomains       : 1;      /* 1 if node is in 2 domains.           */
   Bits                   : 1;      /* explicitAddr not used in Ref. Impl.  */
   Bits                   : 1;      /* Unused.                              */
   Bits                   : 1;      /* Unused.                              */
   Bits  nodeState        : 3;      /* Node State. See cnp_1.h              */
   Bits  addressCnt       : 4;      /* # of entries in address table.       */
   Bits                   : 4;      /* Unused.                              */
   Bits                   : 4;      /* Unused.                              */
   Bits  receiveTransCnt  : 4;      /* RR Cnt = this field + 1              */
   Bits  appOutBufSize    : 4;      /* Special Size Encoding.               */
   Bits  appInBufSize     : 4;      /* Special Size Encoding.               */
   Bits  nwOutBufSize     : 4;      /* Special Size Encoding.               */
   Bits  nwInBufSize      : 4;      /* Special Size Encoding.               */
   Bits  nwOutBufPriCnt   : 4;      /* Special Count Enconding.             */
   Bits  appOutBufPriCnt  : 4;      /* Special Count Enconding.             */
   Bits  appOutBufCnt     : 4;      /* Special Count Enconding.             */
   Bits  appInBufCnt      : 4;      /* Special Count Enconding.             */
   Bits  nwOutBufCnt      : 4;      /* Special Count Enconding.             */
   Bits  nwInBufCnt       : 4;      /* Special Count Enconding.             */
   Byte  reserved0;                 /* Unused                               */
   Byte  reserved1[2];              /* Unused.                              */
   Byte  reserved2[3];              /* Unused.                              */
   Bits                   : 6;      /* Unused.                              */
   Bits  txByAddress      : 1;      /* 0 in reference implementation.       */
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   Bits                   : 1;      /* Unused.                              */
   Bits                   : 2;      /* Unused.                              */
   Bits aliasCnt          : 6;      /* 0 in reference implementation.       */
   Bits msgTagCnt         : 4;      /* 0 in reference implementation.       */
   Bits                   : 4;      /* Unused.                              */
   Byte reserved3[3];               /* Unused.                              */
} ReadOnlyDataStruct;

/* Reference: Tech Device Data Book. Page 9-11. */
typedef struct
{
   Byte domainId[DOMAIN_ID_LEN];
   Byte subnet;
   Bits cloneDomain : 1;
   Bits node        : 7;
   Byte len;
   Byte key[AUTH_KEY_LEN];
} DomainStruct;

typedef struct
{
   AddrMode  addrMode;             /* UNBOUND (0) => turnaround.   */
   Byte      turnaround;           /* 1=>turnaround; 0=>not in use.*/
   Bits      unacdkRptTimer  : 4;  /* unackd_rpt timer.            */
   Bits      retryCount      : 4;  /* retry count.                 */
   Bits                      : 4;  /* UNUSED.                      */
   Bits      txTimerIndex    : 4;  /* transmit timer index.        */
} TurnaroundStruct;

/* Other types used in the following structure are defined in api.h. */
typedef union
{
   uint8             addrFormat; /* Format of entry. */
   GroupAddrMode     groupEntry;
   SNodeAddrMode     snodeEntry;
   BcastAddrMode     bcastEntry;
   TurnaroundStruct  turnaEntry;
} AddrTableEntry;

/* Reference: Tech Device Data: Page 9-24 */
typedef struct
{
   Bits collisionDetect    :1;
   Bits bitSyncThreshHold  :2; /* Encoded. See page 9-28 */
   Bits filter             :2;
   Bits hysteresis         :3;
   Bits unUsed             :6;
   Bits cdTail             :1;
   Bits cdPreamble         :1;
} DirectParamStruct;

typedef struct
{
   int16       channelId;              /* Id of channel assigned */
   Byte        location[LOCATION_LEN]; /* 6 byte ascii string    */
   Bits        commClock    :5;        /* bit rate ratio         */
   Bits        inputClock   :3;        /* input clk osc freq     */
   Bits        commType     :3;        /* type of receiver       */
   Bits        commPinDir   :5;        /* dir comm port pins     */
   Byte        reserved[5];
   Byte        nodePriority;           /* priority slot used     */
   Byte        channelPriorities;      /* # of priority slots    */
   union {
      Byte              xcvrParams[NUM_COMM_PARAMS];
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      DirectParamStruct  dirParams;
   } param;
   Bits        nonGroupTimer     :4;
   Bits        nmAuth            :1; /* NetMgt msgs needs auth? */
   Bits        preemptionTimeout :3; /* free buffer wait time in
                                        preemption mode         */
} ConfigDataStruct;

/* Reference: Tech Device Data: Page 9-17 */
/* Network Variable Tables */
typedef struct
{
   Bits nvPriority     :1;  /* NV uses priority messaging       */
   Bits nvDirection    :1;  /* 1 => output var                  */
   Bits nvSelectorHi   :6;  /* Hi & Lo form NV selector         */
   Byte nvSelectorLo     ;  /* Range 0-0x3FFF                   */
   Bits nvTurnaround   :1;  /* 1 ==> bound to a nv in this node */
   Bits nvService      :2;  /* ACKD or UNACKD_RPT or UNACKD     */
   Bits nvAuth         :1;  /* 1 => NV used auth transactions   */
   Bits nvAddrIndex    :4;  /* index into addr table. 15 is spcl */
} NVStruct;

typedef struct
{
   NVStruct nvConfig;
   uint8    primary;     /* index into NV Cfg tbl. 0xFF => next fld */
   uint16   hostPrimary; /* NV cfg tbl index. For host nodes */
} AliasStruct;

typedef struct
{
   Bits nvSync    :1; /* 1 ==> Synchronous network var */
   Bits           :2; /* unused */
   Bits nvLength  :5; /* # of bytes in the Network Var */
   void *nvAddress;   /* Ptr to variable's data */
} NVFixedStruct;

/* Statistics */
typedef struct
{
   uint16 transmissionErrors;
   uint16 transmitTXFailures;
   uint16 receiveTXFull;
   uint16 lostMessages;
   uint16 missedMessages;
   uint16 layer2Received;
   uint16 layer3Received;
   uint16 layer3Transmitted;
   uint16 transmitTXRetries;
   uint16 backlogOverflow;
   uint16 lateAcknowledgements;
   uint16 collisions;
   uint8  eepromLock;
   uint16 expansion[11]; /* Room for expansion */

   /* Reference Implementation Specific Statistics */
   uint16 layer6_7MsgsRcvd;
   uint16 layer6_7RespRcvd;
   uint16 layer6_7MsgsSent;
   uint16 layer6_7RespSent;
   uint16 lateResponses;    /* No matching req */
   uint16 lateChallenges;   /* No matching transmit record */
   uint16 lateReplies;      /* No matching receive  record */
} StatsStruct;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/* yes: Yes, error is implemented in reference implementation */
/* na:  Not applicable (not implemented) in ref. imp.         */
typedef enum
{
   NO_ERRORS                        = 0,    /* yes */
   BAD_EVENT                        = 129,  /* na  */
   NV_LENGTH_MISMATCH               = 130,  /* yes */
   NV_MSG_TOO_SHORT                 = 131,  /* yes */
   EEPROM_WRITE_FAIL                = 132,  /* na  */
   BAD_ADDRESS_TYPE                 = 133,  /* yes */
   PREEMPTION_MODE_TIMEOUT          = 134,  /* na  */
   ALREADY_PREEMPTED                = 135,  /* na  */
   SYNC_NV_UPDATE_LOST              = 136,  /* na  */
   INVALID_RESP_ALLOC               = 137,  /* na  */
   INVALID_DOMAIN                   = 138,  /* yes */
   READ_PAST_END_OF_MSG             = 139,  /* na  */
   WRITE_PAST_END_OF_MSG            = 140,  /* na  */
   INVALID_ADDR_TABLE_INDEX         = 141,  /* yes */
   INCOMPLETE_MSG                   = 142,  /* na  */
   NV_UPDATE_ON_OUTPUT_NV           = 143,  /* na  */
   NO_MSG_AVAIL                     = 144,  /* na  */
   ILLEGAL_SEND                     = 145,  /* na  */
   UNKNOWN_PDU                      = 146,  /* yes */
   INVALID_NV_INDEX                 = 147,  /* yes */
   DIVIDE_BY_ZERO                   = 148,  /* na  */
   INVALID_APPL_ERROR               = 149,  /* na  */
   MEMORY_ALLOC_FAILURE             = 150,  /* yes */
   WRITE_PAST_END_OF_NET_BUFFER     = 151,  /* na  */
   APPL_CS_ERROR                    = 152,  /* na  */
   CNFG_CS_ERROR                    = 153,  /* na  */
   INVALID_XCVR_REG_ADDR            = 154,  /* na  */
   XCVR_REG_TIMEOUT                 = 155,  /* na  */
   WRITE_PAST_END_OF_APPL_BUFFER    = 156,  /* na  */
   IO_READY                         = 157,  /* na  */
   SELF_TEST_FAILED                 = 158,  /* na  */
   SUBNET_ROUTER                    = 159,  /* na  */
   AUTHENTICATION_MISMATCH          = 160,  /* yes */
   SELF_INST_SEMAPHORE_SET          = 161,  /* na  */
   READ_WRITE_SEMAPHORE_SET         = 162,  /* na  */
   APPL_SIGNATURE_BAD               = 163,  /* na  */
   ROUTER_FIRMWARE_VERSION_MISMATCH = 164   /* na  */
} ErrorLog;

/*-------------------------------------------------------------------
  NWSendParam is used to process Send Request to Network Layer
  from other layers. When other layers deposit a new PDU into the
  Output Queue, they should also fill in the details of this
  data structure to indicate the kind of request.

  altPath is set by transport/session layers for the last 2 retries.
  destAddr has the domainIndex that is used to determine the
  domainId, source subnet/node. The possible values for the
  domainIndex are 0 or 1 or 2. 0 or 1 ==> use the corresponding
  domainTable. 2 ==> use the flex domain.
-------------------------------------------------------------------*/
typedef struct
{
   DestinationAddress    destAddr; /* To give dest addr  */
   PDUType               pduType;  /* APDU, SPDU, TPDU, AuthPDU */
   MsgTag                tag;      /* used only for APDU */
   uint8                 deltaBL;  /* Upper Layers supply this */
   Boolean               altPath;
   uint16                pduSize;  /* Size of the PDU sent */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   Boolean               dropIfUnconfigured; /* drop the packet if the
                           node is unconfigured. */
} NWSendParam;

/*-------------------------------------------------------------------
  NWReceiveParam: Is used by Link Layer when it supplies an incoming
  NPDU to the network layer. This data structure contains information
  regarding the incoming NPDU.
*******************************************************************/
typedef struct
{
   Boolean               priority;   /* Was it a priority message? */
   Boolean               altPath;
   uint16                pduSize;    /* Size of NPDU in queue */
} NWReceiveParam;

/* Type Definitions for Application Layer */
typedef struct
{
   DestinType code;
   Byte data[MAX_DATA_SIZE];
} APDU;

typedef APDU *APDUPtr;

/* APPSendParam is used by API to give info to app layer
   regarding APDU that is going out. Most info is available
   in msg_out in the Application Out Buffers */

/* APPSendParam has everything in MsgOut except code and data. */
typedef struct
{
   MsgTag      tag;           /* to correlate completion codes  */
   uint8       len;           /* message length in app data     */
   Boolean     authenticated; /* TRUE if to be authenticated    */
   ServiceType service;       /* service type used to send msg  */
   RequestId   reqId;         /* Request ID for responses */
   MsgOutAddr  addr;          /* destination address (see above)*/
   Boolean     nullResponse;  /* For responses                  */
} APPSendParam;

/* Types of messages that are received by application layer */
typedef enum {
   MESSAGE    = 0,
   COMPLETION = 1,  /* Indication by Transport/Session layers */
} APIndType;


/* APPReceiveParam is used by Application Layer to figure out
   information regarding incoming APDU. */
typedef struct
{
   APIndType             indication; /* Type of APDU received */
   Boolean               success;    /* Used if indication ==
                                        COMPLETION */
   MsgTag                tag;        /* Tag for indication
                                        or for matching response */
   SourceAddress         srcAddr;    /* Who sent it?            */
   ServiceType           service;    /* Which service type?     */
   Boolean               priority;   /* Was it a priority msg?  */
   Boolean               altPath;    /* Was it sent in alt path? */
   uint16                pduSize;    /* Size of incoming APDU   */
   uint16                taIndex;    /* Turnaround var Index    */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   Boolean              auth;       /* Was it authenticated?   */
   RequestId            reqId;      /* Assigned by session to match
                                       responses later         */
} APPReceiveParam;

/* Type Definition for Transaction Control SubLayer */
typedef struct
{
   TransNum    transNum;       /* initial value = 0. Range 0..15 */
   Boolean     inProgress;     /* Is the transaction in progress */
} TransCtrlRecord;

/* Type definition for table used while assigning new TId. */
/* For more information, see tcs.h or tcs.c */
typedef struct
{
   Byte  domainId[DOMAIN_ID_LEN];
   uint8 len; /* domain length */
   AddrMode             addressMode;
   union
   {
      SubnetAddress         subnetNode;
      MulticastAddress      group;   /* group number of multicast */
      BroadcastAddress      subnet;  /* 0 if domainwide broadcast */
      Byte                  uniqueNodeId[UNIQUE_NODE_ID_LEN];
   } addr;
   MsTimer                  timer;
   TransNum                 tid;    /* Last TID Used for this addr */
} TIDTableEntry;

/* Type Definitions for transport, Session, Auth Layers */

typedef enum
{
   TRANS_CURRENT,
   TRANS_NOT_CURRENT,
   TRANS_NEW,
   TRANS_DUPLICATE
} TransStatus;

typedef enum
{
   UNUSED_TX,        /* Transmit Record is unused */
   SESSION_TX,       /* Transmit Record is used by Session */
   TRANSPORT_TX      /* Transmit Record is used by Transport */
} TXStatus;

typedef enum
{
   UNUSED_RR,       /* Receive Record is unused */
   SESSION_RR,      /* Receive Record is used by Session */
   TRANSPORT_RR     /* Receive Record is used by Transport */
} RRStatus;

typedef enum
{
   JUST_RECEIVED,   /* Msg just received. Nothing has been done    */
   DELIVERED,       /* Msg has been delivered to app layer. Receive
                       Timer has not expired yet. Waiting for Resp */
   DONE,            /* Msg delivered & a null resp received.
                       Or Msg delivered and Server received has received
                       the response.
                       Timer has not expired yet */
   AUTHENTICATED,   /* Msg has been authenticated. Not delivered   */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   AUTHENTICATING,  /* Msg is being authenticated. reply expected  */
   RESPONDED,       /* Rsp has been recvd from app layer & responded
                       at least once. Recv Timer not expired yet   */
} TransactionState;  /* Constant used by Transport & Session */

typedef struct
{
   TXStatus           status;              /* Who owns it? if not free  */
   DestinationAddress nwDestAddr;          /* Destination Address */
   Boolean            ackReceived[MAX_GROUP_NUMBER+1];
                      /* Array[0..MAX_GROUP_NUMBER] of Boolean   */
   uint8              destCount;           /* Number of destinations    */
   uint8              ackCount;            /* Or respCount              */
   TransNum           transNum;
   uint16             xmitTimerValue;
   MsTimer            xmitTimer;           /* Transmit Timer            */
   uint8              retriesLeft;         /* How many left?            */
   APDU              *apdu;                /* APDU transmitted          */
   uint16             apduSize;            /* Size of APDU              */
   Boolean            auth;                /* Does this msg need auth?  */
} TransmitRecord;

typedef struct
{
   RRStatus             status;         /* used? Who is using?     */
   SourceAddress        srcAddr;        /* Who sent this?          */
   TransNum             transNum;
   RequestId            reqId;          /* For matching response   */
   MsTimer              recvTimer;      /* receive timer           */
   TransactionState     transState;     /* What state is it in     */
   Boolean              priority;
   Boolean              altPath;        /* Was alt path used?      */
   Boolean              auth;           /* TRUE if auth succeeded  */
   Boolean              needAuth;       /* Need authentication     */
   ServiceType          serviceType;    /* What type of service    */
   Byte                 rand[8];        /* For authentication      */
   APDU                *response;       /* Store the response      */
   uint16               rspSize;
   APDU                *apdu;           /* Store the APDU received */
   uint16               apduSize;
} ReceiveRecord;

/*****************************************************************
   TSASendParam is used to provide the necessary information
   needed to process the transaction.
   MsgOutAddr has all the necessary information for forming the
   destination address. It also has the domainIndex that can be
   used to determine the domainId, source subnet/node.
   However to facilitate sending of messages even when a node
   is not in any domain (such as sending ManualServiceRequest) this structure
   provides a way to specify the domainIndex with different
   possible values.
   domainIndex = 0 or 1  --> use the corresponding domain table
   domainIndex = 2 ---> use the flexDomain given
   domainIndex = 3 ---> use the domain table as given in destAddr.
   For response messages, several fields such as the destAddr or
   domainIndex is not needed.They are determined from the
   corresponding request message in the receive records.
*****************************************************************/
typedef struct
{
   MsgOutAddr           destAddr;  /* Whom to send? Need Timers too */
   Byte  domainIndex;
   uint8 flexDomainLen;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   Byte  flexDomainId[DOMAIN_ID_LEN];
   ServiceType          service;   /* What type of service? */
   Boolean              auth;      /* Need Authentication?  */
   uint16               apduSize;  /* Size of APDU to be sent */
   MsgTag               tag;       /* For service indication  */
   Boolean              nullResponse; /* TRUE => no resp goes out */
   RequestId            reqId;     /* Set if service=RESPONSE */

   Boolean              altPathOverride;
   Boolean              altPath; /* Used only if altPathOverride is true */
} TSASendParam;

/*****************************************************************
   TSAReceiveParam is used to receive the necessary
   information to process incoming PDU (TPDU or SPDU or AuthPDU).
   Receive Timer is set based on info in srcAddr.
   srcAddr has the source subnet/node, domainIndex used, group if
   any etc.
*****************************************************************/
typedef struct
{
   SourceAddress         srcAddr;    /* Who send it? */
   Boolean               priority;   /* Was it a priority msg?  */
   uint16                pduSize;    /* Size of incoming PDU */
   PDUType               pduType;    /* What type of PDU? */
   Boolean               altPath;    /* Was it sent in alt path? */
} TSAReceiveParam;

typedef struct
{
   uint8   deltaBL;  /* What is the backlog generated by this msg? */
   Boolean altPath;  /* Should altPath be used? */
   uint16  pduSize;  /* Size of NPDU */
} LKSendParam;

/* SNVT data structures */

/* See comment in app.c in APPReset and AddNV for a description */
/* of how the SNVT area is layed out and managed.               */

typedef struct
{
   Bits  extRec            : 1;
   Bits  nvSync            : 1;
   Bits  nvPolled          : 1;
   Bits  nvOffline         : 1;
   Bits  nvServiceConfig   : 1;
   Bits  nvPriorityConfig  : 1;
   Bits  nvAuthConfig      : 1;
   Bits  nvConfigClass     : 1;
   uint8 snvtTypeIndex;
} SNVTdescStruct;

typedef struct
{
   Bits          bindingII    : 1;
   Bits          queryStats   : 1;
   Bits          aliasCount   : 6;
   uint16        hostAlias;
} AliasField;

typedef struct
{
   uint16          length;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   uint8           numNetvars;
   uint8           version;
   uint8           msbNumNetvars;
   uint8           mtagCount;
   char            sb[SNVT_SIZE];
   SNVTdescStruct  *descPtr; /* Point to next SNVTdesc entry in sb */
                             /* Actually, it points Node Self-Doc
                                String. Just before that is the
                                self-id info for the last netwar */
   AliasField      *aliasPtr; /* Point to alias structure in sb */
} SNVTstruct;

typedef struct
{
   Bits  mre  : 1;    /* Maximum rate    */
   Bits  re   : 1;    /* Rate            */
   Bits  nm   : 1;    /* Name            */
   Bits  sd   : 1;    /* Self Doc String */
   Bits  nc   : 1;    /* Array Count     */
   Bits       : 3;    /* Unused          */
} SNVTextension;

/* Partial list of SNVT type index values */
/* Unused in Ref Impl. Given for illustration */
typedef enum
{
    SNVT_STR_ASC  = 36,
    SNVT_LEV_CONT = 21,
    SNVT_LEV_DISC = 22,
    SNVT_COUNT_F  = 51,
} SNVTType;

typedef struct
{
   int16 nvIndex; /* Base index of array network variables */
   int16 dim;     /* The dimension of the array */
} NVArrayTbl;

/* Type Definition for Protocol Stack Data */
typedef struct
{
   /* Number of bytes used so far */
   uint16  mallocUsedSize;

   /* Array of storage space for dynamic allocation of buffers etc */
   Byte mallocStorage[MALLOC_SIZE];

   /* Variables to store error messges generated using ErrorMsg fn */
   char               errorMsg[ERROR_MSG_SIZE];
   uint16             errorMsgNumber;
   uint16             errorMsgIndex;

   /* Variable to keep track of current Time. Points to time */
   uint32             softwareTime;
   uint32            *currentTime; /* Points to softwareTime or
                                      actual hardware timer loc */

   /* Variables for Transaction Control Sublayer */
   TransCtrlRecord priTransCtrlRec;

   TransCtrlRecord nonpriTransCtrlRec;

   TransNum       priTransID;
   TransNum       nonpriTransID;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
TIDTableEntry priTbl[TID_TABLE_SIZE];
TIDTableEntry nonpriTbl[TID_TABLE_SIZE];
/* # entries currently used */
uint16        priTblSize;
uint16        nonpriTblSize;

/* Timer to delay Transport/Session layers after an external or
   power-up reset. */
MsTimer tsDelayTimer;

/* Transmit and Receive Records */
TransmitRecord xmitRec;
TransmitRecord priXmitRec;

ReceiveRecord  *recvRec;  /* Pool of records */
uint16 recvRecCnt;        /* How many Records allocated? */

RequestId reqId; /* Running count for request numbers */
Byte      prevChallenge[8]; /* Used in generation of new challenge. */

/* Various Queues */
/*************************************************************
  There are 3 queues associated with Each Layer except Physical.
  1. Input  Queue 2. Output Queue 3. Output Priority Queue

  Each item in a queue has appropriate Param Structure
  followed by appropriate PDU.

  Buffer Sizes and Count Values are available in readOnlyData
  *************************************************************/
/* Input Queue For App Layer */
Queue      appInQ;
uint16     appInBufSize;
uint16     appInQCnt;

/* Output Queue For APP Layer */
Queue      appOutQ;
uint16     appOutBufSize;
uint16     appOutQCnt;

/* Output Priority Queue for APP Layer */
Queue      appOutPriQ;
uint16     appOutPriBufSize;
uint16     appOutPriQCnt;

/* Input Queue For Transport, Session, Authentication Layers */
Queue         tsaInQ;
uint16        tsaInBufSize;
uint16        tsaInQCnt;

/* Output Queue For Transport, Session, Authentication Layers */
Queue         tsaOutQ;
uint16        tsaOutBufSize;
uint16        tsaOutQCnt;

/* Output Priority Queue For Transport, Session, Auth Layers */
Queue         tsaOutPriQ;
uint16        tsaOutPriBufSize;
uint16        tsaOutPriQCnt;

/* Output Queue for Responses. Just one queue is sufficient.
   Priority or Non-priority is determined based on request */
Queue         tsaRespQ;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
uint16        tsaRespBufSize;
uint16        tsaRespQCnt;

/* Input Queue For network Layer */
Queue     nwInQ;
uint16    nwInBufSize;
uint16    nwInQCnt;

/* Output Queue For network Layer */
Queue     nwOutQ;
uint16    nwOutBufSize;
uint16    nwOutQCnt;

/* Output Priority Queue For network Layer */
/* Buffer size is same as the buffer size for Output Queue */
Queue     nwOutPriQ;
uint16    nwOutPriBufSize;
uint16    nwOutPriQCnt;

/* Input Queue For Link Layer */
Byte         *lkInQ;
uint16        lkInBufSize; /* Size of buffer in lkInPDUQ */
uint16        lkInQCnt;    /* # of Buffers allocated. */
Byte         *lkInQHeadPtr;
Byte         *lkInQTailPtr;

/* Output Queue For Link Layer */
Queue         lkOutQ;
uint16        lkOutBufSize;
uint16        lkOutQCnt;

/* Output Priority Queue For Link Layer */
/* Buffer size is same as the buffer size for Output Queue */
Queue         lkOutPriQ;
uint16        lkOutPriBufSize;
uint16        lkOutPriQCnt;

/* Output Queue For Physical Layer */
Byte      *phyOutQ; /* Not a regular Queue unlike others */
uint16     phyOutBufSize;
uint16     phyOutQCnt;
Byte      *phyOutQHeadPtr;
Byte      *phyOutQTailPtr;

/* Output Priority Queue For Physical Layer */
/* Buffer size is same as the buffer size for Output Queue */
Byte      *phyOutPriQ; /* Not a regular Queue unlike others */
uint16     phyOutPriBufSize;
uint16     phyOutPriQCnt;
Byte      *phyOutPriQHeadPtr;
Byte      *phyOutPriQTailPtr;

/* API Flags */
Boolean msgReceive;    /* TRUE when data in gp->msgIn */
Boolean respReceive;   /* TRUE when data in gp->respIn */

Boolean callMsgFree;   /* Flag to help implicit call to msg_free */
                       /*after DoApp  */
Boolean callRespFree;  /* Flag to help implicit call to resp_free */
                       /* after DoApp */

/* API Variables */
RespIn respIn;
RespOut respOut;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
MsgIn  msgIn;
MsgOut  msgOut;

/* Flag to determine if selected for Net Mgmt queries */
Boolean selectQueryFlag;

/* Unbound selector counter for automatic assignement in AddNV. */
uint16 unboundSelector;

/* Table to keep track of array network variables and dim */
NVArrayTbl nvArrayTbl[MAX_NV_ARRAYS];
uint16     nvArrayTblSize;

/* Queue of nvIndex for network output variables.
   This queue stores the indices of network variables
   (primary or alias) that are scheduled for sending out
   NVUpdate messages. Each index takes 2 bytes. The index
   is optionally followed by the value of the variable to
   be used with the NVUpdate message. This is done for sync
   network output variables so that we use the value when the
   request was made rather than the current value.

   Update to one network output variable may involve zero or one (primary)
   and zero or more alias indices. Thus we have a collection of indices to be
   scheduled. Alias entries can have different service type or priority
   attributes. To make scheduling work properly, we will have only one
   queue for storing these indices. After scheduling all the alias
   indices, if any, for the primary, we will add an extra entry
   that is equal to NV_UPDATE_LAST_TAG_VALUE to indicate end of
   one scheduling of a primary. When these indices are actually processed,
   we will process one at a time. i.e. we don't send NV message unless
   the previous one is completed.

   An NVUpdate for a primary succeeds if all the transactions for
   scheduled for the NVUpdates succeed. We keep track of this in
   the flag nvOutStatus. The flag nvOutSchedule is set to TRUE if
   scheduling can be done, FALSE otherwise. This is reset to TRUE at
   the completion of the primary and all alias schedules. */

Queue         nvOutIndexQ;
uint16        nvOutIndexQCnt;
uint16        nvOutIndexBufSize;
Boolean       nvOutStatus;      /* Used to give NVUpdateCompletes for */
                                /* Propagate. */
Boolean       nvOutCanSchedule; /* TRUE --> can continue to schedule. */
int16         nvOutIndex;       /* current primary index scheduled.   */

/* Queue of nvIndex for network input variables.
   This queue stores the input variables that scheduled
   to be polled. Each item exactly 2 bytes to store the index
   of the variable scheduled. Also, see the comment above for the
   nvOutIndexQ. The comment is valid for this queue for the poll
   messages too.

   Polling one network input variable may involve zero or one (primary)
   and zero or more alias indices. Thus we have a collection of indices to be
   scheduled. The alias can have different service type or priority.

   A poll succeeds if both nvInDataStatus and nvInTranStatus are true. */

Queue         nvInIndexQ;
uint16        nvInIndexQCnt;
Status        nvInDataStatus;  /* true if any valid data from external node
                                  is received or nv is turnaround only.  */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   Status        nvInTranStatus;  /* true if all transactions for the poll */
                                  /* succeeded.                            */
   Boolean       nvInCanSchedule; /* TRUE --> can continue to schedule.    */
   int16         nvInIndex;       /* current primary index scheduled.      */

   int16               nvArrayIndex;
   NvInAddr            nvInAddr;

   /* Flag to indicate scheduler whether reset is needed or not */
   Boolean resetNode; /* TRUE ==> reset needed */
   Boolean resetPinPrevState;  /* For PHYIO's use */

   /* To Check if reset was successful */
   Boolean resetOk;

   /* To Check if the manual service request button was pressed or not */
   Boolean manualServiceRequest;
   Boolean manualServiceRequestPrevState;  /* For PHYIO's use */

   /* To Check if generic io input pin 0 is pressed or not */
   Boolean ioInputPin0;
   Boolean ioInputPin0PrevState; /* For PHYIO's use */

   /* To set generic io output pin 0 */
   Boolean ioOutputPin0;

   /* To set generic io output pin 1 */
   Boolean ioOutputPin1;

   /* To Remember prev state of the buttons. For IOChanges's use */
   Boolean prevPinState[1]; /* Currently only one pin is supported */

   /* Represents the mode for the application program in configured state. */
   Byte appPgmMode;  /* Possible Values: OFF_LINE OFF_LINE NOT_RUNNING */

   /* For Msg Tag assignments */
   uint16 nextBindableMsgTag;
   uint16 nextNonbindableMsgTag;
} ProtocolStackData;

typedef struct
{
   ReadOnlyDataStruct  readOnlyData;
   ConfigDataStruct    configData;
   DomainStruct        domainTable[MAX_DOMAINS];
   AddrTableEntry      addrTable[NUM_ADDR_TBL_ENTRIES];

   /* Checksum for config structure */
   Byte   configCheckSum; /* Exclusive or of successive bytes in
                             config structure */

   /* Define some space above EEPROM to allow EEPROM expansion */
   /* The following variable makes the EEPROM exactly 3k in size */
   /* If actual EEPROM size is < 3k, it should handle addresses
      that are not in range */
   Byte                reserved[3 * 1024 -
                                sizeof(ReadOnlyDataStruct) -
                                sizeof(ConfigDataStruct)   -
                                sizeof(DomainStruct) * MAX_DOMAINS  -
                                sizeof(AddrTableEntry) * NUM_ADDR_TBL_ENTRIES -
                                1];

} EEPROM;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
typedef struct
{
   char       pxyType;    /* Proxy type (Green Book p. 51). -1 => not in use. */
   RequestId  reqId;      /* Request ID of proxy */
} ProxyData;

typedef struct
{
   /* Firmware space. Not used for anything in Ref Imp.
      Usable by users */
   Byte          firmwareSpace[16 * 1024]; /* 16 K size */
   Byte          userSpace[42 * 1024];     /* User space */
   /* RAM starts here */
   StatsStruct   stats;
   SNVTstruct    snvt;
   ProxyData     pxyData;
   ErrorLog      errorLog;
   uint8         resetCause;
   Byte          restOfRAM[2 * 1024 -
                   sizeof(StatsStruct) -
                   sizeof(SNVTstruct) -  sizeof(ProxyData) -
                   sizeof(ErrorLog) - sizeof(uint8)];
   /* EEPROM goes here. But kept separate as it can be loaded
      into Actual EEPROM more easily if it is isolated */
   /* Memory Mapped Space */
   Byte          memMapSpace[1024]; /* 1k in size */

   /* Place network varibale information above the 64K nmap so
      that we can support large number of variables. Since these
      are not kept in EEPROM, some other mechanism should be used
      to make sure that these values are preserved after a power up */
   NVStruct            nvConfigTable[NV_TABLE_SIZE];
   AliasStruct         nvAliasTable[NV_ALIAS_TABLE_SIZE];
   NVFixedStruct       nvFixedTable[NV_TABLE_SIZE];
   uint16              nvTableSize; /* Config or Fixed */
} NmMap; /* Memory Map */

/* Turn the alignment off */
#pragma maxalign()

/*---------------------------------------------------------------------
Section: Global Variables
---------------------------------------------------------------------*/
/* Node Data Structures */
extern   ProtocolStackData   protocolStackDataGbl[NUM_STACKS];
extern   ProtocolStackData  *gp; /* Pointer to current Structure */
extern   EEPROM              eeprom[NUM_STACKS];
extern   EEPROM             *eep; /* Pointer to current eeprom str */
extern   NmMap               nm[NUM_STACKS];
extern   NmMap              *nmp;

#if defined(ALTERNATE_STRUCTURES_NEEDED)
extern msg_in_type     msg_in;
extern msg_out_type    msg_out;
extern resp_in_type    resp_in;
extern resp_out_type   resp_out;
extern nv_in_addr_type nv_in_addr;
extern int16           nv_array_index;
#endif

/*---------------------------------------------------------------------
Section: Function Prototypes
---------------------------------------------------------------------*/
DomainStruct *AccessDomain(uint8 indexIn);
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
void    UpdateDomain(DomainStruct *domainInp, uint8 indexIn);
void    UpdateCloneDomain(DomainStruct *domainInp, uint8 indexIn);
AddrTableEntry *AccessAddress(uint16 indexIn);
void    UpdateAddress(AddrTableEntry *addrEntryInp, uint16 indexIn);
uint16  AddrTableIndex(Byte domainIndexIn, uint8 groupIn);
Boolean IsGroupMember(Byte domainIndex, uint8 groupIn,
                      uint8 *groupMemberOut);
uint16  DecodeBufferSize(uint8 bufSizeIn);
uint16  DecodeBufferCnt(uint8 bufCntIn);
uint16  DecodeRptTimer(uint8 rptTimerIn);
uint16  DecodeRcvTimer(uint8 rcvTimerIn);
uint16  DecodeTxTimer(uint8  txTimerIn);
uint16  DecodeNonGroupTimer(uint8 nonGroupTimerIn);
NVStruct *AccessNV(uint16 indexIn);
void    UpdateNV(NVStruct *nvStructInp, uint16 indexIn);
AliasStruct *AccessAlias(uint16 indexIn);
void    UpdateAlias(AliasStruct *aliasStructInp, uint16 indexIn);
uint16  NVTableIndex(char varNameIn[]);
uint16  AliasTableIndex(char varNameIn[]);
void    ErrorMsg(char errMessageIn[]);
void    DebugMsg(char debugMsg[]);
void   *AllocateStorage(uint16 size);
uint32  GetCurrentTime(void);
uint32  GetCurrentMsTime(void);
void    SetMsTimer(MsTimer *timerOut, uint16 initValueIn);
void    UpdateMsTimer(MsTimer *timerInOut);
Boolean MsTimerExpired(MsTimer *timerInOut);
void    NodeReset(Boolean firstReset);
void    InitEEPROM(void);
Boolean IOChanges(uint8 pinNumberIn);
uint8   CheckSum4(void *data, uint16 lengthIn);
uint8   CheckSum8(void *data, uint16 lengthIn);
uint8   ComputeConfigCheckSum(void);
int16   GetPrimaryIndex(int16 nvIndexIn);
NVStruct *GetNVStructPtr(int16 nvIndexIn);
Boolean IsTagBound(uint8 tagin);
Boolean IsNVBound(int16 nvIndexIn);
Boolean AppPgmRuns(void);
Boolean NodeConfigured(void);
Boolean NodeUnConfigured(void);


#endif   /* #ifndef _NODE_H */
/*******************************node.h*******************************/
/************************************************************************
  File:           custom.h  (custom.h)

  Version:        1.7

  References:CNP, part 1.

  Purpose:        Contains constant definitions that can be
                  used to customize the characteristics of the
                  node running the reference implementation.

  Note:           None.

  To Do:          None.

************************************************************************/
#ifndef _CUSTOM_H
#define _CUSTOM_H

/*-----------------------------------------------------------------------------
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Section: Includes
-------------------------------------------------------------------------------*/
#include <cnp_1.h>  /* To get constants and types (e.g. Byte) */

/*-------------------------------------------------------------------------
Section: Constant Definitions
-------------------------------------------------------------------------------*/
/* ReadOnlyData. Reference: Tech Device Data Book. Revision 3. p.9-6 */
#define MODEL_NUM             128
#define MINOR_MODEL_NUM       0
#define READ_WRITE_PROTECT    0
#define RUN_WHEN_UNCONF       0 /* Set to 1 if applicaiton needs to run
                                   even if the node is unconfigured */

/*********************************************************************
   NUM_ADDR_TBL_ENTRIES:
   The address table in reference implementation supports more than
   15 entries. However, network management tools might support only
   upto 15 entries. So, other entries should be managed by the
   application program itself. The maximum supported value is 255.
*********************************************************************/
#define NUM_ADDR_TBL_ENTRIES   15    /* # of entries in addr tbl    */

#define RECEIVE_TRANS_COUNT    50    /* Can be > 16 for Ref. Impl */

#define NV_TABLE_SIZE         100    /* Check management tool for any
                                        restriction on maximum size */

#define NV_ALIAS_TABLE_SIZE    10    /* Check management tool for any
                                        restriction on maximum size */

#define SNVT_SIZE            1050    /* Maximum allowed storage space
                                        for SNVT structures */

/*********************************************************************
Warning!!! The following constants are encoded. So, don't use the
actual values
*********************************************************************/
#define APP_OUT_BUF_SIZE     15
#define APP_IN_BUF_SIZE      15    /* 11 ==> buf size is 66         */
#define NW_OUT_BUF_SIZE      15
#define NW_IN_BUF_SIZE       15

#define APP_OUT_Q_CNT         3
#define APP_OUT_PRI_Q_CNT     8    /* 3 ==> 2  8 ==> 15             */
#define APP_IN_Q_CNT          3

#define NW_OUT_Q_CNT          5
#define NW_OUT_PRI_Q_CNT      8
#define NW_IN_Q_CNT           3

#define NGTIMER_SPCL_VAL    8.192

#define NON_GROUP_TIMER 8

/* Reference: Tech Device Data Book. Revision 3. p9-27. */
#define NM_AUTH 0

/* Node Self Documentation String */
#define NODE_DOC  "Ref. Impl. Pgm."

/* If the node is a member of group A(say), then typical 709 applications
   set group size to 1 more than actual group size if node is not
   a member of group A. This is done so that the number of
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   acknowledgements to be expected is always (groupsize - 1).
   In reference implementation, there is an option
   to set this to the true group size and let the transport
   and session layers take care of this. Comment the following
   line to do this. For backward compatibility, uncomment it.
*/
#define GROUP_SIZE_COMPATIBILITY

/* Maximum number of array network variables allowed in
   the application program. This constant is used to allocate
   space that keeps track of all arrays and their dimension */
#define MAX_NV_ARRAYS 10

/* Maximum number of network output variables that can
   be scheduled to be sent out at any point in time */
#define MAX_NV_OUT     50

/* To implement synchronous variables, the values of the
   variables are to be stored along with index in the queue.
   Define the maximum size (in bytes) of a network variable
   in the application program. This is used for storage allocation. */
#define MAX_NV_LENGTH 50

/* Maximum number of network input variables that can
   be scheduled to be polled at any point in time */
#define MAX_NV_IN    50

/* Maximum number of bytes in data array for msg_in, msg_out, resp_in etc.
   This value is indepedent of application buffer sizes mentioned
   earlier. Clearly it does not make sense for this value to be
   larger than application buffer size (out or in). */
#define MAX_DATA_SIZE 100

/* The following constant is used to delay transport and session
   layers on an external or power-up reset for sending messages
   to make sure that messages sent after a reset are not discarded
   as duplicates. The dealy should be small enough to ensure that
   there will be no message in receive transaction records of
   target nodes. Normal default is 2 seconds. The amount is given
   in milliseconds.
   The timer duration would be set based on the maximum expected receive
   timer value in all target nodes. */
#define TS_RESET_DELAY_TIME 2000

/*****************************************************************************
   Protocol Stack Implementation uses an array to allocate storage
   space dynamically. The size of the array used for this allocation
   is determined by this constant. If it is too low, it may be
   impossible to allocate necessary buffers or data structures.
   If it is too high, some memory is unused.
   Set to some high value, run program, stop, and check
   gp->mallocUsedSize to determine the current usage.
   This array space is allocated during Reset of all layers.
   Tracing through the Reset code of all layers will indicate
   the approximate size of this array necessary.
   If AllocateStorage function in node.c is rewritten to use malloc, then
   this constant will be of no use.
*****************************************************************************/
#define MALLOC_SIZE     52000

/*****************************************************************************
Section: Type Definitions
*****************************************************************************/
typedef Byte DomainId[DOMAIN_ID_LEN];
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
typedef Byte AuthKey[AUTH_KEY_LEN];

/* Other values that can be initialized are in the following structure.
   The actual values are given in custom.c file. */
typedef struct
{
   /* ReadOnlyData Members */
   Byte uniqueNodeId[UNIQUE_NODE_ID_LEN];
   Byte twoDomains;
   char progId[ID_STR_LEN];

   /* ConfigData Members */
   char location[LOCATION_LEN];

   /* Domain Table Members */
   DomainId domainId[MAX_DOMAINS];
   Byte len[MAX_DOMAINS];
   Byte subnet[MAX_DOMAINS]; /* One for each domain */
   Byte node[MAX_DOMAINS];
   AuthKey key[MAX_DOMAINS];  /* 6 byte authentication key */

   /* Address Table Info. Enter all 5 byte values */
   /* Note: LonBuilder will overwrite the first 15 entries during
      Load/Start. This was used for only for testing purpose */
   /* Alias tables are not handled by some management tools
      and hence this initialization might be useful. */
   Byte addrTbl[NUM_ADDR_TBL_ENTRIES][5];
   Byte aliasTbl[NV_ALIAS_TABLE_SIZE][6];

} CustomData;

extern CustomData  customDataGbl[NUM_STACKS];
extern CustomData *cp;


#endif   /* #ifndef _CUSTOM_H */
/*******************************End of custom.h*****************************/

/*****************************************************************************
          File:        network.h

       Version:        1.7

     Reference:        CNP, part 1, clause 8.

       Purpose:        This is the interface file for network layer.

          Note:        Ref. Impl. does not support Router Nodes.

         To Do:        None.
*****************************************************************************/
#ifndef _NETWRK_H
#define _NETWRK_H

/*---------------------------------------------------------------------------
Section: Includes
---------------------------------------------------------------------------*/
/* None */

/*---------------------------------------------------------------------------
Section: Constant Definitions
---------------------------------------------------------------------------*/
/* None */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/*-------------------------------------------------------------------------
Section: Type Definitions
-------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------------
Section: Globals
-------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------------
Section: Function Prototypes
-------------------------------------------------------------------------*/
void   NWReset(void);
void   NWSend(void);
void   NWReceive(void);

#endif
/*-----------------------------End of network.h-----------------------------*/
/*********************************************************************
          File:        tcs.h

       Version:        1.7

       Purpose:        Interface for Transaction Control Sublayer.
                       Outgoing Sequencing
                       Incoming Sequencing and Duplicate Detection.

          Note:        For assigning TIds, a table is used. We
                       remember the last TID for each unique dest
                       addr. When a new TId is requested for a
                       destination, this table is searched for that
                       destination. If found, we make sure that we
                       don't assign the same TId used for that
                       destination. If the destination is not
                       found, we make a new entry in the table.

                       We have an entry in the table for each
                       subnet/node, group, broadcast, subnet
                       broadcast, unique node id. When a table entry
                       is assigned, we remember the time stamp too.
                       If the table does not have space for a new
                       dest addr, we get rid of one that has
                       remained more than 24 sec. If no such entry,
                       then we fail to allocate the new trans ID.
                       The table size is configurable.

         To Do:        None

*********************************************************************/
#ifndef _TCS_H
#define _TCS_H

/*-------------------------------------------------------------------------
Section: Includes
-------------------------------------------------------------------------*/
#include <cnp_1.h>
#include <node.h>

/*-----------------------------------------------------------------
Section: Constant Definitions
-----------------------------------------------------------------*/
/* None */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/*-------------------------------------------------------------------
Section: Type Definitions
-------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------
Section: Globals
-------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------
Section: Function Prototypes
-------------------------------------------------------------------*/
void TCSReset(void);

void   TransDone(Boolean  priorityIn);

/* Return Values:      SUCCESS or FAILURE  */
Status NewTrans(Boolean   priorityIn, DestinationAddress addrIn,
               TransNum *transNumOut);

/* Return Values: TRAN_CURRENT or TRAN_NOT_CURRENT */
TransStatus ValidateTrans(Boolean  priorityIn, TransNum transNumIn);

#endif
/*------------------------End of tcs.h------------------------*/
/*********************************************************************
          File:        tsa.h (Transport Session Authentication)

       Version:        1.7

     Reference:        CNP, part 1, clauses 10, 11.

       Purpose:        Interface file for tsa.c

          Note:        None

         To Do:        None
*********************************************************************/
#ifndef _TSA_H
#define _TSA_H

/*-------------------------------------------------------------------------------
Section: Includes
-------------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------
Section: Constant Definitions
-------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------
Section: Type Definitions
-------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------
Section: Globals
-------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------
Section: Function Prototypes
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
-------------------------------------------------------------------*/
void TSAReset(void);

void TPSend(void);
void TPReceive(void);

void SNSend(void);
void SNReceive(void);

void AuthSend(void);
void AuthReceive(void);

#endif
/*------------------------End of tsa.h-------------------------*/
/*************************************************************************
          File:        netmgmt.h

       Version:        1.7

     Reference:        CNP, part 1, clause 12.

       Purpose:        App Layer / Network Management

          Note:        For more information, see netmgmt.c.

         To Do:        None

*************************************************************************/
#ifndef _NETMGMT_H
#define _NETMGMT_H

/*-------------------------------------------------------------------------
Section: Includes
-------------------------------------------------------------------------*/
#include <cnp_1.h>
#include <queue.h>
#include <tsa.h>
#include <network.h>
#include <api.h>

/*-------------------------------------------------------------------------
Section: Constant Definitions
-------------------------------------------------------------------------*/
typedef enum
{
   ND_MESSAGE,
   NM_MESSAGE
} NtwkMgmtMsgType;

/* Message codes */

#define NM_QUERY_ID           0x1
#define NM_RESPOND_TO_QUERY   0x2
#define NM_UPDATE_DOMAIN      0x3
#define NM_LEAVE_DOMAIN       0x4
#define NM_UPDATE_KEY         0x5
#define NM_UPDATE_ADDR        0x6
#define NM_QUERY_ADDR         0x7
#define NM_QUERY_NV_CNFG      0x8
#define NM_UPDATE_GROUP_ADDR  0x9
#define NM_QUERY_DOMAIN       0xA
#define NM_UPDATE_NV_CNFG     0xB
#define NM_SET_NODE_MODE      0xC
#define NM_READ_MEMORY        0xD
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
#define NM_WRITE_MEMORY       0xE
#define NM_CHECKSUM_RECALC    0xF
#define NM_INSTALL            0x10
#define NM_WINK               0x10 /* Alias for NM_INSTALL */
#define NM_MEMORY_REFRESH     0x11
#define NM_QUERY_SNVT         0x12
#define NM_NV_FETCH           0x13
#define NM_MANUAL_SERVICE_REQUEST 0x1F


/* Define offsets and masks for constructing request and response codes */

#define NM_opcode_base      0x60
#define NM_opcode_mask      0x1F
#define NM_resp_mask        0xE0
#define NM_resp_success     0x20
#define NM_resp_failure     0x00

#define ND_opcode_base      0x50
#define ND_opcode_mask      0x0F
#define ND_resp_mask        0xF0
#define ND_resp_success     0x30
#define ND_resp_failure     0x10

/* Network Diagnostic Message Codes */
#define ND_QUERY_STATUS     0x01
#define ND_PROXY_COMMAND    0x02
#define ND_CLEAR_STATUS     0x03
#define ND_QUERY_XCVR       0x04


/*-----------------------------------------------------------------------------
Section: Type Definitions
-----------------------------------------------------------------------------*/
#pragma maxalign(1)

typedef enum {
      ABSOLUTE_MEM_ADDR  = 0,
      READ_ONLY_RELATIVE = 1,
      CONFIG_RELATIVE    = 2,
      STAT_RELATIVE      = 3,
} ModeType;

typedef enum
{
   UNCONFIGURED   = 0,
   SELECTED       = 1,
   SELECTED_UNCFG = 2,   /* selected *AND* unconfigured */
} Selector;

typedef enum
{
   NO_ACTION            = 0,
   BOTH_CS_RECALC       = 1,
   CNFG_CS_RECALC       = 4,
   ACTION_RESET         = 8,
} Form;

typedef struct {
   Selector selector;
   ModeType mode;
   uint16   offset;
   Byte     count;
   Byte     data[MAX_DATA_SIZE];
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
} NMQueryIdRequest;

typedef struct {
   ModeType mode;
   uint16   offset;
   Byte     count;
   Form form;
   Byte     data[MAX_DATA_SIZE];
} NMWriteMemoryRequest;

#pragma maxalign()

/*------------------------------------------------------------------------------
Section: Globals
------------------------------------------------------------------------------*/
/* None */

/*------------------------------------------------------------------------------
Section: Function Prototypes
------------------------------------------------------------------------------*/
void HandleNM(APPReceiveParam *appReceiveParamPtr, APDU *apduPtr);
void HandleND(APPReceiveParam *appReceiveParamPtr, APDU *apduPtr);
void HandleProxyResponse(APPReceiveParam *appReceiveParamPtr, APDU *apduPtr);

#endif
/*-----------------------------End of netmgmt.h-------------------------------*/
/*******************************************************************************

          File:        app.h

       Version:        1.7

       Purpose:        App Layer
                       Data Structures for App Layer.
                       (See app.c for more information.)

          Note:        None

         To Do:        None

*******************************************************************************/
#ifndef _APP_H
#define _APP_H

/*------------------------------------------------------------------------------
Section: Includes
------------------------------------------------------------------------------*/
#include <cnp_1.h>
#include <queue.h>
#include <tsa.h>
#include <network.h>
#include <api.h>

/*------------------------------------------------------------------------------
Section: Constant Definitions
------------------------------------------------------------------------------*/

/*******************************************************************************
Define tags for use by the application layer for the messages it generates.
There are several types of messages the application layer generates. Proxy,
ManualServiceRequest, network variable updates, and network variable polls. The
tag
for these messages will be set up in such a way that we can tell what type of
message got completed when the application layer receives completion indication
```

-`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
from transport/session layer. Since tags are also used by the application
program and they are all >= 0, we will use negative tags for the application
layer. For tags for the application layer are used as follows:

0xFFFF   ==> tag for proxy command.
0xFFFE   ==> tag for manual service request.
For all NV tags,
bit15 is 1 (negative tag)
bit14 is 1 => nv update 0 => nv poll
bit13 is 1 => last tag.
bit12-bit0 is the actual primary index of network variable.

The tag for which bit13 is set is a special tag value that is recognized
by transport layer. In this case, the transport layer sends an indication
right away with that tag. No message is sent by the transport layer.

Network variable updates and polls are scheduled sequentially. When the
completion event for the last tag is received, completion event is
generated.
*************************************************************************/
#define PROXY_TAG_VALUE                    ((MsgTag) 0xFFFF)
#define MANUAL_SERVICE_REQ_TAG_VALUE       ((MsgTag) 0xFFFE)
#define NV_UPDATE_LAST_TAG_VALUE           ((MsgTag) 0xE000)
#define NV_POLL_LAST_TAG_VALUE             ((MsgTag) 0xA000)

#define PROXY_TAG(tag)                     (tag == PROXY_TAG_VALUE)
#define MANUAL_SERVICE_REQUEST_TAG(tag)    (tag == MANUAL_SERVICE_REQ_TAG_VALUE)

#define NV_UPDATE_TAG(tag)                 ((tag & 0xC000) == 0xC000)
#define NV_POLL_TAG(tag)                   ((tag & 0xC000) == 0x8000)
#define NV_INDEX_OF_TAG(tag)               (tag & 0x1FFF)
#define NV_LAST_TAG(tag)                   ((tag & 0xA000) == 0xA000)

#define GET_NV_UPDATE_TAG(index)           (0xC000 | index)
#define GET_NV_POLL_TAG(index)             (0x8000 | index)

/* Explicit application message codes */
#define APPL_MSG_OFFLINE       0x3F
#define FOREIGN_FRAME_OFFLINE  0x4F
/*-------------------------------------------------------------------------
Section: Type Definitions
-------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------------
Section: Globals
-------------------------------------------------------------------------*/
/* None */

/*-------------------------------------------------------------------------
Section: Function Prototypes
-------------------------------------------------------------------------*/
void APPInit(void);
void APPReset(void);
void APPSend(void);
void APPReceive(void);

#endif
/*---------------------------------End of app.h---------------------------------*/
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.14 Application Protocol State Variables and Address Recognition Structures

```
#define DOMAIN_ID_LEN    6
#define AUTH_KEY_LEN    6
#define Unique_Node_ID_LEN  6

typedef enum addr_type { UNBOUND, SUBNET_NODE, UNIQUE_NODE_ID, BROADCAST }
addr_type;

typedef struct domain_struct{
   unsigned  id[DOMAIN_ID_LEN];
   unsigned  subnet;
   unsigned         :1;
   unsigned  node   :7;
   unsigned  len;
   unsigned  key[AUTH_KEY_LEN];
} domain_struct;

typedef union address_struct{
   group_struct     gp;
   snode_struct     sn;
   bcast_struct     bc;
   turnaround_struct   ta;
} address_struct;

typedef struct group_struct {
   unsigned  type      : 1;   /* 1 => group                     */
   unsigned  size      : 7;   /* group size (0 => huge group)   */
   unsigned  domain    : 1;   /* domain index                   */
   unsigned  member    : 7;   /* member num                     */
   unsigned  rpt_timer : 4;   /* unackd_rpt timer               */
   unsigned  retry     : 4;   /* retry count                    */
   unsigned  rcv_timer : 4;   /* receive timer index            */
   unsigned  tx_timer  : 4;   /* transmit timer index           */
   unsigned  group;           /* group ID                       */
} group_struct;

typedef struct snode_struct {
   addr_type    type;         /* SUBNET_NODE             */
   unsigned  domain    : 1;   /* domain index            */
   unsigned  node      : 7;   /* node number             */
   unsigned  rpt_timer : 4;   /* unackd_rpt timer        */
   unsigned  retry     : 4;   /* retry count             */
   unsigned            : 4;
   unsigned  tx_timer  : 4;   /* transmit timer index    */
   unsigned  subnet;          /* subnet ID               */
} snode_struct;

typedef struct bcast_struct {
   addr_type    type;         /* BROADCAST                    */
   unsigned  domain    : 1;   /* domain index                 */
   unsigned            : 1;
   unsigned  backlog   : 6;   /* backlog override value       */
   unsigned  rpt_timer : 4;   /* unackd_rpt timer             */
   unsigned  retry     : 4;   /* retry count                  */
   unsigned            : 4;
   unsigned  tx_timer  : 4;   /* transmit timer index         */
   unsigned  subnet;          /* subnet ID (0 = domain bcast  */
} bcast_struct;


typedef struct unid_struct {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   addr_type type;
   unsigned  domain    : 1;
   unsigned            : 7;
   unsigned  rpt_timer : 4;
   unsigned  retry     : 4;
   unsigned            : 4;
   unsigned  tx_timer  : 4;
   unsigned  subnet;
   unsigned  nid [UNIQUE_NODE_ID_LEN];
} unid_struct;


typedef   union msg_out_addr {
   addr_type    no_address;  /* UNBOUND 0 if no address    */
   group_struct group;       /* Defined above              */
   snode_struct snode;       /* Defined above              */
   unid_struct  unid;        /* Defined above              */
   bcast_struct bcast;       /* Defined above              */
} msg_out_addr;

/* Typedef for 'msg_in_addr', which is the type of the field 'msg_in.addr'   */

typedef struct msg_in_addr {
   unsigned  domain          : 1;
   unsigned  flexible_domain  : 1;
   unsigned  format          : 6;   /* NOT the 'addr_type' enum.  */
                                    /* INSTEAD: 0 => Bcast, 1 =>  */
                                    /* Group, 2 = > Subnet/Node,  */
                                    /* 3 => Unique_Node_ID        */
   struct    {
      unsigned  subnet;
      unsigned        : 1;
      unsigned  node   : 7;
   } src_addr;
   union {
      unsigned bcast_subnet;
      unsigned group;
      struct    {
         unsigned  subnet;
         unsigned            : 1;
         unsigned  node      : 7;
      } snode;
      struct {
         unsigned  subnet;
         unsigned  nid [UNIQUE_NODE_ID_LEN];
      } unid;
   } dest_addr;
} msg_in_addr;

/* Typedef for 'resp_in_addr', the type of the field 'resp_in.addr'      */


typedef struct resp_in_addr {
   unsigned  domain          : 1;
   unsigned  flexible_domain  : 1;
   struct    {
      unsigned  subnet;
      unsigned  is_snode  : 1;   /* 0=>group resp,   */
                                 /* 1=>snode resp    */
      unsigned  node      : 7;
   } src_addr;
   union {
      struct {
         unsigned  subnet;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            unsigned        : 1;
            unsigned  node  : 7;
        } snode;
        struct {
            unsigned  subnet;
            unsigned        : 1;
            unsigned  node  : 7;
            unsigned  group;
            unsigned        : 2;
            unsigned  member : 6;
        } group;
    } dest_addr;
} resp_in_addr;

struct {
    byte         code;              /* message code                      */
    byte         len;               /* length of message data            */
    byte         data[];            /* message data                      */
    boolean         authenticated;  /* TRUE if message was authenticated */
    service_type service;           /* Service type used to send the msg */
    boolean         duplicate;      /* TRUE if message is a duplicate    */
    unsigned     rcvtx;             /* index to the transaction record   */
    msg_in_addr     addr;           /* destination address (see above)   */
    } msg_in;

struct {
    boolean            priority_on;   /* TRUE if a priority message        */
    msg_tag            tag;           /* to correlate completion codes     */
    byte               code;          /* message code                      */
    byte               data[];        /* message data                      */
    boolean            authenticated; /* TRUE if to be authenticated       */
    service_type       service;       /* service type used to send the msg */
    msg_out_addr       dest_addr;     /* destination address (see above)   */
    } msg_out;

struct {
    byte         code;     /* message code                      */
    byte         len;      /* message length                    */
    byte         data[];   /* message data                      */
    resp_in_addr addr;     /* destination address (see above)   */
    } resp_in;             /* struct for receiving responses    */

struct {
    byte         code;     /* message code                      */
    byte         data[];   /* message data                      */
    } resp_out;            /* structure for sending responses   */
```

A.15 Query-id Data Structures

```
struct query_id_request {
    byte command;          /* value = 1                                  */
    byte selector;         /* 0 = unconfigured nodes                     */
                           /*  1 = respond to query set                  */
                           /*  2 = respond to query set and unconfigured */
    byte address_mode;     /* See "read memory" command                  */
    byte address_hi;
    byte address_lo;
    byte count;
    byte data;             /* "count" bytes of data */
};

struct query_id_response {
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   byte command;    /* value = 1 */
   byte Unique_Node_ID[6];
   byte program_id_string[8];
};
```

A.16 Respond to Query Data Structure

```
struct respond_to_query_cmd {
   byte command;          /* value = 2           */
   byte mode;             /* 1 => ON; 0 => OFF   */
};

struct respond_to_query_resp {
   byte command;    /* value = 2 */
};
```

A.17 Update Domain Data Structures

```
struct update_domain_request {
   byte command;          /* value = 3           */
   byte domain_index;     /* 0 or 1              */
   byte domain_id[6];
   byte subnet;
   byte node;             /* msb must be set     */
   byte did_length;       /* 0, 1, 3, or 6       */
   byte encrypt_key[6];
};

struct update_domain_response {
   byte command;    /* value = 3 */
};
```

A.18 Leave Domain Data Structures

```
struct leave_domain_request {
   byte command;       /* value = 4    */
   byte domain_index;  /* 0 or 1       */
};

struct leave_domain_response {
   byte command;    /* value = 4 */
};
```

A.19 Update Key Data Structures

```
struct update_key_request {
   byte command;       /* value = 5    */
   byte domain_index;  /* 0 or 1       */
   byte encrypt_key[6];
};

struct update_key_response {
   byte command;    /* value = 5 */
};
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.20 Update Address Data Structures

```
struct group_s {
   byte field1;     /* b7: 1                                    */
                    /* b0-b6: group size, 0 for huge group      */
   byte field2;     /* b7: domain ref                           */
                    /* b0-b6: member number, 0 for huge group   */
   byte field3;     /* b4-b7: unackd_rpt timer                  */
                    /* b0-b3: retry count                       */
   byte field4;     /* b4-b7: receive timer index               */
                    /* b0-b3: transmit timer index              */
   byte group;      /* group id.                                */
};

struct snode_s {
   byte type;       /* 1                                                    */
   byte field2;     /* b7: domain ref set by target b0-b6: node number      */
   byte field3;     /* b4-b7: unackd_rpt timer                              */
                    /* b0-b3: retry count                                   */
   byte field4;     /* b0-b3: transmit timer index                          */
   byte subnet;     /* subnet                                               */
   };

struct bdcst_s {
   byte type;       /* 3                              */
   byte field2;     /* b7: domain ref set by target   */
                    /* b0-b5: backlog; 0 == unknown   */
   byte field3;     /* b4-b7: unackd_rpt timer        */
                    /* b0-b3: retry count             */
   byte field4;     /* b0-b3: transmit timer index    */
   byte subnet;     /*  subnet                        */
};

struct ta_s {       /* Turnaround entry               */
   byte type;       /* 0                              */
   byte ta;         /* 1                              */
   byte field3;     /* b4-b7: unackd_rpt timer        */
                    /* b0-b3: retry count             */
   byte field4;     /* b0-b3: transmit timer index    */
};

struct empty_s {        /* Empty entry  */
   byte type            /* 0            */
   byte empty;          /* 0            */
};

struct update_addr_request {
   byte command;        /* value = 6    */
   byte index;          /* 0-14         */
   union {
       struct group_s;
       struct snode_s;
       struct bdcst_s;
       struct ta_s;
       struct empty_s;
   } address;
};

struct update_addr_response {
   byte command;    /* value = 6 */
};
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.21 Query Address Data Structures

```
struct query_addr_request {
   byte command;       /* value = 7  */
   byte index;         /* 0-14       */
};

struct query_addr_response {
   byte command;    /* value = 7 */
   union {
      struct group_s;
      struct snode_s;
      struct bdcst_s;
      struct ta_s;
      struct empty_s;
   } address;
};
```

A.22 Query NV Cnfg Data Structures

```
struct query_nv_cnfg_request {
   byte command;          /* value = 8                                          */
   byte nv_index;         /* index of nv when < 255                             */
   [uint16 nv_index16;]   /* 16-bit index iff nv_index == 255                   */
   byte pri_index;        /* index of primary NV when nv_index refers to an     */
                          /* alias                                              */
   [uint16 pri_index16;]  /* 16 bit index of primary NV when pri_index == 255   */
};

struct query_nv_cnfg_response {
   byte command;    /* value = 8                    */
   byte field1;     /* b7: priority                 */
                    /* b6: direction                */
                    /* b0-b5: net var selector—msb  */
   byte idlo;       /* net var selector—lsb         */
   byte field2;     /* b7: turnaround               */
                    /* b5-b6: service               */
                    /* b4: authenticated            */
                    /* b0-b3: address table index   */
};
```

A.23 Update Group Address Data Structures

```
struct update_group_addr_request {
   byte command;    /* value = 9 */
   struct group_s;
   } address;
};

struct update_group_addr_response {
   byte command;    /* value = 9 */
};
```

A.24 Query Domain Data Structures

```
struct query_domain_request {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    byte command;    /* value = 10              */
    byte index;      /* Domain index, 0 or 1    */
};

struct query_domain_response {
    byte command;    /* value = 10 */
    byte domain_id[6];
    byte subnet;
    byte node;
    byte did_length; /* 0,1,3, or 6 */
    byte encrypt_key[6];
};
```

A.25 Update Network Variable Configuration Data Structures

```
struct update_nv_cnfg_request {
    byte command;           /* value = 11                                  */
    byte nv_index;          /*  0-255;                                     */
    [uint16 nv_index16;]    /*16-bit index iff nv_index == 255            */

    byte field1;            /*  b7: priority                               */
                            /*  b6: direction                              */
                            /*  b0-b5: net var selector—msb                */
    byte idlo;              /* net var selector—lsb                        */
    byte field2;            /*  b7: turnaround                             */
                            /*  b5-b6: service                             */
                            /*  b4: secure                                 */
                            /*  b0-b3: address table index                 */
    byte pri_index;         /* index of primary nv iff nv_index is to      */
    [uint16 nv_index16;]    /* an alias. 16 bit primary index if           */
                            /* pri_nv == 255.                              */
};

struct update_nv_cnfg_response {
    byte command;       /* value = 11 */
};
```

A.26 Set Node Mode Data Structures

```
struct set_node_mode_request {
    byte command;       /* value = 12                                      */
    byte state;         /*  0: offline                                     */
                        /*  1: online                                      */
                        /*  2: reset                                       */
                        /*  3: change state                                */
    byte state_data;    /* Iff state=3; see “Node States” for values       */

struct set_node_mode_response {
    byte command;    /* value = 12   Note, no response is provided*/
                     /*  for offline, online or reset commands    */
};
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.27 Read Memory Data Structures

```
struct read_memory_request {
   byte command;       /* value = 13   */
   byte address_mode;  /* See below    */
   byte address_hi;
   byte address_lo;
   byte count;
};

struct read_memory_response {
   byte command;    /* value = 13 */
   byte data[count];
};

/* where "address_mode" determines the physical address as follows:             */
/*  0: address = address_hi*256 + address_lo                                    */
/*  1: address = non-volatile memory read-only address+address_hi*256+address_lo */
/*  2: address = non-volatile memory configuration address + address_hi*256 +   */
/*     address_lo                                                               */
```

A.28 Write Memory Data Structures

```
struct write_memory_request {
   byte command;       /* value = 14                                    */
   byte address_mode;  /* See comment in read_memory command above      */
   byte address_hi;
   byte address_lo;
   byte count;
   byte form;          /*  0: no reset, no checksum                     */
                       /*  1: recalculate both checksums                */
                       /*  4: recalculate just configuration checksum   */
                       /*  8: reset                                     */
                       /*  9: reset, recalc both checksums              */
                       /*  12:reset, recalc configuration checksum      */
   byte data[count];
};

struct write_memory_response {    /* used only for non-reset writes      */
   byte command;                  /* value = 14                          */
};
```

A.29 Checksum Recalculate Data Structures

```
struct checksum_request {
   byte command;       /* value = 15                          */
   byte which;         /* 1: do application and configuration */
                       /* 4: do configuration only            */
};

struct checksum_response {
   byte command;    /* value = 15 */
};
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.30 Install Command Data Structures

```
typedef struct {
   unsigned     reserved     :3;     /* Must be set to 0 */
   unsigned     nv_direction :1;
   unsigned     nv_auth      :1;
   unsigned     nv_priority  :1;
   unsigned     nv_service   :2;
} nv_dflts;

typedef struct {
   unsigned     reserved               :1;       /* Must be set to 0 */
   unsigned     nv_sync                :1;
   unsigned     nv_polled              :1;
   unsigned     nv_offline             :1;
   unsigned     nv_service_type_config :1;
   unsigned     nv_priority_config     :1;
   unsigned     nv_auth_config         :1;
   unsigned     nv_config_class        :1;
   byte         snvt_type_index;                 /* use enum SNVT_t */
} nv_attr;

typedef struct {
   unsigned     mre          :1;    /* Max Rate Est data is available    */
   unsigned     re           :1;    /* Rate Estimate data is available   */
   unsigned     nm           :1;    /* NV Name is available              */
   unsigned     sd           :1;    /* NV SD text is available           */
   unsigned  nm_supplied     :1;    /* For QUERY_NV_INFO/NV_INFO_DESC,   */
                                    /* name is appended to message       */
   unsigned  reserved1       :3;    /* Must be set to 0                  */
} nv_exten;

typedef struct {
   unsigned     length    :5;       /* # of bytes in NV value               */
   unsigned     origin    :3;       /* How NV was defined                   */
   nv_dflts     dflts;              /* Default NV Configuration settings    */
   nv_attr      attr;               /* Basic NV attributes                  */
   nv_exten     ext;                /* Available NV Extension attributes    */
   nv_array     array;              /* NV array attributes                  */
} nv_desc;

typedef enum{
   APP_WINK            = 0,      /* basic command                          */
   APP_INSTALL         = 1,      /* get information from multiple stacks   */
   APP_NV_DEFINE       = 2,      /* define a new network variable on a node */
   APP_NV_REMOVE       = 3,      /* remove a network variable from the node */
   APP_QUERY_NV_INFO   = 4,      /* get network variable information       */
   APP_QUERY_NODE_INFO = 5,      /* get node self documentation information */
   APP_UPDATE_NV_INFO  = 6       /* modify nv info. Change its type, etc.  */
} AppCommand;
struct install_msg {
   byte command;          /* value = 16                                        */
   byte subcommand;       /* one of AppCommand above                           */
   byte subdata[*];       /* length dependent upon the command/subcommand pair */
};
struct install_response {
   byte command;          /* value = 16 */
   union {
       struct manual_service_request_message;  /* if subcommand == 0 */
   } data;
};

/* Request to define new NV(s) on the node. The app should respond successfully
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
even
   if the NV is already defined.
*/
typedef struct
{
   byte      app_command;    /* APP_NV_DEFINE (AppCommand enum)    */
   uint16    nv_index;       /* New NV index                       */
   uint16    array_len;      /* # of elements in NV array,         */
                              /* 0 if new NV is not an array        */
   byte      nv_length;      /* # of bytes in NV value (1-31)      */
   nv_dflts  dflts;          /* Default NV Configuration settings  */
   nv_attr   attr;           /* NV self-documentation attributes   */
} NM_app_define_nv;

typedef struct
{
    byte                code;     /* NM_app_cmd */
    NM_app_define_nv    data;
} CMD_app_define_nv;

typedef struct
{
    byte    code;             /* NM_app_cmd_fail/NM_app_cmd_succ */
} CMD_app_response;

/* Request to remove a set of NVs. The host should respond successfully
   even if 1 or more of the specified NVs does not exist.
*/
typedef struct
{
   byte          app_command;        /* APP_NV_REMOVE (AppCommand enum)   */
   uint16        nv_index;           /* Index of first NV to remove       */
   uint16        nv_count;           /* # of NVs to remove                */
} NM_app_remove_nv;

typedef struct
{
    byte                code;               /* NM_app_cmd */
    NM_app_remove_nv    data;
} CMD_app_remove_nv;

/*********************************************************************
    Query Network Variable Self-documentation data
 *********************************************************************/

#define NV_INFO_DESC        0
#define NV_INFO_RATE_EST    1
#define NV_INFO_NAME        2
#define NV_INFO_SD_TEXT     3
#define NV_INFO_SNVT_INDEX  4

typedef struct
{
    byte     app_command;        /* APP_QUERY_NV_INFO (AppCommand enum)  */
    byte     nv_info;            /* Requested NV Info (NV_INFO...)       */
    uint16       nv_index;
    union   {
        struct {
             uint16    offset;
             byte   length;
        } sd;                   /* For requesting NV_INFO_SD_TEXT */
    } addl;
} NM_app_query_nv_info;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
typedef struct
{
    byte                    code;               /* NM_app_cmd */
    NM_app_query_nv_info    data;
} CMD_app_query_nv_info;

typedef union
{
    /* Response for requested info NV_INFO_DESC */
    struct {
          nv_desc      desc;
          byte         nv_name[NV_NAME_LEN]; /* Optional field - Included only if
                                                desc.ext.nm_supplied is set */
    } desc;

    /* Response for requested info NV_INFO_RATE_EST */
    struct {
        byte    nv_rate_est;        /* Encoded rate estimate. Only valid if 're'
                                       Is set in NV desc */
        byte    nv_max_rate_est;    /* Encoded max rate estimate. Only valid if
                                       'mre' is set in NV desc */
    } rate;    /* Response for requested info NV_INFO_NAME */
    byte    nv_name[NV_NAME_LEN];   /* NV name. Only valid if 'nm' set in */
                                    /* NV desc */

    /* Response for requested info NV_INFO_SD_TEXT */
    struct {
        byte    length;
        byte    text[*];             /* depends upon length field        */
    } sd;

    /* Response for requested info NV_INFO_SNVT_INDEX */
    byte    snvt_type_index;
} NM_app_query_nv_info_response;

typedef struct
{
    byte                              code;  /* NM_app_cmd_fail/NM_app_cmd_succ */
    NM_app_query_nv_info_response     data;
} CMD_app_query_nv_info_response;

/*********************************************************************
    Update Network Variable Self-documentation data
 *********************************************************************/

typedef struct
{
    byte     app_command;        /* APP_UPDATE_NV_INFO (AppCommand enum)        */
    byte     nv_info;            /* NV Info to be updated (NV_INFO...)          */
    uint16   nv_index;           /* NV index - If the target NV represents an
                                    array,then the update applies to all elements
                                    of the array.                               */
    union {
        /* Data for updated info NV_INFO_NAME */
            nv_name[NV_NAME_LEN];   /* NV name (ascii) - 0 terminated
                                       if < 16 characters */
        /* Data for updated info NV_INFO_SD_TEXT */
        struct {
            byte       length;
            uint16     offset;
            byte       text[*];       /* May not be null (0x00) terminated */
        } sd;

        /* Data for updated info NV_INFO_RATE_EST */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        struct {
       unsigned  reserved     : 4;  /* Must be set to 0                     */
       unsigned  clear_mre    : 1;  /* Clear 'ext' rec 'mre' field          */
       unsigned  clear_re     : 1;  /* Clear 'ext' rec 're' field           */
       unsigned  update_mre   : 1;  /* Update max rate est with nv_max_rate_est */
       unsigned  update_re    : 1;  /* Update rate est with nv_rate_est     */
       byte      nv_rate_est;       /* Encoded rate estimate.               */
       byte      nv_max_rate_est;   /* Encoded max rate estimate.           */
        } rate;

        /* Data for updated info NV_INFO_SNVT_INDEX */
        byte    snvt_type_index;
    } info;
} NM_app_update_nv_info;

typedef struct
{
    byte                    code;       /* NM_app_cmd */
    NM_app_update_nv_info   data;
} CMD_app_update_nv_info;

/*************************************************************************
    Query Node Self-documentation data
 *************************************************************************/

#define NODE_INFO_SD_TEXT     3

typedef struct
{
    byte    app_command;        /* APP_QUERY_NODE_INFO (AppCommand enum)  */
    byte    node_info;          /* Requested Node Info (NODE_INFO...)     */
    union   {
        struct {
            uint16  offset;
            byte       length;
        } sd;                   /* For requesting NODE_INFO_SD_TEXT */
    } addl;
} NM_app_query_node_info;

typedef struct
{
    byte                        code;               /* NM_app_cmd */
    NM_app_query_node_info    data;
} CMD_app_query_node_info;

typedef union
{
    /* Response for requested info NODE_INFO_SD_TEXT */
    struct {
        byte    length;
        byte    text[*];        /* May not be 0 terminated */
    } sd;
} NM_app_query_node_info_response;

typedef struct
{
    byte                            code;  /* NM_app_cmd_fail/NM_app_cmd_succ */
    NM_app_query_node_info_response data;
} CMD_app_query_node_info_response;

Extended Network Management Message Structures

NM_GET_CAPABILITY_INFO
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
typedef struct {
unsigned  command;  // NM_GET_CAPABILITY_INFO
NetWord   offset;   // base is start of alias_field
Unsigned  count;    // number of bytes to read
} NmGetCapabilityInfo;

NM_SET_NV

typedef struct {
    unsigned command;      // NM_SET_NV
    NetWord  index;        // NV index
    unsigned value[1];     // NV value
} NmSetNv;

NM_NODE::NM_INITIALIZE

typedef struct {
   unsigned     command;  // NM_INITIALIZE
   unsigned     resource; // NM_NODE
   nm_node_state state;   // if non-zero, the state to go to afterwards
} NmNodeInitializeReq;

NM_DOMAIN::NM_INITIALIZE

typedef struct {
   unsigned  command;  // NM_INITIALIZE
   unsigned  resource; // NM_DOMAIN
   NetWord   index;    // start index
NetWord   index_end;   // end index.  All 1's means all remaining entries
} NmDomainInitializeReq;

NM_DOMAIN::NM_CREATE and NM_DOMAIN::NM_UPDATE

typedef struct
{
       nme_command     command;  /* NM_UPDATE */
       nme_resource    resource; /* NM_DOMAIN */
       NetWord         index;    /* domain index */
       domain_struct   value;    /* entry data */
} NmDomainUpdateReq;

NM_DOMAIN::NM_ENUMERATE

typedef struct {
   unsigned  command;     // NM_ENUMERATE
   unsigned  resource;    // NM_DOMAIN
   NetWord   index;       // domain table index
} NmDomainEnumerateReq;

typedef struct {
   NetWord   index;       // specified or next non-empty entry
   domain_struct value;   // entry data
} NmDomainEnumerateResp;

NM_DOMAIN::NM_SET_AUTH

typedef struct {
   unsigned  command;         // NM_SET_AUTH
   unsigned  resource;        // NM_DOMAIN
   NetWord   index;           // the domain index to update, or 0xFFFF for all
domains
   unsigned  increment  : 1;  // True to treat the authentication key below as an
                              // increment based on the current key,
                              //false to use it as an absolute value.
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    unsigned             : 7;
    unsigned  authKey[AUTH_KEY_LEN];
} NmDomainSetAuthenticationReq;

Expanded Address Table Structure

typedef enum
{
    GRP_NORMAL          = 0,
    GRP_OUTPUT_ONLY     = 1,
    GRP_INPUT_NO_ACK    = 2,
} group_restriction;

/* extended address table structure */
typedef struct {
    /* group_struct: */
    bits  type          : 1;   /* 1 => group */
    bits  size          : 7;   /* group size (0 => huge group) */
    bits  restriction   : 2;   /* group_restriction */
    bits  member        : 6;   /* member num (if huge group, only use 0) */
    bits  rpt_timer     : 4;   /* unackd_rpt timer */
    bits  retry         : 4;   /* retry count */
    bits  rcv_timer     : 4;   /* receive timer index */
    bits  tx_timer      : 4;   /* transmit timer index */
    unsigned group;            /* group ID */
   NetWord  domain;            /* 16-bit domain index */
} group_struct_ext;

typedef struct {
    /* snode_struct: */
    bits             : 2;   /* must be zero */
    bits  type       : 6;   /* SUBNET_NODE */
    bits             : 1    /* formerly domain index bit */
    bits  node       : 7;   /* node number */
    bits  rpt_timer  : 4;   /* unackd_rpt timer */
    bits  retry      : 4;   /* retry count */

    bits             : 4;
    bits  tx_timer   : 4;   /* transmit timer index */
    unsigned           subnet;       /* subnet ID */
   NetWord  domain;            /* 16-bit domain index */
} snode_struct_ext;

typedef struct {
    /* bcast_struct: */
    bits                : 2;    /* must be zero */
    bits  type          : 6;   /* BROADCAST */
    bits                : 1;   /* formerly domain index bit */
    bits                : 1;
    bits  backlog       : 6;   /* backlog override value */
    bits  rpt_timer     : 4;   /* unackd_rpt timer */
    bits  retry         : 4;   /* retry count */
    bits  max_responses : 4;
    bits  tx_timer      : 4;   /* transmit timer index */
    unsigned     subnet;   /* subnet ID (0 => domain broadcast) */
   NetWord  domain;         /* 16-bit domain index */
} bcast_struct_ext;

typedef turnaround_struct turnaround_struct_ext;

typedef union address_struct_ext
{
    group_struct_ext            gp;
    snode_struct_ext            sn;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    bcast_struct_ext            bc;
    turnaround_struct_ext       ta;
    struct
    {
        address_struct base_struct;
        NetWord    domain;
    } common; /* another way of looking at the structure */
} address_struct_ext;
```

The address structure for Unique Node ID addressing is also modified, though it is not stored in the address table:

```
typedef struct {
   /* UnId_struct: */
    bits  key_override  : 2;
    bits  type          : 6;   /* Unique Node_ID */
    bits                : 1;   /* formerly domain index bit */
    bits                : 7;
    bits  rpt_timer     : 4;
    bits  retry         : 4;
    bits                : 4;
    bits  tx_timer      : 4;
    unsigned    subnet;
    unsigned    unid [Unique Node_ID_LEN];
   NetWord    domain;          /* 16-bit domain index */
} UnId_struct_ext;

NM_ADDRESS::NM_INITIALIZE

typedef struct {
   unsigned     command;  // NM_INITIALIZE
   unsigned     resource; // NM_ADDRESS
   NetWord      index;    // start index.
   NetWord   index  end;          // end index.  All 1's means all
                          // remaining entries
} NmAddressInitializeReq;

NM_ADDRESS::NM_CREATE and NM_ADDRESS::NM_UPDATE

typedef struct {
   unsigned         command;     // NM_UPDATE
   unsigned         resource;    // NM_ADDRESS
   NetWord             index;    // address table index
   address_struct_ext  value;    // entry data
} NmAddressUpdateReq;

NM_ADDRESS::NM_ENUMERATE

typedef struct {
   unsigned  command;     // NM_ENUMERATE
   unsigned  resource;    // NM_ADDRESS
   NetWord      index;    // address table index
} NmAddressEnumerateReq;

typedef struct {
   NetWord   index;          // specified or next non-empty entry
   address_struct_ext value;  // entry data
} NmAddressEnumerateResp;

NV Configuration Table Entry Structure

typedef struct nv_struct_ext
{
                                    /* unsigned 0: */
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   bits     nv_priority       : 1;   /* (same as nv_struct) */
   bits     nv_direction      : 1;   /* (same as nv_struct) */
   bits     nv_selector_hi    : 6;   /* (same as nv_struct) */
                                     /* unsigned 1: */
   unsigned nv_selector_lo;          /* (same as nv_struct) */
                                     /* unsigned 2: */
    bits  nv_turnaround       : 1;   /* (same use as nv_struct) */
    bits  nv_auth             : 1;   /* (same use as nv_struct) */
    bits  nv_write_by_index   : 1;   /* target NV can be written by index, for
                                        outputs only) */
    bits  nv_remote_nm_auth   : 1;   /* target node is nm authenticated */
    bits                      : 4;
    bits  nv_read_by_index    : 1;   /*target NV is be read fetching, not polling
*/
    bits  nv_service          : 2;   /* for outputs only: same use as nv_struct */
    bits                      : 5;
                                    /* Bytes 4 - 7 */
    NetWord nv_addr_index; /*0xffff indicates no corresponding address table entry
*/
    NetWord nv_target  index; /* nv index of the target 0xffff means no target */
} nv_struct_ext;

NM_NV_CONFIG::NM_INITIALIZE

typedef struct {
   unsigned  command;     // NM_INITIALIZE
unsigned  resource;       // NM_NV_CONFIG
NetWord   index;          // start index.
NetWord   index_end;      // end index. All 1's
                          // means all remaining entries
} NmNvConfigInitializeReq;

NM_NV_CONFIG::NM_CREATE and NM_NV_CONFIG::NM_UPDATE

typedef struct {
   unsigned         command;  // NM_UPDATE
   unsigned         resource; // NM_NV_CONFIG
   NetWord          index;    // nv table index
   nv_struct_ext    value;    // entry data
} NmNvConfigUpdateReq;

NM_NV_CONFIG::NM_ENUMERATE

typedef struct {
   unsigned     command;  // NM_ENUMERATE
   unsigned     resource; // NM_NV_CONFIG
   NetWord      index;    // nv index
} NmNvConfigEnumerateReq;

typedef struct {
   NetWord          index; // specified or next allocated nv
   nv_struct_ext    value; // entry data
} NmNvConfigEnumerateResp;

Alias Configuration Table Structures

typedef struct alias_struct_ext
{
   nv_struct_ext alias_nv;
   NetWord   alias_primary;  /* 0xffff means no primary index */
} alias_struct_ext;

NM_ALIAS_CONFIG::NM_INITIALIZE
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
typedef struct {
   unsigned  command;      // NM_INITIALIZE
   unsigned  resource;     // NM_ALIAS_CONFIG
   NetWord   index;        // start index.
   NetWord   index_end;    // end index. All 1's means all remaining entries
} NmAliasConfigInitializeReq;

NM_ALIAS_CONFIG::NM_CREATE and NM_ALIAS_CONFIG::NM_UPDATE

typedef struct {
   unsigned         command;     // NM_UPDATE
   unsigned         resource;    // NM_ALIAS_CONFIG
   NetWord             index;    // nv table index
   alias_struct_ext    value;    // entry data
} NmAliasConfigUpdateReq;

NM_ALIAS_CONFIG::NM_ENUMERATE

typedef struct {
   unsigned  command;      // NM_ENUMERATE
   unsigned  resource;     // NM_ALIAS_CONFIG
   NetWord      index;     // alias index
} NmAliasConfigEnumerateReq;

typedef struct {
   NetWord          index; // specified or next allocated alias
   alias_struct_ext value; // entry data
} NmAliasConfigEnumerateResp;
```

A.31 Memory Refresh Data Structures

```
struct memory_refresh_request {
   byte   command;      /* value = 17                                          */
   uint16    offset;    /* 16 bit offset from start of non-volatile memory     */
   byte   count;        /* Number of bytes to write                            */
   byte   offchip;      /* 1=> refresh off chip memory                         */
};

struct memory_refresh_response {
   byte command;    /* value = 17 */
};
```

A.32 Query SI Data Structures

```
struct query_SI_request {
   byte   command;      /* value = 18 */
   uint16 offset;       /* 16 bit offset into micro's SNVT table    */
   byte   count;        /* number of bytes to return (up to 16)     */
};

struct query_SI_response {
   byte command;        /* value = 18                  */
   byte data[count];    /* return data (count bytes)   */
};
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.33 NV Fetch Data Structures

```
struct nv_fetch_request {
   byte command;       /* value = 19                              */
   byte index;         /* Network Variable selector               */
   [uint16 index16;]   /* 16-bit index, used only if index == 255 */
                       /* This format supported only by host nodes*/
};

struct nv_fetch_response {
   byte command;       /* value = 19                        */
   byte index;         /* Network Variable selector         */
   [uint16 index16;]   /* Iff index == 255; host nodes only */
   byte data[NVLEN];   /* return data (based on NV length)  */
};
```

A.34 Manual Service Request Message Data Structures

```
struct manual_service_request_message {
   byte command; /* value = 31 */
   byte Unique_Node_ID[6];
   byte program_id_string[8];
   };
```

A.35 Product Query Data Structures

```
struct product_query_request {
   byte command;    /* Destination: NM, code: 0x7D (escape) */
   byte data[2];    /* Product query subcode = 0x01         */
                    /* Product query command = 0x01         */
};

struct product_query_short_response {
   byte command;       /* success = 0x3D, failure = 0x1D    */
   byte product_ID;    /* product identifier                */
}

struct product_query_complete_response {
   byte response;   /* success = 0x3D, failure = 0x1D          */
   byte product;    /* product identifier                      */
   byte model[2];   /* model number                            */
   byte version;    /* version number                          */
   byte config;     /* current configuration or mode of device */
   byte Xcvr_ID;    /* transceiver type                        */
};
```

A.36 Router Mode Data Structures

```
struct router_mode_request {
   byte command;    /* Destination: NM, code: 20                        */
   byte mode;       /*  0: resume, 1: init subnet table, 2: mode flood  */
};

struct router_mode_response {
   byte command;    /* Destination: APPL, code: 20 */
};
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.37 Router Table Clear Group or Subnet Table Data Structures

```
struct router_table_clear_request {
   byte command;    /* Destination: NM, code: 21  */
   byte field1;     /* b7: 1 = group, 0 = subnet  */
                    /* b6: domain ref             */
                    /* b0-3: 8x index             */
};

struct router_group_clear_response {
   byte command;    /* Destination: APPL, code: 21 */
};
```

A.38 Router Group or Subnet Download Data Structures

```
struct groupsubnet_table_download {
   byte command;    /* Destination: NM, code: 22         */
   byte field1;     /* b7: 1 = group, 0 = subnet         */
                    /* b6: domain ref                    */
                    /* b0-3: 8x index                    */
   byte table[8];   /* l.s. bit is l.s. group/subnet #   */
};

struct groupsubnet_table_download_response {
   byte command;    /* Destination: APPL, code: 22 */
};
```

A.39 Router Group Forward Data Structures

```
struct group_forward_request {
   byte command;    /* Destination: NM, code: 23                          */
   byte field1;     /* b0: 0: RAM only, 1: RAM + non-volatile memory      */
                    /* b6: domain ref                                     */
   byte group;      /* 0-255                                              */
};

struct group_forward_response {
   byte command;    /* Destination: APPL, code: 23 */
};
```

A.40 Router Subnet Forward Data Structures

```
struct subnet_forward_request {
   byte command;    /* Destination: NM, code: 24                          */
   byte field1;     /* b0: 0: RAM only, 1: RAM + non-volatile memory      */
                    /* b6: domain ref                                     */
   byte subnet;     /* 1-255                                              */
};

struct subnet_forward_response {
   byte command;    /* Destination: APPL, code: 24 */
};
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.41 Router Group No-Forward Data Structures

```
struct group_noforward_request {
   byte command;    /* Destination: NM, code: 25                          */
   byte field1;     /* b0: 0: RAM only, 1: RAM + non-volatile memory      */
                    /* b6: domain ref                                     */
   byte group;      /* 0-255                                              */
};

struct group_noforward_response {
   byte command;    /* Destination: APPL, code: 25 */
};
```

A.42 Router Subnet No-Forward Data Structures

```
struct subnet_noforward_request {
   byte command;    /* Destination: NM, code: 26                          */
   byte field1;     /* b0: 0: RAM only, 1: RAM + non-volatile memory      */
                    /* b6: domain ref                                     */
   byte subnet;     /* 1-255                                         */
};

struct subnet_noforward_response {
   byte command;    /* Destination: APPL, code: 26 */
};
```

A.43 Group / Subnet Table Report Data Structures

```
struct groupsubnet_table_report_request {
   byte command;    /* Destination: NM, code: 27  */
   byte field1;     /* b7: 1 = group, 0 = subnet  */
                    /* b6: domain ref             */
                    /* b0-3: 8x index             */
};

struct groupsubnet_table_report_response {
   byte command;       /* Destination: APPL, code: 27       */
   byte table[8];      /* l.s. bit is l.s. group/subnet #   */
};
```

A.44 Router Status Data Structures

```
struct router_status_request {
   byte command;    /* Destination: NM, code: 28  */
};

struct router_status_response {
   byte command;          /* Destination: APPL, code: 28          */
   byte router_cnfg;      /* type: 1 = learning, 0 = configured   */
                          /* 2 = bridge, 3 = bridge_repeater      */
   byte mode;             /* 0 = normal, 2 = flood                */
};
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.45 Query Status Data Structures

```
struct query_status_request {
   byte command;       /* value = 1 */
};

struct query_status_response {
   byte command;                /* value = 1 */
   struct status_response data;  /* see below */
};

struct status_response {
   uint15    transmission_errors;
   uint16    transaction_timeouts;
   uint16    receive_transaction_full;
   uint16    lost_messages;
   uint16    missed_messages;
   byte   reset_cause;
   byte   node_state;
   byte   version;
   byte   error_log;
   byte   model_number;
};
```

A.46 Proxy Status Data Structures

```
struct proxy_command {
   byte command;       /* value = 2                                */
   byte sub_command;   /* 0=> query id (unconfigured); 1=> status */
                       /* 2=> transceiver status request          */
   union {
      struct group_s;
      struct snode_s;
      struct bdcst_s;
      struct nid_s;
   } address;          /* Address structures as defined in the network
                          management modify address command and below */
};

struct nid_s {
   byte type;       /* 2                              */
   byte field2;     /* b7: domain ref set by target   */
   byte field3;     /* b0-b3: retry count             */
   byte field4;     /* b0-b3: transmit timer index    */
   byte subnet;     /* subnet                         */
   byte nid[6];     /* Unique_Node_ID                 */
};

struct proxy_response {
   byte command;    /* value = 2                                                */
   byte resp_data;  /* data and length are functions of the sub_command         */

};
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.47 Clear Status Data Structures

```
struct clear_status_command {
   byte command;    /* value = 3 */
};

struct clear_status_response {
   byte command;    /* value = 3 */
};
```

A.48 Query Transceiver Status Data Structures

```
struct query_xcvr_status_command {
   byte command;    /* value = 4 */
};

struct query_xcvr_status_response {
   byte command;    /* value = 4*/
   byte data[7];    /* register values */
};
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex B
Additional Data Structures
(Normative)

B.1 General

This appendix contains information on additional protocol processor data structures. The bit-field ordering in all data structures presented here are from high-order to low-order bit within a byte. The byte ordering is high-order to low-order byte within a field.

The software in the protocol processor may be divided into three main sections: system image, application image, and network image.

B.1.1 The System Image

This contains the control network protocol and the runtime environment used in a particular implementation.

B.1.2 The Application Image

This contains the code generated from the user's application program, along with other application-specific parameters. A network management tool may query these parameters. They include:

- Network variable fixed and self-identification data;
- Program ID string;
- Optional self-identification and self-documentation data;
- Number of address table entries;
- Number of domain table entries;
- Number and size of network buffers;
- Number and size of application buffers;
- Number of receive transaction records;
- Transceiver type and bit rate.

The application image data structures described here are:

- Fixed read-only structure, whose size is independent of the application on the node;
- Network variable fixed table, with entries for every network variable defined by this node;
- Optional self-identification and self-documentation information describing this node and its network variables.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.1.3 The Network Image

The network image contains:

- address assignments of the node;
- binding information connecting network variables and message tags between the nodes in the network;
- parameters of the protocol that may be set at installation time; and
- configuration variables of the application program.

A network management tool typically downloads the network image over the network into on-chip EEPROM memory when a node is installed. For simple networks, a node can update its own network image.

This subclause is intended for application programmers who need to understand the internal data structures of the protocol processor that are used for address assignment, binding, and network configuration.

These data structures shall be accessible by network management messages received over the network from a network management tool.

The network image data structures described here are:

- domain table, with an entry for every domain to which this node belongs;
- address table, with an entry for every network address referenced by this node;
- network variable configuration table, with entries for every network variable defined by this node;,
- channel configuration structure, defining the transceiver interface of the node;

B.2 Read-Only Structures

The structures defined in the following subclauses begin with a memory address of 0xF000 for the fixed read-only data structures. This need not map to the actual physical memory in the node. However, commands for read and write memory will provide addresses as though this is the physical layout.

B.2.1 Fixed Read-Only Data Structures

This structure defines the node identification, as well as some of the application image parameters.

```
#define UNIQUE_NODE_ID_LEN 6
#define ID_STR_LEN  8
typedef struct {
   unsigned  unique_node_id[UNIQUE_NODE_ID_LEN];  // offset 0x00
   unsigned  model_num;       // offset 0x06
   unsigned  Checksum  : 4;   // XOR of each of the
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            // 4 bit quantities
            // in unique_node_id
            // and model_num
   unsigned  minor_model_num  : 4;   // offset 0x07
   const  nv_fixed_struct *nv_fixed;    // offset 0x08
   unsigned  read_write_protect  : 1;  // offset 0x0A
   unsigned  run_when_unconfigured   : 1;
   unsigned  nv_count  : 6;
   const  snvt_struct *snvt;      // offset 0x0B
   unsigned  id_string[ID_STR_LEN];    // offset 0x0D
   unsigned  NV_processing_off   : 1;  // offset 0x15
   unsigned  two_domains  : 1;
   unsigned  explicit_addr : 1;
   unsigned  reserved  : 1;
   unsigned  msg_process  : 1;
   unsigned  state  : 3;   // node state
            // 2: unconfigured
            // 3: applicationless
            // 4: configured
            // 6: hard offline
   unsigned  address_count : 4;   // offset 0x16
   unsigned  reserved  : 4;
   unsigned  reserved  : 4;
   unsigned  receive_trans_count : 4;  // offset 0x17
   unsigned  app_buf_out_size : 4;   // offset 0x18
   unsigned  app_buf_in_size  : 4;
   unsigned  net_buf_out_size : 4;   // offset 0x19
   unsigned  net_buf_in_size     : 4;
   unsigned  net_buf_out_priority_count    : 4;   // offset 0x1A
   unsigned  app_buf_out_priority_count    : 4;
   unsigned  app_buf_out_count   : 4;  // offset 0x1B
   unsigned  app_buf_in_count : 4;
   unsigned  net_buf_out_count   : 4;  // offset 0x1C
   unsigned  net_buf_in_count : 4;
   byte   reserved1[6];
   unsigned     : 6;   // offset 0x22
   unsigned  tx_by_address : 1;
   unsigned  idempotent_duplicate   : 1;
} read_only_data_struct;


typedef struct {
   unsigned     : 2;
   unsigned  alias_count   : 6;
   unsigned  msg_tag_count : 4;
   unsigned     : 4;
   byte   reserved2[3];
} read_only_data_struct_2;

const read_only_data_struct read_only_data;
const read_only_data_struct_2 read_only_data_2;
```

The application program may read from but not write to these structures, using the global declaration `read_only_data` and `read_only_data_2`. The structure may be read and mostly written (except for the first eight bytes) over the network using the *Read Memory* and *Write Memory* network management messages with `address_mode=1`. It is written during the process of downloading a new application image into the node.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

B.2.2 Read-only Structure Field Descriptions

```
unsigned  unique_node_id[UNIQUE_NODE_ID_LEN];  // offset 0x00
```
This field is a 6-byte ID assigned by the manufacturer of the protocol processor that is unique to each unit manufactured. It may be read over the network using the *Query ID* network management message. It is also part of the unsolicited *Manual Service Request* network management message.

```
unsigned  model_num;       // offset 0x06
unsigned  minor_model_num  : 4;   // offset 0x07
```
These are two fields that specify the model of the protocol processor.

```
Unsigned  checksum  : 4;   // offset 0x07
```
This field contains the exclusive OR of the other 15 nibbles in the 8 bytes composed of the unique_node_id, model_num, and minor_model_num.

```
const  nv_fixed_struct *nv_fixed; // offset 0x08
```
This field is a pointer to the Network Variable fixed data table (see B.4). If there are no network variables on the node, or this node is a network interface and processing network variable selectors in the application, then this pointer is not used.

```
unsigned  read_write_protect  : 1;   // offset 0x0A
```
This bit specifies that parts of the protocol processor's memory may not be read or written over the network with the *Read Memory* and *Write Memory* network management messages. If this bit is set, only the Read-only Structure (B.1), the SNVT Structures (B.5), the Configuration Structure (B.6), and the application data area may be read, and only the Configuration Structure may be written. The write-protected data includes the `read_write_protect` bit itself, so that once set, the bit may not be reset over the network.

```
Unsigned  run_when_unconfigured  : 1;
```
This bit specifies that the application will run when the node is in the unconfigured state.

```
unsigned  nv_count  : 6;
```
This field specifies the number of network variables declared in the application program running on this node (0-62). Each element of a network variable array is counted separately. If this node is a network interface that is processing network variable selectors in the application, this field is zero.

```
const  si_struct *si;      // offset 0x0B
```
This field is a pointer to the data structure that gives self-identification information for the network variables (see B.5). The self-identification information may be read directly with the read memory command. If the self-identification information is not present, this is a null (0) pointer. A pointer containing all 1's means that the self-identification information is located outside the addressable space of the protocol processor (e.g., on a co-processor). The Query SI command (see 13.7.18) is used in that case to read the self-identification information.

```
unsigned  id_string[ID_STR_LEN]; // offset 0x0D
```
This field contains an 8-byte program identifying information as specified in either of the following directives:

```
#pragma set_id_string "ssssssss"
```
or

```
#pragma set_std_prog_id fm:mm:mm:cc:cc:ss:ss:nn
```

The second format is reserved for nodes that conform to the LONMARK interoperability guidelines (see Bibliography [1]). The bit assignment for this format is as follows:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— • First 4 bits are the format code (0x8 - 0xF). Format 8 is reserved for devices passing the LONMARK conformance review and format 9 should be used by devices with standard program IDs that have not yet passed the conformance review. Formats 0xA - 0xF are reserved for future use.

— • Next 20 bits are the manufacturer code. Assigned to manufacturers when they become members of the LONMARK program.

— • Next 16 bits are the device class. Drawn from a registry of pre-defined class definitions defined by the LONMARK program.

— • Next 16 bits are the device subclass. Drawn from a registry of pre-defined subclass definitions defined by the LONMARK program.

— • Last 8 bits are the manufacturer-assigned model number.

The program ID string may be read over the network using the *Query ID* network management message. It is also part of the unsolicited *Manual service request* network management message.

```
unsigned  two_domains   : 1;
```
This bit specifies that the domain table has two entries (see Annex B.3). If this bit is zero, the domain table has one entry.

```
unsigned  explicit_addr : 1;
```
This bit specifies that the node uses explicit message addressing in its application. If this bit is set, the application buffers contain an 11-byte explicit address field. This is set for any application using explicit addressing or the *nv_in_addr* structure.

```
Unsigned  msg_process   : 1;
```
When this bit field is set to 1, it means that the application can process explicit messages

```
Unsigned  state      : 3;
```
This three bit field represents the node state. The values for the different states are described in 13.4

```
unsigned  address_count : 4;       // offset 0x16
```
This field specifies the number of entries (0-15) in the address table (see B.3).

```
unsigned  receive_trans_count : 4;      // offset 0x17
```
This field specifies the number of receive transactions. The number of receive transactions is one more than the number specified in this field. A value of 0xF means that there are 16 or more receive transactions. .

```
unsigned  app_buf_out_size : 4;       // offset 0x18
unsigned  app_buf_in_size  : 4;
unsigned     net_buf_out_size : 4;      // offset 0x19
unsigned     net_buf_in_size  : 4;
```

These fields specify the sizes of the application and network buffers. The fields are encoded as follows:

Table B1 — Buffer Size Encodings

Field Value	Buffer Size
0	255
2	20
3	21
4	22
5	24
6	26
7	30
8	34
9	42
10	50
11	66
12	82
13	114
14	146
15	210

```
unsigned  net_buf_out_priority_count : 4;   // offset 0x1A
unsigned  app_buf_out_priority_count : 4;
unsigned  app_buf_out_count   : 4;   // offset 0x1B
unsigned  app_buf_in_count : 4;
unsigned  net_buf_out_count   : 4;   // offset 0x1C
unsigned  net_buf_in_count : 4;
```

These fields specify the number of application and network buffers. Note that if one of the priority output buffer counts is zero, then both of them must be zero. The fields are encoded as follows:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B2 — Buffer Count Encodings

Field Value	Number of Buffers
0	0
2	1
3	2
4	3
5	5
6	7
7	11
8	15
9	23
10	31
11	47
12	63
13	95
14	127
15	191

`unsigned tx_by_addr : 1;`
This bit specifies that the node is maintaining a separate outgoing transaction space for each unique destination address in the address table.

`unsigned idempotent_duplicate : 1;`
This bit specifies that the protocol processor sets the idempotent retry bit in the application buffer when a request retry is sent up to the application.

`unsigned alias_count : 6;`
This field specifies the number of entries in the network variable alias table. If this node is an network interface and is processing network variable selectors in the application, this field is zero. Maximum value for this field is 62.

`unsigned msg_tag_count : 4;`
This field specifies the number of bindable message tags used by the node. The range is 0 to 15.

B.3 Domain Table

This table defines the domains to which this node belongs. It is located in non-volatile memory, and is part of the network image written during node installation. During development, the contents of this table are downloaded when the node is loaded.

```
#define AUTH_KEY_LEN    6
#define DOMAIN_ID_LEN   6

typedef struct {
   unsigned  id[ DOMAIN_ID_LEN ];      // offset 0x00
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   unsigned  subnet;       // offset 0x06
   unsigned      : 1;
   unsigned  node   : 7;   // offset 0x07
   unsigned  len;       // offset 0x08
   unsigned  key[ AUTH_KEY_LEN ];      // offset 0x09
} domain_struct;
```

B.3.1 Domain Table Field Descriptions

`unsigned id[DOMAIN_ID_LEN];`
Each domain in a control protocol network has a unique ID of 0, 1, 3 or 6 bytes in length. If the ID is shorter than 6 bytes, it is left-justified in this field.

`unsigned subnet;`
This field specifies the ID of the subnet within this domain to which this node belongs. A subnet ID may be in the range 1 - 255 for each domain. Zero is an invalid subnet ID.

`unsigned node;`
This field specifies the ID of the node within this subnet (1 – 127). Zero is an invalid node ID.

`unsigned len;`
This field specifies the length of the domain ID in bytes (0, 1, 3 or 6). The value 0xFF (255) means that this domain table entry is not in use.

`unsigned key[AUTH_KEY_LEN];`
This field specifies the six-byte authentication key to be used in this domain for authenticated transactions. This key must match the key of all the other nodes on this domain that participate in authenticated transactions with this node. The authentication key may be incremented over the network using the *Update Key* network management message.

B.4 Address Table

This table defines the network addresses to which this node may send implicitly-addressed messages and network variables. It also defines the groups to which this node belongs. It is located in non-volatile memory, and is part of the network image written during node installation.

```
typedef enum { UNBOUND, SUBNET_NODE, Unique_Node_ID, BROADCAST } addr_type;

typedef union {
      group_struct           gp;
      snode_struct           sn;
      bcast_struct           bc;
      turnaround_struct      ta;
} address_struct;

const address_struct *access_address(byte index);
update_address(const address_struct * address_entry, byte index);
byte addr_table_index(message_tag_name);
```

The application program may read and write any entry in this table using the access routines `access_address()` and `update_address()`. The index of the address table entry corresponding to any message tag declared in the program may be determined with the function `addr_table_index()`. The address table consists of up to 15 entries, each 5 bytes in length.

Each entry may be in one of five formats: group address, subnet/node address, broadcast address, turnaround address, or not in use. A group address is used for multicast addressing, when a network variable or message tag is used in a connection having more than two members. A subnet/node

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

address is used for unicast addressing, when an output network variable or message tag is used in a connection with one other node. A turnaround address is used for network variables that are only bound to other network variables in the same node, and not to any network variables on other nodes. Destination addresses in the 48-bit Unique_Node_ID format are never used in the address table, but they may be used as destination addresses in explicitly addressed messages. For completeness, this format is described below.

The entries in this table may be read and written over the network with the *Query Address* and *Update Address Table* network management messages. An entry using group address format may be updated over the network with the *Update Group Address Data* network management message.

The first byte in an address table entry specifies the format of the entry:

0 not in use / turnaround format
1 (subnet, node) format
3 broadcast format
128 - 255 group format

Any bindable message tags declared in the application program are assigned to the first entries in the address table in order of declaration. This is followed by address table entries used for network variables – typically, the binder assigns these entries.

The repeat timer, retry count, receive timer, and transaction timer are common to several of these address formats, and are described in Annex B.4.11.

B.4.1 Declaration of Group Address Format

```
typedef struct {
      unsigned     type                  : 1;          // offset 0x00
      unsigned     size                  : 7;
      unsigned     domain                : 1;          // offset 0x01
      unsigned     member                : 7;
      unsigned     rpt_timer             : 4;          // offset 0x02
      unsigned     retry                 : 4;
      unsigned     rcv_timer             : 4;          // offset 0x03
      unsigned     tx_timer              : 4;
      unsigned     group                 : 8;          // offset 0x04
} group_struct;
```

B.4.2 Group Address Field Descriptions

`unsigned type : 1;`
This bit is 1 for a group address, 0 for any of the other formats.

`unsigned size : 7;`
This field specifies the size of the group (2-64). The size of a group includes the sender of the message. If this field is 0, then the group is of unlimited size, and unacknowledged or repeated service must be used.

`unsigned domain : 1;`
This field specifies the index into the domain table for this address (0 or 1).

`unsigned member : 7;`
This field specifies the member ID of this node within this group (0-63). A group of unlimited size has 0 in this field. The member ID is used in acknowledgments to allow the sender of an acknowledged multicast message to keep track of which nodes have responded.

`unsigned group : 8;`

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

This field specifies the ID of this group within this domain. A group ID may be in the range of 0 - 255. The group ID is typically allocated by a network variable binder.

B.4.3 Declaration of Subnet/Node Address Format

```
typedef struct {
      addr_type    type;                                  // offset 0x00
      unsigned     domain                 : 1;            // offset 0x01
      unsigned     node                   : 7;
      unsigned     rpt_timer              : 4;            // offset 0x02
      unsigned     retry                  : 4;
      unsigned                            : 4;            // offset 0x03
      unsigned     tx_timer               : 4;
      unsigned     subnet                 : 8;            // offset 0x04
} snode_struct;
```

B.4.4 Subnet/Node Address Field Descriptions

`addr_type type;`
This field contains the value `SUBNET_NODE` (1).

`unsigned  domain : 1;`
This field specifies the index into the domain table for this address (0 or 1).

`unsigned  node      : 7;`
This field specifies the node ID (1-127) within the specified subnet and domain. Zero is not a valid node ID.

`unsigned  subnet : 8;`
This field specifies the subnet ID (1-255) within the specified domain. Zero is not a valid subnet ID.

B.4.5 Declaration of Broadcast Address Format

```
typedef struct {
      addr_type    type;                                  // offset 0x00
      unsigned     domain                 : 1;            // offset 0x01
      unsigned                            : 1;
      unsigned     backlog                : 6;
      unsigned     rpt_timer              : 4;            // offset 0x02
      unsigned     retry                  : 4;
      unsigned                            : 4;            // offset 0x03
      unsigned     tx_timer               : 4;
      unsigned     subnet                 : 8;            // offset 0x04
} bcast_struct;
```

B.4.6 Broadcast Address Field Descriptions

`addr_type type;`
This field contains the value `BROADCAST` (3).

`unsigned  domain : 1;`
This field specifies the index into the domain table for this address (0 or 1).

`unsigned  backlog   : 6;`
This field specifies an estimate of the channel backlog that would be created by an acknowledged or request/response message broadcast using this address. It should be set to the expected number of acknowledgements or responses. For example, this might be the worst case number of nodes expected to respond to a *Query ID* message on a channel. If this is unknown, this field can be set to zero in which case a backlog of fifteen is assumed.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

`unsigned subnet : 8;`
This field specifies the subnet number (1-255) within the specified domain. The message is delivered to all nodes in this subnet. If the subnet number is 0, the message is delivered to all nodes in the domain. If request/response or acknowledged service is used with a broadcast address, the transaction completes as soon as the first response or acknowledgment is received. Subsequent responses or acknowledgments are discarded.

B.4.7 Declaration of Turnaround Address Format

```
typedef struct {
       addr_type    type;                                  // offset 0x00
       unsigned     turnaround;                            // offset 0x01
       unsigned     rpt_timer              : 4;            // offset 0x02
       unsigned     retry                  : 4;
       unsigned                            : 4;            // offset 0x03
       unsigned     tx_timer               : 4;
} turnaround_struct;
```

B.4.8 Turnaround Address Field Descriptions

`addr_type type;`
Contains the value `UNBOUND (0)`

`unsigned turnaround;`
This field contains the value one. If the turnaround field is zero, this address table entry is not in use.

B.4.9 Declaration of Protocol Processor's Address Format

NOTE This format is not used in the address table, but it may be used as a destination address for an explicitly addressed message.

```
typedef struct {
       addr_type    type;                                  // offset 0x00
       unsigned     domain                 : 1;            // offset 0x01
       unsigned                            : 7;
       unsigned     rpt_timer              : 4;            // offset 0x02
       unsigned     retry                  : 4;
       unsigned                            : 4;            // offset 0x03
       unsigned     tx_timer               : 4;
       unsigned     subnet                 : 8;            // offset 0x04
       unsigned     nid[UNIQUE_NODE_ID_LEN];               // offset 0x05
} unid_struct;
```

B.4.10 Protocol Processor Address Field Descriptions

`addr_type type;`
This field contains the value `UNIQUE_NODE_ID` (2).

`unsigned domain : 1;`
This field specifies the index into the domain table for the destination address (0 or 1). However, Unique_Node_ID-addressed messages may be sent on any domain. Unconfigured nodes will receive messages addressed in Unique_Node_ID format on any domain. When this occurs, the message is said to be received on the flexible domain, and any response or acknowledgement will be sent back on the domain in which it was received, with source subnet and node identifiers of zero.

`unsigned subnet : 8;`
This field specifies the destination subnet number (1-255) within the domain. It is only used for routing of the message, and may be set to zero if the message should pass through all routers in the domain.

`unsigned nid[Unique_Node_ID_LEN];`

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

This field specifies the unique 48-bit Unique_Node_ID of the destination node.

B.4.11 Timer Field Descriptions

`unsigned rpt_timer : 4;`
This field specifies the time interval between repetitions of an outgoing message when repeated service is used. The encoding of this field is in table 6.

`unsigned retry : 4;`
This field specifies the number of retries for acknowledged, request/response, or repeated service (0-15). The maximum number of messages sent is one more than this number.

`unsigned rcv_timer : 4;`
When the node receives a multicast (group) message, the receive timer is set to the time interval specified by this field. If a message with the same transaction ID is received before the receive timer expires, it is considered to be a retry of the previous message. The encoding of this field is in table 6.

`unsigned tx_timer : 4;`
This field specifies the time interval between retries when acknowledged or request/response service is used. The transaction retry timer is restarted when each attempt is made, and also when any acknowledgment or response (except for the last one) is received. For request/response service, the requesting node should take into account the delay necessary for the application to respond when setting the transaction timer. This is important, for example, for network management messages that write into non-volatile memory. The encoding of the transaction timer field is specified in table B3.

See Annex B.7.2 for a description of the `non_group_timer` field defined in table B3.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B3 — Encoding of timer field values in msec.

Value	rpt_timer	rcv_timer	tx_timer	non_group_timer
0	16	128	16	128
1	24	192	24	192
2	32	256	32	256
3	48	384	48	384
4	64	512	64	512
5	96	768	96	768
6	128	1,024	128	1,024
7	192	1,536	192	1,536
8	256	2,048	256	2,048
9	384	3,072	384	3,072
10	512	4,096	512	4,096
11	768	6,144	768	6,144
12	1,024	8,192	1,024	8,192
13	1,536	12,288	1,536	12,288
14	2,048	16,384	2,048	16,384
15	3,072	24,576	3,072	24,576

B.5 Network Variable Tables - Informative

There are three tables associated with network variables: the network variable configuration table, the network variable alias table, and the network variable fixed table. The network variable configuration table defines the configurable attributes of the network variables in the node. It is located in non-volatile memory so that it can be modified during node installation and is part of the network image written during node installation. The network variable alias table defines the configurable attributes of the alias network variables in the node, and is located in non-volatile memory, immediately following the network variable configuration table. The structure of the network variable fixed table within a node is implementation specific. The structures documented here are the same ones manipulated by the corresponding network management messages, so the description and use of the data in this subclause matches that described in Annex A for network variables.

```
typedef struct {
      unsigned     nv_priority           : 1;        // offset 0x00
      unsigned     nv_direction          : 1;
      unsigned     nv_selector_hi        : 6;
      unsigned     nv_selector_lo        : 8;        // offset 0x01
      unsigned     nv_turnaround         : 1;        // offset 0x02
      unsigned     nv_service            : 2;
      unsigned     nv_auth               : 1;
      unsigned     nv_addr_index         : 4;
} nv_struct;

typedef struct {
      nv_struct    nv_cnfg;                          // offset 0x00
      unsigned     primary;                          // offset 0x03
      unsigned     long host_primary;                // offset 0x04
} alias_struct;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
const nv_struct *access_nv(byte index);

void update_nv(const nv_struct *nv_entry, byte index);

const alias_struct *access_alias (byte index);

void update_alias (const alias_struct *alias_entry, byte index);



typedef struct {

      unsigned     nv_sync               : 1;          // offset 0x00
      unsigned                           : 2;
      unsigned     nv_length             : 5;
      void         *nv_address;                        // offset 0x01
} nv_fixed_struct;
```

Entries in the network variable configuration table may be read and written over the network with the *Query/Update Net Variable Configuration* network management messages. The network variable fixed table may not be written, except when downloading the application image.

The network variable alias table can have up to 62 entries.

B.5.1 Network Variable Configuration Table Field Descriptions - Informative

`unsigned nv_priority : 1;`
This bit is set to 1 if the network variable uses priority messaging.

`unsigned nv_direction : 1;`
This bit is set to one if this is an output network variable, zero if an input. This bit must not be changed by the application program or an *Update Net Variable Configuration* network management message. This bit is technically not network configuration memory data; it resides in this table for efficiency.

```
unsigned nv_selector_hi  : 6;
unsigned nv_selector_lo  : 8;
```
These two fields form a 14-bit network variable selector in the range 0 - 0x3FFF. Selector values 0x3000 - 0x3FFF are reserved for unbound network variables, with the selector value equal to 0x3FFF minus the network variable index. Selector values 0 - 0x2FFF are available for bound network variables. The input network variables on any one node must all have different selectors. The output network variables on any one node must all have different selectors.

`unsigned nv_turnaround : 1;`
This bit is set to one if this is a turnaround network variable, that is, bound to another network variable on the same node.

`unsigned nv_service  : 2;`
This field specifies the type of service used to deliver this network variable.

`unsigned nv_auth  : 1;`
This bit is set to one if this network variable uses authenticated transactions.

`unsigned nv_addr_index : 4;`
This field specifies the index into the address table for this network variable (0-14). The value 15 (0x0F) is used if the network variable is not associated with an address table entry. Multiple network variables may use the same address table index.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.5.2 Network Variable Alias Table Field Descriptions - Informative

`nv_struct nv_cnfg;`
This has the same definition as in the network variable configuration table above.

`unsigned primary;`
This is the index into the network variable configuration table. A value of 0xFF indicates that the following two bytes should be used for the index instead.

`unsigned long host_primary;`
Network variable configuration table index. This field is present only if the primary field is set to 0xFF. These bytes are only necessary if the network variable index used is larger than 254.

B.5.3 Network Variable Fixed Table Field Descriptions - Informative

`unsigned nv_sync : 1;`
This bit is set to one if this is a synchronous network variable.

`unsigned nv_length : 5;`
This field specifies the number of bytes in the network variable (1-31).

`void *nv_address;`
This field is a pointer to the location of the network variable's data in memory.

B.6 Self-Identification Structures

There are three versions of the node Self-Identification structures. Version 0 is obsolete, but is included for support of legacy nodes. Version 1 is used for nodes that do not support dynamic network variables or the extended network management commands described under the Install command in 13.7.17 of this document. Version 2 is used for nodes that implement dynamic network variables and/or nodes that implement the extended network management command set.

There are five structures associated with all three versions of the self-identification (SI) information as follows:

- fixed size SI structure;
- table containing a self-identification descriptor for each network variable;
- self-documentation string for the node;
- self-documentation information for network variables;
- Self identification data for binding and status information.

This information forms part of the application image written during node manufacture.

```
typedef struct {
      unsigned long  length;                          // offset 0x00
      unsigned     num_netvars;                       // offset 0x02
      unsigned     version;                           // offset 0x03
      union {
            struct {
                  unsigned msb_num_netvars;          // offset 0x04
                  unsigned mtag_count;               // offset 0x05
            } ver1;
            struct {
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                    unsigned mtag_count;                // offset 0x04
                } ver0;
        }  variable_part;
} snvt_struct;

typedef struct {
        unsigned      ext_rec                 : 1;      // offset 0x00
        unsigned      nv_sync                 : 1;
        unsigned      nv_polled               : 1;
        unsigned      nv_offline              : 1;
        unsigned      nv_service_type_config  : 1;
        unsigned      nv_priority_config      : 1;
        unsigned      nv_auth_config          : 1;
        unsigned      nv_config_class         : 1;
        unsigned      snvt_type_index;                  // offset 0x01
} snvt_desc_struct;
typedef struct {
        unsigned      binding_II              :1;
        unsigned      query_stats             :1;
        unsigned      alias_count             :6;
        unsigned long host_alias;                       // Host node only
} alias_field;
```

The SI header structure for version 0 and 1 is five bytes long. The NV descriptor table follows immediately afterwards, and it has one entry for every network variable declared in the application program. Version 0 format has a separate entry for each element of a network variable array. Version 1 format allows elements of a network variable array elements to share a single entry. Each entry in the NV descriptor table is two bytes long.

Following the NV descriptor table is a null-terminated text string containing the self-documentation of the node. This string may be up to 255 bytes in length. If there is no self-documentation string, the null termination byte is still present.

Following the self-documentation string for the node are extension records for those network variables that require them.

Following the extension records for the network variables are the self identification bits that reflect the state of the various binding and query status conditions.

B.6.1 SI Structure Field Descriptions

`unsigned long length;`
This field specifies the total number of bytes in the self-identification and self-documentation data structures described here.

`unsigned  num_net_vars;`
This field specifies the number of network variables declared in this node. Each element of a network variable array counts separately. In version 1 format, this byte is the least significant byte of the number of network variables declared in this node, where an NV array counts as only one item.

`unsigned  version;`
This field specifies the format of the NV structures to follow. Versions 0 and 1 are defined.

`unsigned  msb_num_net_vars;`
If the version number in the previous byte is zero, this byte is not present. If the version number is one, this byte is the most significant byte of the number of network variables declared in this node, where an NV array counts as only one item.

`unsigned  mtag_count;`

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This field specifies the number of bindable message tags declared by the application program on this node. These message tags take the first entries in the address table (see B.4).

B.6.2 NV Descriptor Table Field Descriptions

`unsigned ext_rec : 1;`
This bit is set to one if this network variable has an extension record following the node's self-documentation string.

`unsigned nv_sync : 1;`
This bit is set to one if this is a synchronous network variable.

`unsigned nv_polled : 1;`
This bit is set to one if this network variable is polled.

`unsigned nv_offline : 1;`
This bit is set to one if a network management tool should take the node off-line before this network variable is updated.

`unsigned nv_service_type_config : 1;`
This bit is set to one if the service type of this network variable (`ACKD`, `UNACKD`, `UNACKD_RPT`) may be modified by a network management message.

`unsigned nv_priority_config : 1;`
This bit is set to one if the priority of this network variable may be modified by a network management message.

`unsigned nv_auth_config : 1;`
This bit is set to one if the authentication of this network variable may be modified by a network management message.

`unsigned nv_config_class : 1;`
This bit is set to one if this is a configuration network variable whose value is stored in non-volatile memory.

`unsigned snvt_type_index;`
If this is a network variable of a standard type (i.e., a SNVT), this field specifies the type (1-250).

B.6.3 SNVT Table Extension Records

Extension records may be present for any of the network variables. If an extension record is present, the field `ext_rec` is set to one in the SNVT descriptor. The extension records appear in the order that the network variables were declared in the application program. Each extension record begins with a one-byte bit-mask defining which fields are to follow. The fields follow in the order of the bits in the bit mask defined here. These bits appear in the mask starting with the most significant bit.

`unsigned mre : 1;`
If this bit is set to one, the extension record contains an estimate of the maximum rate at which this network variable is updated. The field is an `unsigned byte` in the range 0-127 representing the rate in the range 0 - 1878. The rate is given by the formula $2^{(n/8)-5}$ messages/per second, rounded to the nearest tenth of a message per second.

`unsigned re : 1;`
If this bit is set to one, the extension record contains an estimate of the average rate at which this network variable is updated. The field is an `unsigned byte` in the range 0-127 representing the

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

rate in the range 0 - 1878.0. The rate is given by the formula $2^{(n/8)-5}$ messages/per second, rounded to the nearest tenth of a message per second.

`unsigned nm : 1;`
If this bit is set to one, the extension record contains the name of the network variable. The name is represented as a null terminated string of up to 17 bytes, or up to 22 bytes if it is a network variable array element. Each array element has its own extension record containing the name of the array, a left square bracket, one, two, or three decimal digits denoting the index of the element, and a right square bracket.

`unsigned sd : 1;`
If this bit is set to one, the extension record contains the self-documentation string for the network variable. This is a null-terminated string of up to 1023 bytes.

`unsigned nc : 1;`
If this bit is set, the extension record contains a 16-bit count of the number of network variables of this type (version 1 format only). This is used to define network variable arrays. If this bit is clear, one variable of this type is defined by this record.

B.6.4 SNVT Alias Field Descriptions

`unsigned binding_II :1;`
This bit is one if the node is using the new binding constraints. Nodes supporting these binding constraints do not require that a unique selector be assigned to output network variables in non-polling connections. For example, two different output network variables in a node supporting these constraints could be connected to the same input network variable in any type of node as long as that input network variable does not initiate polls. Note that polling of such output network variables by a network management or monitor node would have to be done using *Fetch Network Variable* network management messages rather than network variable poll messages.

`unsigned query_stats :1;`
This bit is one if the query statistics addressing mode of the *Read Memory* network management command can be used to extract the extended statistics information.

`unsigned alias_count :6;`
This field specifies the number of alias network variables used by the node (0-62). A value of 63 indicates that the next two bytes contain the actual number of alias network variables.

`unsigned long host_alias;`
This field specifies the number of alias network variables used by the node (0-4095). This field is only present if `alias_count` is 63.

B.6.5 Version 2 SI Data

The new capacities and capabilities of nodes are appended to the SI data structure of all versions. To eliminate the need for scanning all the preceding NV information in Version 0 and 1, a new command NM_GET_CAPABILITY_INFO is provided for querying information after the end of the NV extension records (i.e., beginning with alias_field). The SI data structure for version 0 and 1 is defined above. All nodes that implement dynamic network variables and/or support the extended network management protocol described in 13.7.19 of this European Standard shall implement the NM_GET_CAPABILITY_INFO command and append capability information to their current SI data as described below.

Version 0 and 1 SI data describe a node's capacities, self-documentation, and static network variables. The two versions differ in that version 0 has one NV Descriptor record per array element,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

while version 1 has one per array. The current organization is extended with two new records: the SI Extended Header Record and Capability Info Record. Below is the organization of the SI data:

Header Record	NV Descriptor Records	Node SD String	NV Extension Records	Alias Record (1 or 3 bytes)	SI Extended Header Record	Capability Info Record

Figure B1 —SI Data

B.6.5.1 Alias Record (alias_field Annex B.6.4 above)

alias_count and host_alias_count

If the number of aliases exceed 62, set alias_count to 62 and store the actual count in host_alias_count. Note that read_only_data_struct_2.alias_count is capped at 62.

B.6.5.2 SI Extended Header Record (si_header_ext)

The SI Extended Header Record contains useful fields that are not in the previous Version 0 and 1 records. The SI Extended Header Record is required when the Capability Info Record is also present, and the Alias Record is required when the SI Extended Header Record is present.

```
typedef struct
{
    NetWord  node_sd_text_length;
    NetWord  static_nv_count;    // static NV count
}si_header_ext;
```

node_sd_text_length

This is the length of the node SD string, excluding the zero terminator. It enables the network manager to obtain the length without scanning the entire string. The start of the NV extension record may be computed as the sum of the sizes of the header, nv descriptor records, node_sd_text_length, plus one byte of node sd string terminator.

static_nv_count

The number of static NVs. Each array element counts as one. This field allows the network manager to obtain the total number of discrete NVs without parsing the NV extension records for array counts.

B.6.5.3 Clarification of the use of existing fields in the snvt_struct (Annex B.6)

length

The value of this field includes the size of the SI Extended Header Record plus the Capability Info Record when using the new capabilities described in this European Standard (`si_header_ext + snvt_capability_info`).

num_netvars(version 0 & 1) and msb_num_netvars(version 1)

This count includes the sum of static and dynamic NVs, but not aliases. Alias counts are indicated in snvt_alias_field.

version

Encode this as version 0 or 1 as before.

mtag_count

This count is superseded by snvt_capability_info.static_mtag_capacity. If the number of message tags exceeds 15, this count should be set to 15. This allows the node to be functional, but with a lower capacity, when installed by a legacy network manager. By setting it to 15, it prevents the legacy network manager from allocating any address table entries for network variable bindings.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

B.6.5.4 Capability Info Record (snvt_capability_info)

The Capability Info Record describes the expanded capacities and level of support described in this document. This record is appended to the end of all versions of SI data.

```
#define    EXTCAP_FIXED_STATIC_NVS          = 0x01
#define    EXTCAP_INCOMING_GROUP_RESTRICTED  = 0x02


typedef struct
{
  NetWord  length      // length of structure, including the length field.
  unsigned ver_struct; // version number of structure.  Current version is 1
  unsigned ver_nm_min; // minimum nm version supported.  See ver_nm_max.
  unsigned ver_nm_max; // 0 or 1(supports dynamic NVs and/or extended
                       // network management commands)
  unsigned ver_binding;// 1, or 2 see Annex B.5.4
  ExtCapFlags flags[6];// an array of bits specifying capabilities
  NetWord  domain_capacity;   // maximum number of domain entries
  NetWord  address_capacity;  // maximum number of address table entries
  NetWord  static_mtag_capacity; // maximum number of statically defined
message
                                 // tags
} snvt_capability_info;
```

length

Length of this structure, including the length field.

ver_struct

Version of this data structure. Currently 1.

ver_nm_min

The minimum network management version number as defined in ver_nm_max. To support all versions, set this value to zero. But regardless of the value, a node must always support the normative network management messages.

ver_nm_max

A number representing the version of extended network management protocol supported. Value 0 indicates that the node does not support extended commands. Value 1 indicates the support for the protocol and structure defined in this document. Versions are cumulative, so each version also supports all prior versions except for obsolete commands. The version is to be incremented whenever an existing command's data structure is changed in a non-backward compatible way, or when additional commands or resources are introduced. A node must support all commands defined between ver_nm_min to ver_nm_max inclusive, and the normative network management messages defined in this European Standard.

ver_binding

A number representing the binding constraint level. Binding constraints are cumulative, so each level implies the capabilities of the prior levels. See Annex B.5.4.

flags[6]

EXTCAP_FIXED_STATIC_NVS 0x01

If this bit is not set, the name, sd string, and rate estimates of static NVs are configurable via `UPDATE_NV_INFO`. However, the change is not reflected in the SI data; a network manager must query them using `QUERY_NV_INFO` rather than parsing the SI data.

If this bit is set, static NVs' name, sd string, and rate estimates are not configurable. The nv's attributes may be queried by parsing the nv descriptor and nv extension records in the SI data, a more efficient method since more data is transferred at a time.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

EXTCAP_INCOMING_GROUP_RESTRICTED 0x02

If this bit is set, the incoming groups are restricted to the address table elements whose count is indicated in read_only_data_struct.address_count. This allows existing implementations to take advantage of some of the capabilities of the extended network management commands.

domain_capacity

The maximum number of domain entries.

address_capacity

The maximum number of address table entries.

static_mtag_capacity

The number of statically declared message tags defined. This is semantically equivalent to the same field in snvt_struct, but is expanded to 16 bits. If the number of message tags exceeds 254, set the snvt_struct mtag_count to 255; otherwise, set both fields to the same value.

B.6.5.5 Read-Only Data Structure

The read-only data structure is modified in a backward compatible manner by utilizing a spare bit in the structure to indicate that the node has extended capability information. The definitions for these two data structures is in Annex B.2.1.

read_only_data_struct

The field definitions are as defined in Annex B.2.1 except for these encodings.

nv_count

If the number of network variables exceed 62, this count should be set to 0. This prevents a legacy network manager from binding any NVs, but is less harmful than allowing a limited number of NVs to be bound. The actual number of nvs is reflected in snvt_struct.

address_capacity

If the number of address table entries exceed 15, this count should be set to 15. This allows the node to be functional, but with a lower capacity, when installed by a legacy network manager.

read_only_data_struct_2

Only nodes that do not support the QUERY_SNVT command use the read_only_data_struct_2. The field definitions are as defined in Annex B.2.1 except that one of the previously undefined bits is now defined to indicate that the node supports the new command:

NM_GET_CAPABIBILITY_INFO.

```
typedef struct read_only_data_struct_2
{
    bits                        : 2;   // reserved must be zero
    bits    alias_count         : 6;

    bits    msg_tag_count       : 4;
    bits    capability_info     : 1;   // new - supports
NM_GET_CAPABILITY_INFO
    bits                        : 3;   // unused
    unsigned reserved2 [3];            // reserved, must be zero
} read_only_data_struct_2;
```

alias_count

This count is superseded by the alias_count/host_alias fields in Annex B.6.4. If the number of aliases exceed 62, this count should be set to 62. This allows the node to be functional, but with a lower capacity, when installed by a legacy network manager that relies on this count instead of snvt_alias_field.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

mtag_count

This count is superseded by snvt_capability_info.static_mtag_capacity. If the number of message tags exceeds 15, this count should be set to 15. This allows the node to be functional, but with a lower capacity, when installed by a legacy network manager. By setting it to 15, it prevents the legacy network manager from allocating any address table entries for network variable bindings.

capability_info

When set, the node supports the NM_GET_CAPABILITY_INFO command as described in this standard.

B.7 Configuration Structure

B.7.1 General

This structure defines the hardware and transceiver properties of this node. It is located in non-volatile memory, and parts of it belong to the application image written during node manufacture, and to the network image written during node installation.

```
#define LOCATION_LEN  6
#define NUM_COMM_PARAMS 7
typedef struct {
   unsigned long channel_id;                // offset 0x00
   char          location[LOCATION_LEN];    // offset 0x02
   unsigned      comm_clock          : 5;   // offset 0x08
   unsigned      input_clock         : 3;
   unsigned      comm_type           : 3;   // offset 0x09
   unsigned      comm_pin_dir        : 5;
   unsigned      preamble_length;           // offset 0x0A
   unsigned      packet_cycle;              // offset 0x0B
   unsigned      beta2_control;             // offset 0x0C
   unsigned      xmit_interpacket;          // offset 0x0D
   unsigned      recv_interpacket;          // offset 0x0E
   unsigned      node_priority;             // offset 0x0F
   unsigned      channel_priorities;        // offset 0x10
   union {
   unsigned  xcvr_params[ NUM_COMM_PARAMS ];
   direct_param_struct dir_params;          // offset 0x11
   }
   unsigned      non_group_timer     : 4;   // offset 0x18
   unsigned      nm_auth             : 1;
   unsigned      preemption_timeout  : 3;
} config_data_struct;

typedef struct {
   unsigned      collision_detect    : 1;      // offset 0x11
   unsigned      bit_sync_threshold  : 2;
   unsigned      filter              : 2;
   unsigned      hysteresis          : 3;
   unsigned      cd_to_end_packet    : 6;      // offset 0x12
   unsigned      cd_tail             : 1;
   unsigned      cd_preamble         : 1;
} direct_param_struct;

const config_data_struct config_data;
void update_config_data(const config_data_struct * config_data);
```

The structure is 25 bytes long, and it may be read and written over the network using the *Read Memory* and *Write Memory* network management messages with `address_mode=2`. The Media Access Control processor reads the channel parameters (offsets 0x08 through 0x17) from the

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

configuration structure when the node is reset and initializes the transceiver. Therefore, after changing any of the channel parameters, the node should be reset for the changes to take effect.

B.7.2 Configuration Structure Field Descriptions

`unsigned long channel_id;`
This field specifies the ID of the channel to which this node is assigned.

`char location[LOCATION_LEN];`
The location field is used to pass a 6-byte ASCII string describing the physical location of the node to the network management tool.

`unsigned node_priority;`
This field specifies the priority slot used by the node when sending priority messages on the channel (1–255). It should not be greater than the number of priority slots on the channel. If the node has no priority slot allocated, this is zero.

`unsigned channel_priorities;`
This field specifies the number of priority slots on the channel (0-255). The slots are numbered starting at one.

`unsigned non_group_timer : 4;`
When the node receives a unicast or broadcast (non-group) message requiring a response or acknowledgment, the receive timer is set to the time interval specified by this field. If a message with the same transaction ID, priority, and source and destination address is received before the receive timer expires, it is considered to be a retry of the previous message. The encoding is of this field is in table 7. When network management messages using Unique_Node_ID addressing are received, a receive timer of 8.192 seconds is used, instead of the value of this field.

`unsigned nm_auth : 1;`
This field specifies that network management messages are to be authenticated. Setting this bit prevents the node from being configured by an unauthorized network management tool. However, network management messages received when the node is unconfigured cannot be authenticated. Before setting a node's state to configured, a network management tool should ensure that the network management authentication bit is set to the desired state.

`unsigned preemption_timeout : 3;`
This field specifies the maximum time the node will wait for a free buffer in preemption mode. A preemption mode timeout logs an error and resets the node. The encoding is as follows:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table B4 — Buffer Timeout Encoding

Encoding	Time
0	forever
1	2 secs
2	4 secs
3	6 secs
4	8 secs
5	10 secs
6	12 secs
7	14 secs

B.8 Statistics Relative Structure

The following structure definition should be used for a Read Memory command with statistics relative addressing mode:

```
Typedef struct [
       unsigned long transmission_errors;
       unsigned long transmit_tx_failures;
       unsigned long receive_tx_full;
       unsigned long lost_messages;
       unsigned long missed_messages;
       unsigned long layer2_received;
       unsigned long layer3_received;
       unsigned long layer3_transmitted;
       unsigned long transmit_tx_retries;
       unsigned long backlog_overflows;
       unsigned long late_acknowledgements;
       unsigned long collisions;
       unsigned long eeprom_lock;
} stats_struct;
```

The fields for the above structure are defined as follows:

`unsigned long transmission_errors;`
The number of CRC errors detected during packet reception. These may be due to collisions or noise on the transceiver input.

`unsigned long transmit_tx_failures;`
The number of times that the node failed to receive expected acknowledgments or responses after retrying the configured number of times.

`unsigned long receive_tx_full;`
The number of times that an incoming packet was discarded because there was no room in the transaction database.

`unsigned long lost messages;`
The number of times that an incoming packet was discarded because there was no application buffer available. If the incoming message is too large for the application buffer, an error is logged, but the lost message count is not incremented

`unsigned long missed_messages;`
The number of times that an incoming packet was discarded because there was no network buffer available.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

`unsigned long layer2 received;`
The number of Layer-2 messages received by this node. Layer-2 messages are those that have correct CRC and can be addressed to any node.

`unsigned long layer3 received;`
The number of messages transmitted from layer 3 of the protocol processor. These can include network variable updates, explicit messages, acknowledgments, retries, reminders, service pin messages, and any other type of message.

`unsigned long transmit_tx_ retries;`
The number of retries sent by this node. This does not include retries used for messages sent with repeated service.

`unsigned long backlog overflows;`
The number of times the backlog reached its maximum value of 63.

`unsigned long late_acknowledgements;`
The number of acknowledgments or responses that arrived at this node after the transmit transaction had expired.

`unsigned long collisions;`
This field specifies the number of occurrences of either collision detection or collision resolution, if those features are enabled.

`unsigned long eeprom_lock;`
The state of the EEPROM lock for the node. If this field has a value of one, then the checksummed EEPROM on the node is protected against memory write.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex C
Behavioral Characteristics
(Informative)

C.1 Channel Capacity and Throughput

In defining the key performance parameters, the following notations are used:

- bps — raw channel speed in bits/sec
- Beta — randomizing slot size (bits)
- w — size of the MAC randomizing window: 16 slots
- D_{mean}= w/2 — mean busy channel interpacket spacing: 8 slots
- p=(1/2w+p_e) — probability of packet loss from collisions and transmission errors
- p_e — probability of loss due to transmission error
- AvgPktSize — length of the packet including preamble
- C_{cost} — collision cost (2 packets)

Assuming zero collisions and zero transmission errors, the number of frames that can be transmitted while using randomizing window w = 2*D_{mean} is as follows:

$$\text{frames} = \text{bps} / (\text{Beta2} * (D_{mean} + \#\,\text{pri slots}) + \text{Beta1} + \text{AvgPktSize})\ [\text{pkts/sec}]$$

Some frames are lost because of collisions and some because of transmission errors. The packet bandwidth *Net_L3* available to the L3 Network Service is then:

$$\text{Net_L3} = \text{frames} * (1 - p)[\text{pkts/sec}]$$

When packets are lost due to collisions, the LAYER 4 and LAYER 5 protocols retransmit, thus further reducing the effective bandwidth. The *net bandwidth available to the LAYER 4 and LAYER 5 protocols* when the probability of packet loss is p, is Net_L4:

$$\text{Net_L4} = \text{Net_L3} * (1 - C_{cost} * p)[\text{TPDUs/sec}]$$

Net_L4 is maximized by choosing the proper value of w. This “net bandwidth” defines the maximum *Transaction Rate*. With the exception of authenticated transactions, a transaction within a group of N requires N PDUs to be transmitted. Assuming that all groups have the same size, this rate is:

$$\text{Trans_Rate} = \text{Net_L4} / \text{group_size}\ [\text{trans/sec}]$$

Table C1 shows the key throughput parameters for an existing implementation of a protocol processor running at 10MHz. The following assumptions are made: error free channel, correct backlog estimation, physical NPDU length of 120 bits, no priority slots configured on the channel. The number of transactions per second assumes acknowledged service. The 120 bit packet length assumes a domain id length of 1 byte, and average user data size of 3–4 bytes.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table C1 — Key Throughput Parameters

		Channel Speed	
		10 kbits/s	78 kbits/s
Capacity (Net TPDUs/Second)		43 36	388 329
L4 or L5 Transactions/Second (for Group Size N)	N = 2 N = 4 N = 8 N = 16	~ 18 ~ 9 ~ 4 ~ 2	~ 164 ~ 82 ~ 41 ~ 20
(For Single Error Free Channel @ 10 kbits/s, and @ 78 kbits/s)			

C.2 Network Metrics

For a *single channel* with backlog BL, the expected *mean Network Delay* is

$$N_{delay} = (BL/2 + 1) * busy_cycle$$

where

$$busy_cycle = Beta2 * (Dmean + \#\,pri\,slots) + Beta1 + AvgPktSize$$

In the general case, the delay over k hops is

$$N_{delay} = SUM\{(BL_i/2 + 1) * busy_cycle\}_{i=1,...,k}$$

For a channel or a network where the load exceeds throughput (i.e., BL = 1 is never true), the worst case N_{delay} is unbounded. This may happen with some very small probability but it may happen. When BL = 1 at least occasionally, N_{delay} is bounded. Its worst case value is then determined by the arrival rate distribution—the *more uniform* the distribution, the *less* is the worst case *delay*.

Assuming constant transmission error probability p_e, and a constant randomizing window w over each of the k hops, the *probability of successful delivery* is

$$P(L3) = (1 - p) * *k$$

where

$p = (1/2w + p_e)$ is the probability of packet loss

$1/2w$ is the probability of loss due to collision

p_e is the probability of the packet being corrupted by a transmission error

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Figure C1 — C1 plots the probability of successful delivery (P) of a single packet over k hops where p_e is 1 % and w is 16. This yields a probability of error over a single channel of 4.13 %. The probability of a failure in delivery is 1 - P. The probability of failure is used to calculate the optimal number of attempts to send the packet since the probability of failure is given by $(1-P)^{**attempts}$.

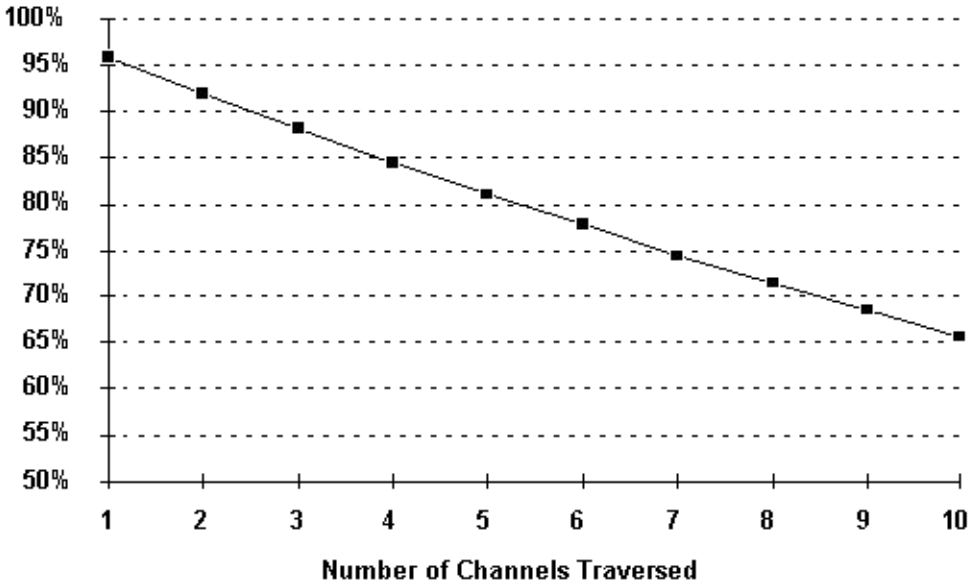

Figure C1 — Probability of Successful Delivery Over k Hops

C.3 Transaction Metrics

The *Transaction Completion Time* in the single channel, single transaction environment is:

$$T_{time} = group_size * busy_cycle + T_{np}$$

where T_{np} is the transaction processing time at the destination node. This time is negligible for transport transactions, and it may vary considerably for Request-Response transactions.

Above, collisions (or transmission errors) were ignored. When collisions and multiple transactions are considered, this time increases.

$$T_{time} = x * L4_timer + y * busy_cycle + T_{np}$$

where

x = 0, 1, ..., L4_retries

$y \leq \max(group_size, BL)$

Given that the combined collision + error probability is p, the *probability* of a transaction in a group of n *completing within k retries* is

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$P\{\text{no retry}\} = (1-p)^{n}$$

$$P\{\leq k \text{ retries}\} = \sum_{i=0}^{k} \binom{k+1}{i} p^{i} (1-p)^{(k-i+1)} \left(1-p^{(k-i+1)}\right)^{(n-1)}$$

$$k \geq 0,\ n \geq 2$$

where

p is the probability of packet loss over the channel.

P{≤ k retries} is computed as a product of the following two probabilities:

P {lose i messages in k+1 attempts}

P {successfully receive all n-1 ACKs given k-i+1 successful msgs}

The first probability is the binomial probability represented by the first three terms in the equation; the second probability is given by the last term. The above probabilities are plotted in Figure C1.

C.4 Boundary Conditions — Power-Up

Power-up has an impact on the Transaction Control sublayer: no duplicate detection is done for the first transaction following the reset (transaction number 0). For this reason, the first application operation after a reset should be idempotent.

Rebooting a learning router also has an effect: messages to a specific (subnet, node) address are routed by flooding until the router learns about the location of that subnet. This normally happens with the first acknowledgment from that subnet.

C.5 Boundary Conditions — High Load

Loads exceeding channel capacity will result in increased delays for some transactions. This may lead to timeouts and transactions being aborted with failure or only partial success.

As long as the estimated and the real backlogs match, channel capacity will stay constant at the level(s) defined in Table 8. Backlog mismatch will always reduce channel capacity. The negative effect of underestimating tends to produce excessive collisions and loss of throughput. Overestimating the backlog by a small amount has a relatively small impact on channel throughput, so backlog calculations in the protocol tend to overestimate rather than underestimate the backlog.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex D
PDU Summary
(Normative)

This appendix provides a summary description of all Protocol Data Units (PDUs). PDU syntax is specified pictorially in Figure D1.

The Physical Protocol Data Unit (PPDU) shown includes the bit sync and byte sync fields that precede the MPDU/LPDU. See relevant channel specification for bit sync and byte sync format for a given channel.

The following explanatory comments accompany Figure D1.

— Field Size. The number above each field specifies the field size in bits;

— Field Values. In order to facilitate the description of semantics, most field values are defined as symbolic ranges. A symbolic range (S0, S1, S2, ..., Sn) always maps onto a numeric range (0, 1, 2, ..., n) in the order specified;

— Bit Ordering. Bit transmission order within a byte is “most significant first”, meaning that the most significant bit is transmitted first. In the attached figures, the most significant bit is the *leftmost* bit of a byte;

— Byte Ordering. Byte transmission order is also “most significant first”, meaning that the most significant byte of a field is transmitted first. In the attached figures, the most significant byte is the leftmost byte of a field.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

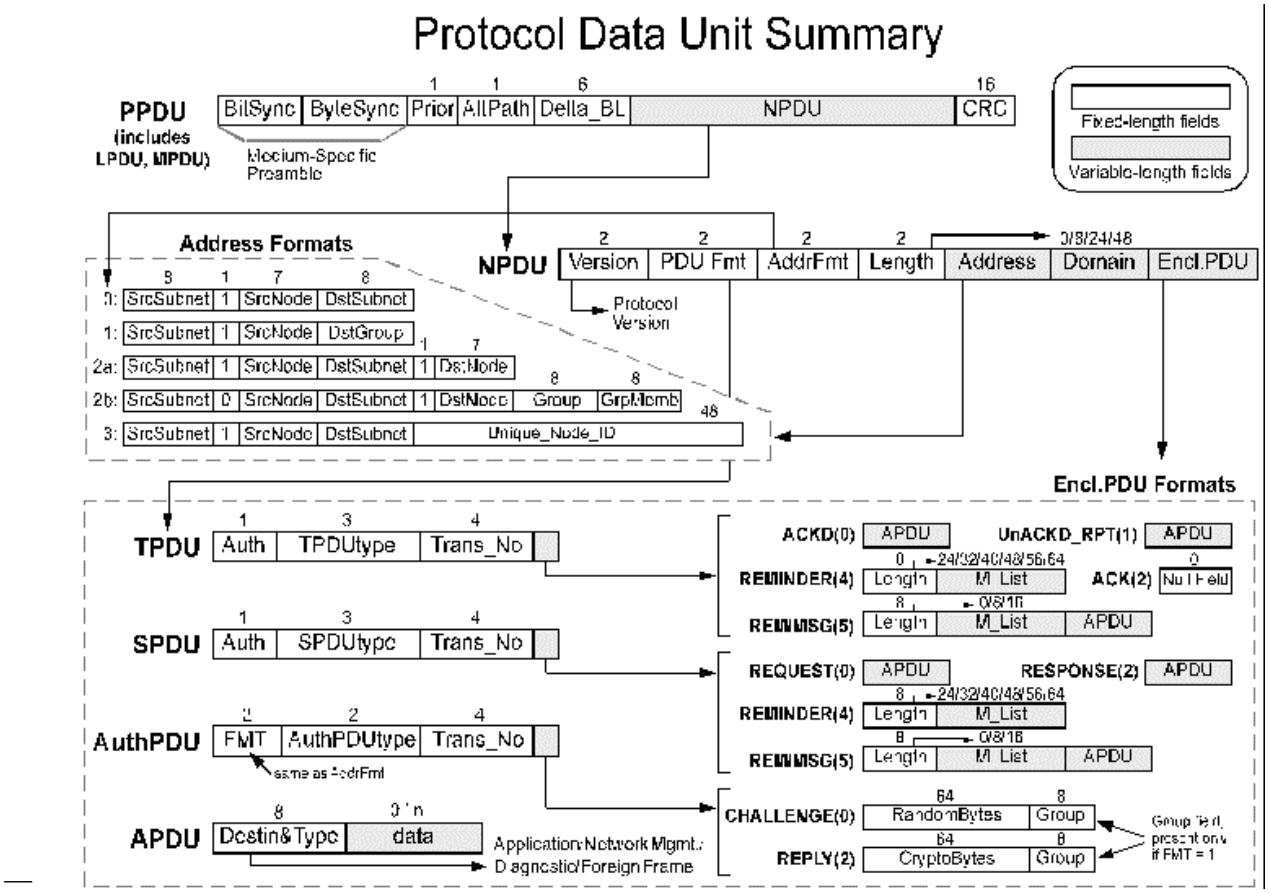

Figure D1 — Protocol PDU Summary

Annex E
Naming and Addressing
(Normative)

E.1 Address Types and Formats

The addresses of this protocol are *hierarchically* structured. There are three basic address types, with three components per address, as shown below.

(Domain, Subnet, Node)

(Domain, Subnet, Unique_Node_ID)

(Domain, Group, Member)

The syntax and semantics of the individual address components are described in E.2–E.5. The source and destination addresses are transported within every PDU. For this purpose, the basic addressing formats shown above are further combined into addressing pairs, defined in E.6.

Each address is a combined layer 3 and layer 4 address; no addressing is employed at layer 2. The domain and subnet address components are used in routing and could be called a network address. The Unique_Node_ID is a name rather than an address, since it does not change when the node is moved. Thus, address components used in routing (layer 3) and naming (layer 4) are combined in the addressing.

Address size varies in general while being invariant within each domain. The size of the domain field varies (0, 1, 3, or 6 bytes); the subnet and group fields are 8 bits wide, allowing for up to 256 groups and 255 subnets per domain (subnet 0 is a reserved value); the size of the node field is 7 bits (an all zeros field not being used); the size of the group member field is 6 bits with a range of 0..63; and the Unique_Node_ID field is 48 bits wide. This yields 2^8-1 *2^7-1 or ~2^{15} nodes per domain. Multiple domains can be used in a single system to increase the number of group addresses, nodes, etc.

E.2 Domains

The domain identifier is the first component of the addressing hierarchy. This identifier *uniquely* identifies a domain within some context. The size of the domain identifier depends on this context; 48-bit domain identifiers are provided for worldwide uniqueness. When or where this context is otherwise limited (e.g., physically, say within a single building), domain identifiers of smaller size may be used.

A domain is a *virtual network*, with all communication being limited to a single domain. One reason is that the source and destination addresses of an NPDU/TPDU must belong to the same domain (see E.3 below). Another reason is that the protocol stack does not support the equivalent of an inter-net protocol; where inter-domain communication is needed, it must be facilitated by application level gateways.

A domain is also the unit of *management* and administration. In particular, group and subnet addresses are assigned by the administrator responsible for the domain, and they have meaning only in the context of that domain.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

E.3 Subnets and Nodes

The subnet identifier is the second component of the addressing hierarchy. Its value uniquely identifies a subnet within a domain; the subnet address of 0 signifies that the subnet is undefined or unknown.

A subnet is a domain subset containing 0-127 nodes with the property that there is *no routing* within a subnet. Subnets are a routing abstraction for a channel; that is, subnets are logical channels, and need not correspond to the physical channel topology. One or more subnets may be mapped onto a single channel (or onto two or more channels connected via store and forward repeaters or bridges).

NOTE The term subnet is normally used in communications when referring to a subset of a network such that there is no routing within that subset. Subnets of this protocol have the additional property that they contain at most 127 nodes. As a result, two or more subnets may be contained in what would normally be called a subnet. In this protocol, this is referred to as a channel. That is, a channel is a physical unit of bandwidth and a subnet is a logical construct used for routing in this protocol. Note also that a channel is a physical unit of bandwidth, so a channel is composed of one or more network segments. When a channel consists of multiple segments, these segments shall be connected by physical layer repeaters so that the bandwidth of the channel remains constant regardless of the number of segments that it contains.

The node identifier identifies a (logical) node within a subnet. A physical node may belong to more than one subnet; when it does, it is assigned one (logical) node number for each subnet to which it belongs. A physical node may belong to at most two subnets, and those subnets must be in different domains.

E.4 Groups

The group identifier is the second component of the addressing hierarchy. It uniquely identifies a set of nodes within a domain. Within this set, individual members are identified by the third addressing component (i.e., the member number).

Group addressing facilitates one-to-many communication. Groups are intended to support functional addressing, and, in particular, the concept of network variables used in the application protocol.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

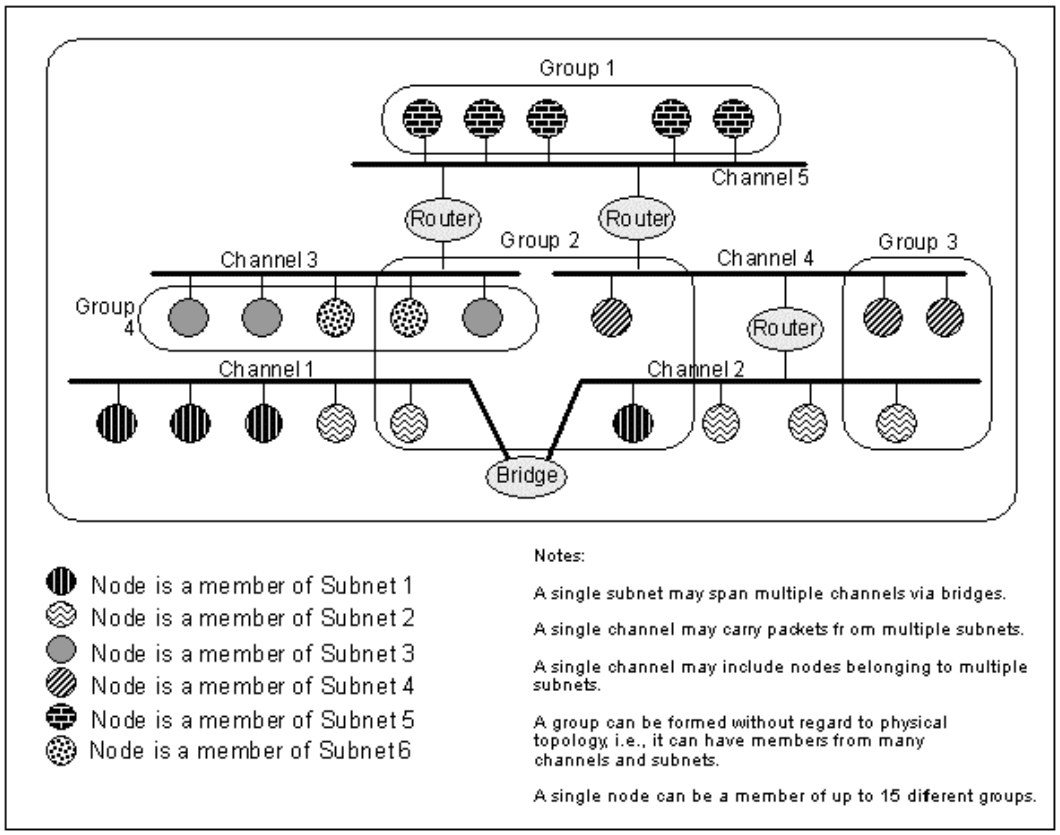

Figure E1 — Physical Topology And Logical Addressing (Single Domain)

E.5 Unique_Node_ID and Node Address Assignment

Each compliant node is assigned a unique 48-bit identifier called Unique_Node_ID. This ID is unique worldwide and is set at the time of manufacture. The value of this identifier does not change from the time of manufacture.

An unconfigured node has no assigned address apart from its 48-bit Unique_Node_ID. These unconfigured nodes receive packets from all domains, looking for and responding to any packet containing the node's 48-bit Unique_Node_ID.

A node may be assigned multiple addresses. In addition to its Unique_Node_ID, a node is usually assigned one address of type (domain, subnet, node) and zero to fifteen addresses of type (domain, group, member). A node is typically a member of only one domain; a node may be a member of up to two domains, however. Nodes that belong to multiple domains have two (domain, subnet, node) addresses—one for each domain.

When the Unique_Node_ID is used as an address, it may only be used as a destination address, and it must be combined with the domain and the source subnet addressing components (see E.6).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The process for obtaining Unique_Node_IDs is available.[2]

E.6 NPDU Addressing

For NPDU addressing, the basic addressing formats introduced in Figure E1 are combined into (Source, Destination) address pairs as defined in Table E1 and Figure E2. An NPDU carries both the source and the destination address in one of the five possible addressing formats.

Table E1 — NPDU/TPDU/SPDU Addressing - Logical Address Formats

Type	Logical Address Format	Used with TPDU/SPDU Type	
#0:	(Domain, sourceSub-Node, destSubnet)	broadcast:	Domain wide or by individual subnet
#1:	(Domain, sourceSub-Node, destGroup)	multicast:	Message or Reminder
#2a:	(Domain, sourceSub-Node, destSub-Node)	uni-cast:	Message, Reminder, or Acknowledgment
#2b:	(Domain, sourceSub-Node, destSub-Node, Group, Mem)	multicast:	Acknowledgment
#3:	(Domain, sourceSub-Node, destSub, Unique_Node_ID)	unicast:	Message or Reminder

In each address format, a source subnet of 0 means that the node does not know its subnet number. This is the condition of a node prior to configuration with network management messages. In Figure E2, note that each of the address formats contains a 7 bit source node field. The eighth bit in the source node field byte is the selector field. It is used to supply sub-variants of addressing formats. Address format #2 is the only address format using this capability. In figure E2, the numbers above each of the fields represent their bit widths. The first byte of the NPDU contains the NPDU header, that contains the protocol version, the format of the enclosed PDU, the addressing format, and the length of the domain field. The next part of the NPDU header specifies one of the four primary address formats. The final part of the NPDU header contains the length of the domain. The address field immediately follows the NPDU header.

[2] For licensing information, contact: Echelon Corporation, 4015 Meridian Avenue, San Jose, CA 94304, USA, phone +1-408-938-5234, fax: +1-408-790-3800 http://www.echelon.com.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

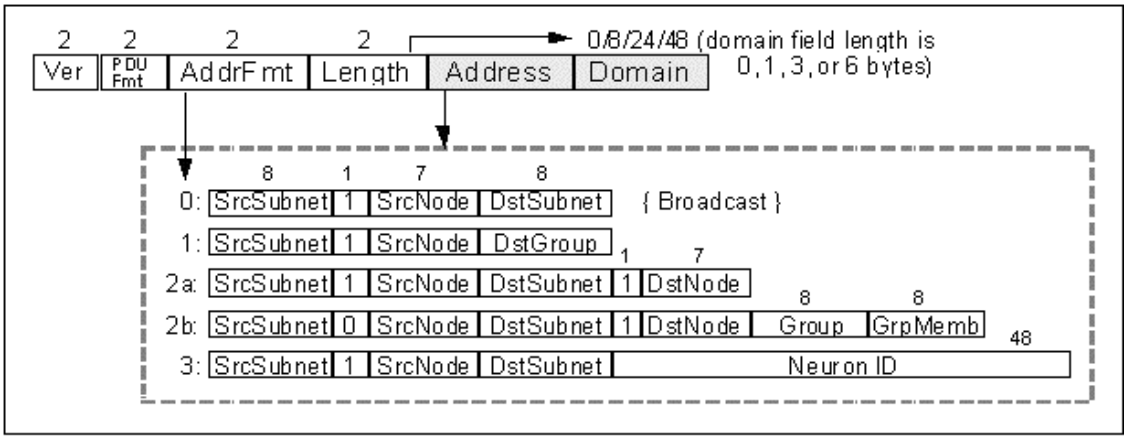

Figure E2 — NPDU/TPDU/SPDU Addressing—Physical Address Formats

The first part of the address field is the source subnet field. This field is used for routers to both learn the topology and to prevent packet looping. The combination of the source subnet field and the source node field is used for acknowledgments, authentication challenges, replies to authentication challenges, responses (when the request/response mechanism is used), and rejection of packets that are received from the same node to which they were sent. The domain field of the length specified in the NPDU header follows the source and destination address pair. Finally, the PDU of the format specified in the NPDU header.

Address format #0 facilitates domain-wide broadcast. The NPDU contains the (subnet, node) address of the source node and the destination subnet. A destination subnet of 0 implies all subnets, while a destination subnet ranging from 1 to 255 shall broadcast only to the nodes on the named destination subnet.

Address format #1 supports multicast message delivery. It uses a source address of the form (subnet, node), while the destination address is (group), implying message delivery to the entire group.

Address format #2 has two variants. With variant #2a, both the source and the destination address are of the form (subnet, node). Variant 2a is used for unicast message delivery and acknowledgments. Variant #2b supports group acknowledgments. Its source component is (subnet, node). The source and destination fields are identical to format #2a to facilitate routing. Then, appended to the source and destination fields, are the group and member numbers of the acknowledging node.

Address format #3 supports addressing by Unique_Node_ID. Since the primary intention of this addressing mode is to facilitate address assignment, Unique_Node_ID can only be used as destination address. The ID may be obtained from the node via a special network management message described in the Network Management chapter, and can also be had by initiating a Manual Service Request for the node (also described in the Network Management chapter). In cases where the destination subnet is unknown, a destination subnet of zero is used to propagate the packet throughout the network.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex F
List of patents that pertain to this European Standard
(Normative)

The US patents of Echelon Corporation that pertain to this European Standard are listed in Table F.1.

Table F.1 — Patents in the US

U.S. Patent No. 4,918,690	Network and Intelligent Cell for Providing Sensing, Bi-Directional Communications and Control
U.S. Patent No. 4,941,143	Protocol for Network Having a Plurality of Intelligent Cells
U.S. Patent No. 4,955,018	Protocol for Network Having a Plurality of Intelligent Cells
U.S. Patent No. 4,969,147	Network and Intelligent Cell for Providing Sensing, Bi-Directional Communications and Control
U.S. Patent No. 5,182,746	Transceiver Interface
U.S. Patent No. 5,297,143	Network Communication Protocol Including a Reliable Multi-Tasking Technique
U.S. Patent No. 5,319,641	Multiaccess Carrier Sensing Network Communication Protocol with Priority Messages
U.S. Patent No. 5,420,572	Configuration Device for Use in a Networked Communication System
U.S. Patent No. 5,500,852	Method and Apparatus for Network Variable Aliasing
U.S. Patent No. 5,513,324	Method and Apparatus Using Network Variables in a Multi-Node Network
U.S. Patent No. 5,519,878	Method and Apparatus for Network Node Identification
U.S. Patent No. 5,856,972	Duplicate Message Detection Method and Apparatus
U.S. Patent No. 5,737,529	Networked Variables

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The patents of Echelon Corporation registered in other countries are listed in Table F.2.

Table F.2 — Patents in Europe and other countries

U S. Patent No. 4,918,690		
Country	Patent No.	All Expire in 2008
AUS	619514	
AUSdivA	630358	
AUSdivB	640733	
CAN	1318719	
d1 CAN	1326275	
d2 CAN	1326274	
EPO	0388436	
GBR	2231426	
GBR d1	2244576	
GBR d2	2244577	
HKG	746/1996	
HKG d1	77/1993	
HKG d2	79/1993	
JPN	2066668	
KOR	121373	
RUS	2090926	
SIN based on UK Pat. 2231426	9690701-9	
SIN d1 based on UK Pat. 2244576	9290864-9	
SIN d2 based on UK Pat. 2244577	9290861-5	

(continued)

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table F.2 — Patents in Europe and other countries *(concluded)*

U.S. Patent No. 5,319,641		
EPO Serial No. 00560886 is a divisional of EP Patent No. 0560886 Grant date 04-04-2001	Countries designated: GB, FR, DE	Expires November 2011
EP Patent No. 0560886 Grant date 04-04-2001	Countries at issue: GB, FR, DE	Expires November 2011
U.S. Patent No. 5,500,852		
EPO Serial No. 95931064.0	Countries designated: FR, DE, SUI, UK	Expires March 2012
U.S. Patent No. 5,182,746		
EPO Serial No. 92917304.5	Countries designated: All EPO	Expires March 2012

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bibliography

[1] LonMark Application Layer Interoperability Guidelines, Version 3.2, Document No. 078-0120-01E, LonMark Interoperability Association. LonMark Interoperability Association, 550 Meridian Avenue, San Jose, CA USA 95126; Phone: 408-938-5266; Fax: 408-790-3800; Internet http://www.lonmark.org; E-mail: info@lonmark.org

[2] File Transfer, Document No. 005-0025-01D, Echelon Corporation. Echelon Corporation, 550 Meridian Avenue, San Jose, CA USA 95126; Phone: 408-938-5200; Fax: 408-790-3800; Internet http://www.echelon.com; E-mail: lonworks@echelon.com

[3] ISO 7498-1:1994, Information Technology - Open Systems Interconnection - Basic Reference Model:The basic model IEC Central Office, 3, rue de Varembe, PO Box 131, CH-1211 Geneva 20, Switzerland; Phone +41 22 919 02 11; Fax +41 22 919 03 00; Internet http://www.iec.ch; Email pubinfor@iec.ch

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale