ZigBee Specification

ZigBee Document 05-3474-21	
August 5, 2015	
Sponsored by: ZigBee Alliance	
Accepted by	ZigBee Alliance Board of Directors
Abstract	The ZigBee Specification describes the infrastructure and services available to applications operating on the ZigBee platform.
Keywords	ZigBee, Stack, Network, Application, Profile, Framework, Device Description, Binding, Security

August 5, 2015

This page intentionally left blank.

Notice of Use and Disclosure

ZigBee Alliance, Inc.

508 Second Street
Suite 206
Davis, CA 95616
USA

This page intentionally left blank.

Document History

ZigBee Specification History

Revision	Date	Description
	December 14, 2004	ZigBee v.1.0 draft ratified
r06	February 17, 2006	ZigBee Specification (ZigBee document number 053474r06/07) incorporating errata and clarifications: ZigBee document numbers 053920r02, 053954r02, 06084r00, and 053474r07
r07	April 28, 2006	Changes made per Editorial comments on spreadsheet
r13	October 9, 2006	ZigBee-2006 Specification (see letter ballot comments and resolution in ZigBee document 064112)
r14	November 3, 2006	ZigBee-2007 Specification (adds features described in 064270, 064269, 064268, 064281, 064319, and 064293)
r15	December 12, 2006	ZigBee-2007 Specification incorporating errata and clarifications: 074746
r16	May 31, 2007	ZigBee-2007 Specification incorporating errata and clarifications: 07819
r17	October 19, 2007	ZigBee-2007 specification incorporating errata: 075318, 075053, 075164, 075098
r18	June 16, 2009	ZigBee-2007 specification incorporating errata: 08012
r19	September 28, 2010	ZigBee-2007 specification incorporating errata described in document 105413r04
r20	September 18, 2012	ZigBee-2007 specification incorporating errata described in 11-53778-r13 and 12-0030-01
r21	August 5, 2015	ZigBee specification incorporating large updates as follows: 1. Chapter 2 – Application Layer a. Addition of Parent Announce ZDO message b. Addition of over-the-air mechanism for detecting device's implemented specification version. 2. Chapter 3 – Network Layer a. Add End device timeout protocol and aging mechanism 3. Chapter 4 – Security a. Removal of High Security b. Addition of Trust Center Link Key update services c. Cleanup of frame counter handling, d. Addition of Distributed Trust Center mode 4. Annex D – MAC And PHY Sub-layer Clarifications a. Update to 802.15.4-2011 5. Annex G – Inter-PAN

Revision	Date	Description
		a. Formalization of Inter-PAN frame formats and service handling. 6. Annex H – Inter-PAN Test Vectors a. Addition of Green Power Inter-PAN test vectors.

This page intentionally left blank.

Table of Contents

List of Tables

List of Figures

This page intentionally left blank.

CHAPTER 1 ZIGBEE PROTOCOL OVERVIEW

1.1 Protocol Description

The ZigBee Alliance has developed a very low-cost, very low-power-consumption, two-way, wireless communications standard. Solutions adopting the ZigBee standard will be embedded in consumer electronics, home and building automation, industrial controls, PC peripherals, medical sensor applications, toys, and games.

1.1.1 Scope

This document contains specifications, interface descriptions, object descriptions, protocols and algorithms pertaining to the ZigBee protocol standard, including the application support sub-layer (APS), the ZigBee device objects (ZDO), ZigBee device profile (ZDP), the application framework, the network layer (NWK), and ZigBee security services.

1.1.2 Purpose

The purpose of this document is to provide a definitive description of the ZigBee protocol standard as a basis for future implementations, such that any number of companies incorporating the ZigBee standard into platforms and devices on the basis of this document will produce interoperable, low-cost, and highly usable products for the burgeoning wireless marketplace.

1.1.3 Stack Architecture

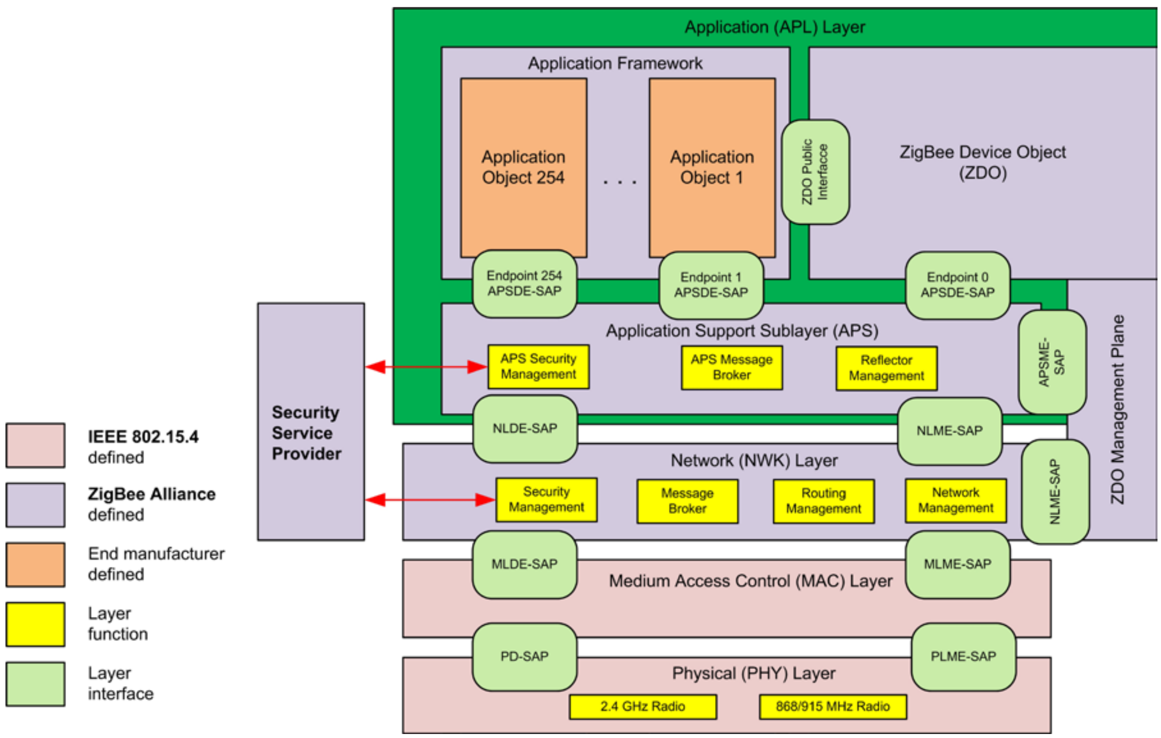
The ZigBee stack architecture is made up of a set of blocks called layers. Each layer performs a specific set of services for the layer above. A data entity provides a data transmission service and a management entity provides all other services. Each service entity exposes an interface to the upper layer through a service access point (SAP), and each SAP supports a number of service primitives to achieve the required functionality.

The IEEE 802.15.4 standard defines the two lower layers: the physical (PHY) layer and the medium access control (MAC) sub-layer. The ZigBee Alliance builds on this foundation by providing the network (NWK) layer and the framework for the application layer. The application layer framework consists of the application support sub-layer (APS) and the ZigBee device objects (ZDO). Manufacturer-defined application objects use the framework and share APS and security services with the ZDO.

The PHY layer operates in two separate frequency ranges: 868/915 MHz and 2.4 GHz. The lower frequency PHY layer covers both the 868 MHz European band and the 915 MHz band, used in countries such as the United States and Australia. The higher frequency PHY layer is used virtually worldwide. A complete description of the PHY layers can be found in [B1].

The MAC sub-layer controls access to the radio channel using a CSMA-CA mechanism. Its responsibilities may also include transmitting beacon frames, synchronization, and providing a reliable transmission mechanism. A complete description of the IEEE 802.15.4 MAC sub-layer can be found in [B1]. Figure 1.1 represents the outline of the ZigBee Stack Architecture.

Figure 1.1 Outline of the ZigBee Stack Architecture



1.1.4 Network Topology

The ZigBee network layer (NWK) supports star, tree, and mesh topologies. In a star topology, the network is controlled by one single device called the ZigBee coordinator. The ZigBee coordinator is responsible for initiating and maintaining the devices on the network. All other devices, known as end devices, directly communicate with the ZigBee coordinator. In mesh and tree topologies, the ZigBee coordinator is responsible for starting the network and for choosing certain key network parameters, but the network may be extended through the use of ZigBee routers. In tree networks, routers move data and control messages through the network using a hierarchical routing strategy. Tree networks may employ beacon-oriented communication as described in the IEEE 802.15.4 specification. Mesh networks allow full peer-to-peer communication. ZigBee routers in mesh networks do not currently emit regular IEEE 802.15.4 beacons. This specification describes only intra-PAN networks, that is, networks in which communications begin and terminate within the same network.

1.2 Conventions and Abbreviations

1.2.1 Symbols and Notation

Notation follows from ANSI X9.63-2001, §2.2 [B7].

1.2.2 Integers, Octets, and Their Representation

Throughout Annexes A through D, the representation of integers as octet strings and of octet strings as binary strings shall be fixed. All integers shall be represented as octet strings in most-significant-octet first order. This representation conforms to the convention in Section 4.3 of [B7]. All octets shall be represented as binary strings in most-significant-bit first order.

1.2.3 Transmission Order

Unless otherwise indicated, the transmission order of all frames in this specification follows the conventions used in [B1]:

- Frame formats are depicted in the order in which they are transmitted by the PHY layer—from left to right—where the leftmost bit is transmitted first in time.
- Bits within each field are numbered from 0 (leftmost, and least significant) to k-1 (rightmost, and most significant), where the length of the field is k bits.
- Fields that are longer than a single octet are sent to the PHY in order from the octet containing the lowest numbered bits to the octet containing the highest- numbered bits.

1.2.4 Strings and String Operations

A string is a sequence of symbols over a specific set (for example, the binary alphabet {0,1} or the set of all octets). The length of a string is the number of symbols it contains (over the same alphabet). The empty string has length 0. The right-concatenation of two strings x and y of length m and n respectively (notation: $x \| y$), is the string z of length $m+n$ that coincides with x on its leftmost m symbols and with y on its right-most n symbols. An octet is a symbol string of length 8. In our context, all octets are strings over the binary alphabet.

1.3 Acronyms and Abbreviations

For the purposes of this standard, the following acronyms and abbreviations apply:

Acronym or Abbreviation	**Definition**
AIB	Application support layer information base
AF	Application framework
APDU	Application support sub-layer protocol data unit
APL	Application layer
APS	Application support sub-layer
APSDE	Application support sub-layer data entity
APSDE-SAP	Application support sub-layer data entity – service access point
APSME	Application support sub-layer management entity

Acronym or Abbreviation	Definition
APSME-SAP	Application support sub-layer management entity – service access point
ASDU	APS service data unit
BRT	Broadcast retry timer
BTR	Broadcast transaction record
BTT	Broadcast transaction table
CCM*	Carrier sense multiple access – collision avoidance
CSMA-CA	Carrier sense multiple access – collision avoidance.
EPID	Extended PAN ID
FFD	Full function device
GPD	Green Power Device
GPDF	Green Power Device Frame
GPEP	Green Power Endpoint
GTS	Guaranteed time slot
HDR	Header
IB	Information base
LQI	Link quality indicator
LR-WPAN	Low rate wireless personal area network
MAC	Medium access control
MCPS-SAP	Medium access control common part sub-layer service access point
MIC	Message integrity code
MLME-SAP	Medium access control sub-layer management entity service access point
MSC	Message sequence chart

Acronym or Abbreviation	Definition
MSDU	Medium access control sub-layer service data unit
MSG	Message service type
NBDT	Network broadcast delivery time
NHLE	Next higher layer entity
NIB	Network layer information base
NLDE	Network layer data entity
NLDE-SAP	Network layer data entity – service access point
NLME	Network layer management entity
NLME-SAP	Network layer management entity – service access point
NPDU	Network layer protocol data unit
NSDU	Network service data unit
NWK	Network
OSI	Open systems interconnection
PAN	Personal area network
PD-SAP	Physical layer data service access point
PDU	Protocol data unit
PHY	Physical layer
PIB	Personal area network information base
PLME-SAP	Physical layer management entity – service access point
POS	Personal operating space
QOS	Quality of service
RFD	Reduced function device

Acronym or Abbreviation	Definition
RREP	Route reply
RREQ	Route request
RN	Routing node
SAP	Service access point
SKG	Secret key generation
SSP	Security services provider
SSS	Security services specification
WPAN	Wireless personal area network
XML	Extensible markup language
ZB	ZigBee
ZDO	ZigBee device object

1.4 Glossary

1.4.1 Definitions

1.4.1.1 Conformance Levels

The conformance level definitions shall follow those in clause 13, section 1 of [B14].

Expected: A key word used to describe the behavior of the hardware or software in the design models assumed by this Specification. Other hardware and software design models may also be implemented.

May: A key word indicating a course of action permissible within the limits of the standard (may equals is permitted to).

Shall: A key word indicating mandatory requirements to be strictly followed in order to conform to the standard; deviations from shall are prohibited (*shall* equals *is required to*).

Should: A key word indicating that, among several possibilities, one is recommended as being particularly suitable, without mentioning or excluding others; that a certain course of action is preferred but not necessarily required; or, that (in the negative form) a certain course of action is deprecated but not prohibited (*should* equals *is recommended that*).

Reserved Codes: A set of codes that are defined in this specification, but not otherwise used. Future specifications may implement the use of these codes. A product implementing this specification shall not generate these codes.

Reserved Fields: A set of fields that are defined in this specification, but are not otherwise used. Products that implement this specification shall zero these fields and shall make no further assumptions about these fields nor perform processing based on their content.

ZigBee Protocol Version: The name of the ZigBee protocol version governed by this specification. The protocol version sub-field of the frame control field in the NWK header of all ZigBee Protocol Stack frames conforming to this specification shall have a value of 0x02 for all ZigBee frames, and a value of 0x03 for the ZigBee Green Power frames. The protocol version support required by various ZigBee specification revisions appears in Table 1.1.

Table 1.1 ZigBee Protocol Versions

Specification	Protocol Version	Comment
ZigBee Green Power	0x03	ZigBee Green Power feature. See Annex G.
ZigBee Pro ZigBee 2006	0x02	Backwards compatibility not required. ZigBee Pro and ZigBee 2006 compatibility required.
ZigBee 2004	0x01	Original ZigBee version.

A ZigBee device that conforms to this version of the specification may elect to provide backward compatibility with the 2004 revision of the specification. If it so elects, it shall do so by supporting, in addition to the frame formats and features described in this specification version, all frame formats and features as specified in the older version. [All devices in an operating network, regardless of which revisions of the ZigBee specification they support internally, shall, with respect to their external, observable behavior, consistently conform to a single ZigBee protocol version.] A single ZigBee network shall not contain devices that conform, in terms of their external behavior, to multiple ZigBee protocol versions. [The protocol version of the network to join shall be determined by a backwardly compatible device in examining the beacon payload prior to deciding to join the network; or shall be established by the application if the device is a ZigBee coordinator.] A ZigBee device conforming to this specification may elect to support only protocol version 0x02, whereby it shall join only networks that advertise commensurate beacon payload support. A ZigBee device that conforms to this specification shall discard all frames carrying a protocol version sub-field value other than 0x01, 0x02, or0x03. Itshall process only protocol versions of 0x01 or 0x02, consistent with the protocol version of the network that the device participates within. A ZigBee device that conforms to this specification shall pass the frames carrying the protocol version sub-field value 0x03 to the Interpan APS (see Annex G), if it supports the ZigBee Green Power, otherwise it shall drop them.

1.4.1.2 ZigBee Definitions

For the purposes of this standard, the following terms and definitions apply. Terms not defined in this section can be found in [B1] or in [B7].

Access control list: This is a table used by a device to determine which devices are authorized to perform a specific function. This table may also store the security material (for example, cryptographic keys, frame counts, key counts, security level information) used for securely communicating with other devices.

Active network key: This is the key used by a ZigBee device to secure outgoing NWK frames and that is available for use to process incoming NWK frames.

Alternate network key: This is a key available to process incoming NWK frames in lieu of the active network key.

Application domain: This describes a broad area of applications, such as building automation.

Application key: This is a link key transported by the Trust center to a device for the purpose of securing end-to-end communication.

Application object: This is a component of the top portion of the application layer defined by the manufacturer that actually implements the application.

Application profile: This is a collection of device descriptions, which together form a cooperative application. For instance, a thermostat on one node communicates with a furnace on another node. Together, they cooperatively form a heating application profile.

Application support sub-layer protocol data unit: This is a unit of data that is exchanged between the application support sub-layers of two peer entities.

APS command frame: This is a command frame from the APSME on a device addressed to the peer entity on another device.

Association: This is the service provided by the IEEE 802.15.4 MAC sub-layer that is used to establish membership in a network.

Attribute: This is a data entity which represents a physical quantity or state. This data is communicated to other devices using commands.

Beacon-enabled personal area network: This is a personal area network containing at least one device that transmits beacon frames at a regular interval.

Binding: This is the creation of a unidirectional logical link between a source endpoint/cluster identifier pair and a destination endpoint, which may exist on one or more devices.

Broadcast: This is the transmission of a message to every device in a particular PAN belonging to one of a small number of statically defined broadcast groups, for example all routers, and within a given transmission radius measured in hops.

Broadcast jitter: This is a random delay time introduced by a device before relaying a broadcast transaction.

Broadcast transaction record: This is a local receipt of a broadcast message that was either initiated or relayed by a device.

Broadcast transaction table: This is a collection of broadcast transaction records.

Cluster: This is an application message, which may be a container for one or more attributes. As an example, the ZigBee Device Profile defines commands and responses. These are contained in Clusters with the cluster identifiers enumerated for each command and response. Each ZigBee Device Profile message is then defined as a cluster. Alternatively, an application profile may create sub-types within the cluster known as attributes. In this case, the cluster is a collection of attributes specified to accompany a specific cluster identifier (sub-type messages.)

Cluster identifier: This is a reference to an enumeration of clusters within a specific application profile or collection of application profiles. The cluster identifier is a 16-bit number unique within the scope of each application profile and identifies a specific cluster. Conventions may be established across application profiles for common definitions of cluster identifiers whereby each application profile defines a set of cluster identifiers identically. Cluster identifiers are designated as inputs or outputs in the simple descriptor for use in creating a binding table.

Coordinator: This is an IEEE 802.15.4 device responsible for associating and disassociating devices into its PAN. A coordinator must be a full-function device (FFD).

Data integrity: This is assurance that the data has not been modified from its original form.

Data key: This is a key derived from a link key used to protect data messages.

Device: This is any entity that contains an implementation of the ZigBee protocol stack.

Device application: This is a special application that is responsible for Device operation. The device application resides on endpoint 0 by convention and contains logic to manage the device's networking and general maintenance features. Endpoints 241-254 are reserved for use by the Device application or common application function agreed within the ZigBee Alliance.

Device description: This is a description of a specific device within an application profile. For example, the light sensor device description is a member of the home automation application profile. The device description also has a unique identifier that is exchanged as part of the discovery process.

Direct addressing: This is a mode of addressing in which the destination of a frame is completely specified in the frame itself.

Direct transmission: This is a frame transmission using direct addressing.

Disassociation: This is the service provided by the IEEE 802.15.4 MAC sub-layer that is used to discontinue the membership of a device in a network.

End application: This is for applications that reside on endpoints 1 through 254 on a Device. The end applications implement features that are non-networking and ZigBee protocol related. Endpoints 241 through 254 shall only be used by the End application with approval from the ZigBee Alliance. The Green Power cluster, if implemented, SHALL use endpoint 242.

End device binding: This is the procedure for creating or removing a binding link initiated by each of the end devices that will form the link. The procedure may or may not involve user intervention.

Endpoint: This is a particular component within a unit. Each ZigBee device may support up to 254 such components.

Endpoint address: This is the address assigned to an endpoint. This address is assigned in addition to the unique, 64-bit IEEE address and 16-bit network address.

Extended PAN ID: This is the globally unique 64-bit PAN identifier of the network. This identifier should be unique among the PAN overlapping in a given area. This identifier is used to avoid PAN ID conflicts between distinct networks.

Information base: This is a collection of variables that define certain behavior in a layer. These variables can be specified or obtained from a layer through its management service.

Key establishment: This is a mechanism that involves the execution of a protocol by two devices to derive a mutually shared secret key.

Key-load key: This is a key derived from a link key used to protect key transport messages carrying a link key.

Key transport: This is a mechanism for communicating a key from one device to another device or other devices.

Key-transport key: This is a key derived from a link key used to protect key transport messages carrying a key.

Key update: This is a mechanism implementing the replacement of a key shared amongst two or more devices by means of another key available to that same group.

Local device: This is the initiator of a ZDP command.

Link key: This is a key that is shared exclusively between two, and only two, peer application-layer entities within a PAN.

Mesh network: This is a network in which the routing of messages is performed as a decentralized, cooperative process involving many peer devices routing on each other's behalf.

Multicast: This is a transmission to every device in a particular PAN belonging to a dynamically defined multicast group, and within a given transmission radius measured in hops.

Multihop network: This is a network, in particular a wireless network, in which there is no guarantee that the transmitter and the receiver of a given message are connected or linked to each other. This implies that intermediate devices must be used as routers.

Non-beacon-enabled personal area network: This is a personal area network that does not contain any devices that transmit beacon frames at a regular interval.

Neighbor table: This is a table used by a ZigBee device to keep track of other devices within the POS.

Network address: This is the address assigned to a device by the network layer and used by the network layer for routing messages between devices.

Network broadcast delivery time: This is the time required by a broadcast transaction to reach every device of a given network.

Network manager: This is a ZigBee device that implements network management functions as described in Clause 3, including PAN ID conflict resolution and frequency agility measures in the face of interference.

Network protocol data unit: This is a unit of data that is exchanged between the network layers of two peer entities.

Network service data unit: This is the information that is delivered as a unit through a network service access point.

Node: This is a collection of independent device descriptions and applications residing in a single unit and sharing a common 802.15.4 radio.

Normal operating state: This is the processing which occurs after all startup and initialization processing has occurred and prior to initiation of shutdown processing.

NULL: a parameter or variable value that means unspecified, undefined, or unknown. The exact value of NULL is implementation-specific, and must not conflict with any other parameters or values.

Octet: eight bits of data, used as a synonym for a byte.

OctetDuration: transmission time (in seconds) of an octet on PHY layer. This time is calculated as 8/phyBitRate where phyBitRate can be found in Table 1 of [B1]. To get milliseconds from N OctetDurations for 2.4 GHz the follow formula has to be used: N*(8/250000)*1000 where 250000 bit rate on 2.4 GHz and 8 number of bits in one octet.

One-way function: a function whose forward computation is much easier to perform than its inverse.

Orphaned device: a device, typically a ZigBee end device that has lost communication with the ZigBee device through which it has its PAN membership.

PAN coordinator: the principal controller of an IEEE 802.15.4-based network that is responsible for network formation. The PAN coordinator must be a full function device (FFD).

PAN information base: a collection of variables in the IEEE 802.15.4 standard that are passed between layers, in order to exchange information. This database may include the access control list, which stores the security material.

Personal operating space: the area within reception range of a single device.

Private method: attributes and commands which are accessible to ZigBee device objects only and unavailable to the end applications.

Protocol data unit: the unit of data that is exchanged between two peer entities.

Public method: attributes and commands which are accessible to end applications.

Radio: the IEEE 802.15.4 radio that is part of every ZigBee device.

Remote device: the target of a ZDP command.

Route discovery: an operation in which a ZigBee coordinator or ZigBee router attempts to discover a route to a remote device by issuing a route request command frame.

Route discovery table: a table used by a ZigBee coordinator or ZigBee router to store temporary information used during route discovery.

Route reply: a ZigBee network layer command frame used to reply to route requests.

Route request: a ZigBee network layer command frame used to discover paths through the network over which subsequent messages may be delivered.

Routing table: a table in which a ZigBee coordinator or ZigBee router stores information required to participate in the routing of frames.

Service discovery: the ability of a device to locate services of interest.

Stack profile: an agreement by convention outside the scope of the ZigBee specification on a set of additional restrictions with respect to features declared optional by the specification itself.

Trust center: the device trusted by devices within a ZigBee network to distribute keys for the purpose of network and end-to-end application configuration management.

Unicast: the transmission of a message to a single device in a network.

ZigBee coordinator: an IEEE 802.15.4 PAN coordinator.

ZigBee device object: the portion of the application layer responsible for defining the role of the device within the network (for example, ZigBee coordinator or end device), initiating and/or responding to binding and discovery requests, and establishing a secure relationship between network devices.

ZigBee end device: an IEEE 802.15.4 RFD or FFD participating in a ZigBee network, which is neither the ZigBee coordinator nor a ZigBee router.

ZigBee router: an IEEE 802.15.4 FFD participating in a ZigBee network, which is not the ZigBee coordinator but may act as an IEEE 802.15.4 coordinator within its personal operating space, that is capable of routing messages between devices and supporting associations.

1.5 References

The following standards contain provisions, which, through reference in this document, constitute provisions of this standard. Normative references are given in ZigBee/IEEE References and Normative References and informative references are given in Informative References At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the references, as indicated in this section.

1.5.1 ZigBee/IEEE References

[B1] 802.15.4-2011, IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)

[B3] Document 03-285r00: Suggestions for the Improvement of the IEEE 802.15.4 Standard, July 2003.

[B4] Document 09-5499r26: Green Power specification

1.5.2 Normative References

[B5] ISO/IEC 639-1:2002 Codes for the representation of names of languages — Part 1: Alpha-2 code.

[B6] ISO/IEC 646:199 Information technology — ISO 7-bit coded character set for information interchange.

[B7] ANSI X9.63-2001, Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography, American Bankers Association, November 20, 2001. Available from http://www.ansi.org.

[B8] FIPS Pub 197, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, US Department of Commerce/N.I.S.T, Springfield, Virginia, November 26, 2001. Available from http://csrc.nist.gov/.

[B9] FIPS Pub 198, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication 198, US Department of Commerce/N.I.S.T., Springfield, Virginia, March 6, 2002. Available from http://csrc.nist.gov/.

[B10] ISO/IEC 9798-2, Information Technology - Security Techniques — Entity Authentication Mechanisms — Part 2: Mechanisms Using Symmetric Encipherment Algorithms, International Standardization Organization, Geneva, Switzerland, 1994 (first edition). Available from http://www.iso.org/.

[B11] NIST Pub 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation — Methods and Techniques, NIST Special Publication 800-38A, 2001 Edition, US Department of Commerce/N.I.S.T., December 2001. Available from http://csrc.nist.gov/.

[B12] NIST, Random Number Generation and Testing. Available from http://csrc.nist.gov/rng/.

1.5.3 Informative References

[B13] FIPS Pub 140-2, Security requirements for Cryptographic Modules, US Department of Commerce/N.I.S.T., Springfield, Virginia, June 2001 (supersedes FIPS Pub 140-1). Available from http://csrc.nist.gov/.

[B14] IEEE Standards Style Manual, published and distributed in May 2000 and revised on September 20, 2001. Available from http://standards.ieee.org/guides/style/.

[B15] ISO/IEC 7498-1:1994 Information technology — Open systems interconnection — Basic reference model: The basic model.

[B16] ISO/IEC 10731:1994, Information technology — Open Systems Interconnection — Conventions for the definition of OSI services.

[B17] ISO/IEC 9646-1:1991, Information technology — Open systems Interconnection — Conformance testing methodology and framework — Part 1: General concepts.

[B18] ISO/IEC 9646-7:1995, Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 7. Implementation conformance statements.

[B19] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, Boca Raton: CRC Press, 1997.

[B20] FIPS Pub 113, Computer Data Authentication, Federal Information Processing Standard Publication 113, US Department of Commerce/N.I.S.T., May 30, 1985. Available from http://csrc.nist.gov/.

[B21] R. Housley, D. Whiting, N. Ferguson, Counter with CBC-MAC (CCM), submitted to N.I.S.T., June 3, 2002. Available from http://csrc.nist.gov/encryption/modules/proposedmodes/.

[B22] J. Jonsson, On the Security of CTR + CBC-MAC, in Proceedings of Selected Areas in Cryptography — SAC 2002, K. Nyberg, H. Heys, Eds., Lecture Notes in Computer Science, Vol. 2595, pp. 76-93, Berlin: Springer, 2002.

[B23] J. Jonsson, On the Security of CTR + CBC-MAC, NIST Mode of Operation — Additional CCM Documentation. Available from http://csrc.nist.gov/encryption/modes/proposedmodes/.

[B24] P. Rogaway, D. Wagner, A Critique of CCM, IACR ePrint Archive 2003-070, April 13, 2003.

[B25] ZigBee Document 053298- CSG Framework Profile Identifier Database

[B26] ZigBee Document 09-5499r22 – Green Power Specification

This page intentionally left blank.

CHAPTER 2 APPLICATION LAYER SPECIFICATION

2.1 General Description

The ZigBee stack architecture includes a number of layered components including the IEEE 802.15.4 Medium Access Control (MAC) layer, Physical (PHY) layer, and the ZigBee Network (NWK) layer. Each component provides an application with its own set of services and capabilities. Although this chapter may refer to other components within the ZigBee stack architecture, its primary purpose is to describe the component labeled Application (APL) Layer shown in Figure 1.1 of “ZigBee Protocol Overview.”

As shown in Figure 1.1, the ZigBee application layer consists of the APS sub-layer, the ZDO (containing the ZDO management plane), and the manufacturer-defined application objects.

2.1.1 Application Support Sub-Layer

The application support sub-layer (APS) provides an interface between the network layer (NWK) and the application layer (APL) through a general set of services that are used by both the ZDO and the manufacturer-defined application objects. The services are provided by two entities:

- The APS data entity (APSDE) through the APSDE service access point (APSDE-SAP).
- The APS management entity (APSME) through the APSME service access point (APSME-SAP).

The APSDE provides the data transmission service between two or more application entities located on the same network.

The APSME provides a variety of services to application objects including security services and binding of devices. It also maintains a database of managed objects, known as the APS information base (AIB).

2.1.2 Application Framework

The application framework in ZigBee is the environment in which application objects are hosted on ZigBee devices.

Up to 254 distinct application objects can be defined, each identified by an endpoint address from 1 to 254. Two additional endpoints are defined for APSDE-SAP usage: endpoint 0 is reserved for the data interface to the ZDO, and endpoint 255 is reserved for the data interface function to broadcast data to all application objects. Endpoints 241-254 are assigned by the ZigBee Alliance and shall not be used without approval. The Green Power cluster, if implemented, shall use endpoint 242.

2.1.2.1 Application Profiles

Application profiles are agreements for messages, message formats, and processing actions that enable developers to create an interoperable, distributed application employing application entities that reside on separate devices. These application profiles enable applications to send commands, request data, and process commands and requests.

2.1.2.2 Clusters

Clusters are identified by a cluster identifier, which is associated with data flowing out of, or into, the device. Cluster identifiers are unique within the scope of a particular application profile.

2.1.3 ZigBee Device Objects

The ZigBee device objects (ZDO), represent a base class of functionality that provides an interface between the application objects, the device profile, and the APS. The ZDO is located between the application framework and the application support sub-layer. It satisfies common requirements of all applications operating in a ZigBee protocol stack. The ZDO is responsible for the following:

- Initializing the application support sub-layer (APS), the network layer (NWK), and the Security Service Provider.
- Assembling configuration information from the end applications to determine and implement discovery, security management, network management, and binding management.

The ZDO presents public interfaces to the application objects in the application framework layer for control of device and network functions by the application objects. The ZDO interfaces with the lower portions of the ZigBee protocol stack, on endpoint 0, through the APSDE-SAP for data, and through the APSME-SAP and NLME-SAP for control messages. The public interface provides address management of the device, discovery, binding, and security functions within the application framework layer of the ZigBee protocol stack. The ZDO is fully described in clause 2.5.

2.1.3.1 Device Discovery

Device discovery is the process whereby a ZigBee device can discover other ZigBee devices. There are two forms of device discovery requests: IEEE address requests and NWK address requests. The IEEE address request is unicast to a particular device and assumes the NWK address is known. The NWK address request is broadcast and carries the known IEEE address as data payload.

2.1.3.2 Service Discovery

Service discovery is the process whereby the capabilities of a given device are discovered by other devices. Service discovery can be accomplished by issuing a query for each endpoint on a given device or by using a match service feature (either broadcast or unicast). The service discovery facility defines and utilizes various descriptors to outline the capabilities of a device.

Service discovery information may also be cached in the network in the case where the device proffering a particular service may be inaccessible at the time the discovery operation takes place.

2.2 ZigBee Application Support (APS) Sub-Layer

2.2.1 Scope

This clause specifies the portion of the application layer providing the service specification and interface to both the manufacturer-defined application objects and the ZigBee device objects. The specification defines a data service to allow the application objects to transport data, and a management service providing mechanisms for binding. In addition, it also defines the application support sub-layer frame format and frame-type specifications.

2.2.2 Purpose

The purpose of this clause is to define the functionality of the ZigBee application support (APS) sub-layer. This functionality is based on both the driver functionality necessary to enable correct operation of the ZigBee network layer and the functionality required by the manufacturer-defined application objects.

2.2.3 Application Support (APS) Sub-Layer Overview

The application support sub-layer provides the interface between the network layer and the application layer through a general set of services for use by both the ZigBee device object (ZDO) and the manufacturer-defined application objects. These services are offered via two entities: the data service and the management service. The APS data entity (APSDE) provides the data transmission service via its associated SAP, the APSDE-SAP. The APS management entity (APSME) provides the management service via its associated SAP, the APSME-SAP, and maintains a database of managed objects known as the APS information base (AIB).

2.2.3.1 Application Support Sub-Layer Data Entity (APSDE)

The APSDE shall provide a data service to the network layer and both ZDO and application objects to enable the transport of application PDUs between two or more devices. The devices themselves must be located on the same network.

The APSDE will provide the following services:

- **Generation of the application level PDU (APDU):** The APSDE shall take an application PDU and generate an APS PDU by adding the appropriate protocol overhead.
- **Binding:** Once two devices are bound, the APSDE shall be able to transfer a message from one bound device to the second device.
- **Group address filtering:** The ability to filter group-addressed messages based on endpoint group membership.
- **Reliable transport:** Increases the reliability of transactions above that available from the NWK layer alone by employing end-to-end retries.
- **Duplicate rejection:** Messages offered for transmission will not be received more than once.
- **Fragmentation:** Enables segmentation and reassembly of messages longer than the payload of a single NWK layer frame.

2.2.3.2 Application Support Sub-Layer Management Entity (APSME)

The APSME shall provide a management service to allow an application to interact with the stack.

The APSME shall provide the ability to match two devices together based on their services and their needs. This service is called the binding service, and the APSME shall be able to construct and maintain a table to store this information.

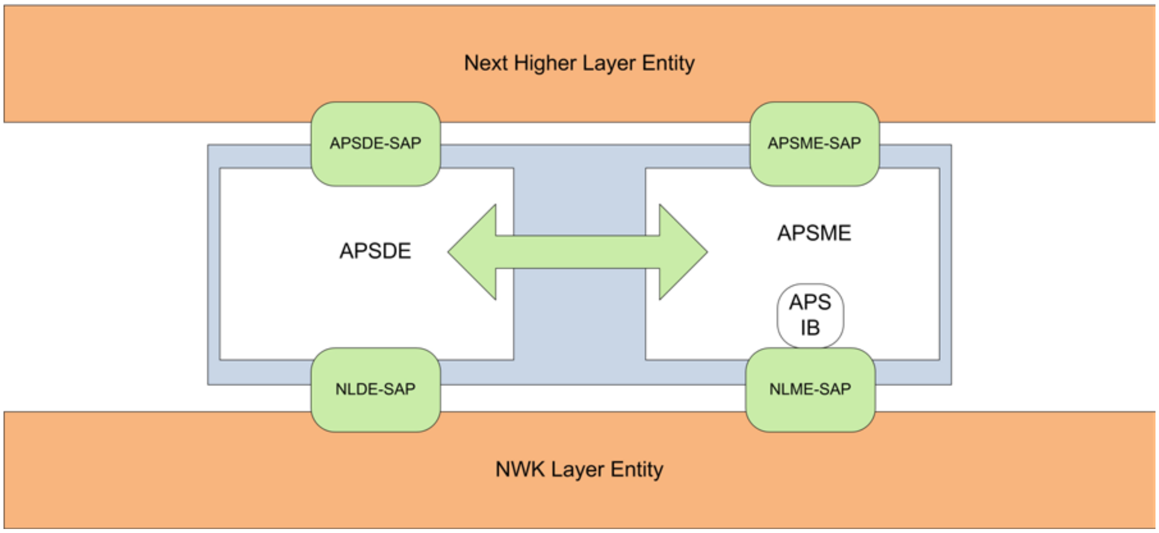
In addition, the APSME will provide the following services:

- **Binding management:** The ability to match two devices together based on their services and their needs.
- **AIB management:** The ability to get and set attributes in the device's AIB.
- **Security:** The ability to set up authentic relationships with other devices through the use of secure keys.
- **Group management:** The ability to declare a single address shared by multiple devices, to add devices to the group, and to remove devices from the group.

2.2.4 Service Specification

The APS sub-layer provides an interface between a next higher layer entity (NHLE) and the NWK layer. The APS sub-layer conceptually includes a management entity called the APS sub-layer management entity (APSME). This entity provides the service interfaces through which sub-layer management functions may be invoked. The APSME is also responsible for maintaining a database of managed objects pertaining to the APS sub-layer. This database is referred to as the APS sub-layer information base (AIB). Figure 2.1 depicts the components and interfaces of the APS sub-layer.

Figure 2.1 The APS Sub-Layer Reference Model



The APS sub-layer provides two services, accessed through two service access points (SAPs). These are the APS data service, accessed through the APS sub-layer data entity SAP (APSDE-SAP), and the APS management service, accessed through the APS sub-layer management entity SAP (APSME-SAP). These two services provide the interface between the NHLE and the NWK layer, via the NLDE-SAP and, to a limited extent, NLME-SAP interfaces (see section 3.1). The NLME-SAP interface between the NWK layer and the APS sub-layer supports only the NLME-GET and NLME-SET primitives; all other NLME-SAP primitives are available only via the ZDO (see section 2.5). In addition to these external interfaces, there is also an implicit interface between the APSME and the APSDE that allows the APSME to use the APS data service.

2.2.4.1 APS Data Service

The APS sub-layer data entity SAP (APSDE-SAP) supports the transport of application protocol data units between peer application entities. Table 2.1 lists the primitives supported by the APSDE-SAP. Each of these primitives will be discussed in the following sections.

Table 2.1 APSDE-SAP Primitives

APSDE-SAP Primitive	Request	Confirm	Indication
APSDE-DATA	2.2.4.1.1	2.2.4.1.2	2.2.4.1.3

2.2.4.1.1 APSDE-DATA.request

This primitive requests the transfer of a NHLE PDU (ASDU) from the local NHLE to one or more peer NHLE entities.

2.2.4.1.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
APSDE-DATA.request          {
                            DstAddrMode,
                            DstAddress,
                            DstEndpoint,
                            ProfileId,
                            ClusterId,
                            SrcEndpoint,
                            ASDULength,
                            ASDU,
                            TxOptions,
                            UseAlias,
                            AliasSrcAddr,
                            AliasSeqNumber,
                            RadiusCounter
                            }
```

Table 2.2 specifies the parameters for the APSDE-DATA.request primitive. Support of the parameters UseAlias, AliasSrcAddr, and AliasSeqNumb in the APSDE-DATA.request primitive is required if Green Power feature is supported by the implementation.

Table 2.2 APSDE-DATA.request Parameters

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x00 – 0xff	The addressing mode for the destination address used in this primitive and of the APDU to be transferred. This parameter can take one of the non-reserved values from the following list: 0x00 = DstAddress and DstEndpoint not present 0x01 = 16-bit group address for DstAddress; DstEndpoint not present 0x02 = 16-bit address for DstAddress and DstEndpoint present 0x03 = 64-bit extended address for DstAddress and DstEnd-point present 0x04 – 0xff = reserved
DstAddress	Address	As specified by the DstAddrMode parameter	The individual device address or group address of the entity to which the ASDU is being transferred.

Name	Type	Valid Range	Description
DstEndpoint	Integer	0x00 – 0xff	This parameter shall be present if, and only if, the DstAddr-Mode parameter has a value of 0x02 or 0x03 and, if present, shall be either the number of the individual endpoint of the entity to which the ASDU is being transferred or the broadcast endpoint (0xff).
ProfileId	Integer	0x0000 – 0xffff	The identifier of the profile for which this frame is intended.
ClusterId	Integer	0x0000 – 0xffff	The identifier of the object for which this frame is intended.
SrcEndpoint	Integer	0x00 – 0xfe	The individual endpoint of the entity from which the ASDU is being transferred.
ASDULength	Integer	0x00 – 256 * (NsduLength - apscMinHeader Overhead)	The number of octets comprising the ASDU to be transferred. The maximum length of an individual APS frame payload is given as NsduLength - *apscMinHeaderOverhead*. Assuming fragmentation is used, there can be 256 such blocks comprising a single maximum sized ASDU.
ASDU	Set of octets	-	The set of octets comprising the ASDU to be transferred.
TxOptions	Bitmap	0000 0000 – 0001 1111	The transmission options for the ASDU to be transferred. These are a bitwise OR of one or more of the following: 0x01 = Security enabled transmission 0x02 = Use NWK key 0x04 = Acknowledged transmission 0x08 = Fragmentation permitted 0x10 = Include extended nonce in APS security frame.
UseAlias	Boolean	TRUE or FALSE	The next higher layer may use the UseAlias parameter to request alias usage by NWK layer for the current frame. If the *UseAlias* parameter has a value of FALSE, meaning no alias usage, then the parameters *AliasSrcAddr* and *AliasSeqNumb* will be ignored. Otherwise, a value of TRUE denotes that the values supplied in *AliasSrcAddr* and *AliasSeqNumb* are to be used.
AliasSrcAddr	16-bit address	Any valid device address except a broadcast address	The source address to be used for this NSDU. If the *UseAlias* parameter has a value of FALSE, the AliasSrcAddr parameter is ignored.

Name	Type	Valid Range	Description
AliasSeqNumb	integer	0x00-0xff	The sequence number to be used for this NSDU. If the *UseAlias* parameter has a value of FALSE, the *AliasSeqNumb* parameter is ignored.
Radius	Unsigned integer	0x00-0xff	The distance, in hops, that a transmitted frame will be allowed to travel through the network.

2.2.4.1.1.2 When Generated

This primitive is generated by a local NHLE whenever a data PDU (ASDU) is to be transferred to one or more peer NHLEs.

2.2.4.1.1.3 Effect on Receipt

On receipt of this primitive, the APS sub-layer entity begins the transmission of the supplied ASDU.

If the DstAddrMode parameter is set to 0x00 and this primitive was received by the APSDE of a device supporting a binding table, a search is made in the binding table with the endpoint and cluster identifiers specified in the SrcEndpoint and ClusterId parameters, respectively, for associated binding table entries. If no binding table entries are found, the APSDE issues the APSDE-DATA.confirm primitive with a status of NO_BOUND_DEVICE. If one or more binding table entries are found, then the APSDE examines the destination address information in each binding table entry. If this indicates a device itself, then the APSDE shall issue an APSDE-DATA.indication primitive to the next higher layer with the DstEndpoint parameter set to the destination endpoint identifier in the binding table entry. If UseAlias parameter has the value of TRUE, the supplied value of the AliasSrcAddr shall be used for the SrcAddress parameter of the APSDE-DATA.indication primitive. Otherwise if the binding table entries do not indicate the device itself,, the APSDE constructs the APDU with the endpoint information from the binding table entry, if present, and uses the destination address information from the binding table entry when transmitting the frame via the NWK layer. If more than one binding table entry is present, then the APSDE processes each binding table entry as described above; until no more binding table entries remain. If this primitive was received by the APSDE of a device that does not support a binding table, the APSDE issues the APSDE-DATA.confirm primitive with a status of NOT_SUPPORTED.

If the DstAddrMode parameter is set to 0x03, the DstAddress parameter contains an extended 64-bit IEEE address and must first be mapped to a corresponding 16-bit NWK address by using the nwkAddressMap attribute of the NIB (see Table 3.43). If a corresponding 16-bit NWK address could not be found, the APSDE issues the APSDE-DATA.confirm primitive with a status of NO_SHORT_ADDRESS. If a corresponding 16-bit NWK address is found, it will be used in the invocation of the NLDE-DATA.request primitive and the value of the DstEndpoint parameter will be placed in the resulting APDU. The delivery mode sub-field of the frame control field of the APS header shall have a value of 0x00 in this case.

If the DstAddrMode parameter has a value of 0x01, indicating group addressing, the DstAddress parameter will be interpreted as a 16-bit group address. This address will be placed in the group address field of the APS header, the DstEndpoint parameter will be ignored, and the destination endpoint field will be omitted from the APS header. The delivery mode sub-field of the frame control field of the APS header shall have a value of 0x03 in this case.

If the DstAddrMode parameter is set to 0x02, the DstAddress parameter contains a 16-bit NWK address, and the DstEndpoint parameter is supplied. The next higher layer should only employ DstAddrMode of 0x02 in cases where the destination NWK address is employed for immediate application responses and the NWK address is not retained for later data transmission requests.

The application may limit the number of hops a transmitted frame is allowed to travel through the network by setting the RadiusCounter parameter of the NLDE-DATA.request primitive to a non-zero value.

If the DstAddrMode parameter has a value of 0x01, indicating group addressing, or the DstAddrMode parameter has a value of 0x00 and the corresponding binding table entry contains a group address, then the APSME will check the value of the nwkUseMulticast attribute of the NIB (see Table 3.44). If this attribute has a value of FALSE, then the delivery mode sub-field of the frame control field of the resulting APDU will be set to 0b11, the 16-bit address of the destination group will be placed in the group address field of the APS header of the outgoing frame, and the NSDU frame will be transmitted as a broadcast. A value of 0xfffd, that is, the broadcast to all devices for which macRxOnWhenIdle = TRUE, will be supplied for the DstAddr parameter of the NLDE-DATA.request that is used to transmit the frame. If the *nwkUseMulticast* attribute has a value of TRUE, then the outgoing frame will be transmitted using NWK layer multicast, with the delivery mode sub-field of the frame control field of the APDU set to 0b10, the destination endpoint field set to 0xff, and the group address not placed in the APS header.

The parameters UseAlias, AliasSrcAddr and AliasSeqNumb shall be used in the invocation of the NLDE-DATA.request primitive.

If the UseAlias parameter has the value of TRUE, and the Acknowledged transmission field of the TxOptions parameter is set to 0b1, then the APSDE issues the APSDE-DATA.confirm primitive with a status of NOT_SUPPORTED.

If the TxOptions parameter specifies that secured transmission is required, the APS sub-layer shall use the security service provider (see section 4.2.3) to secure the ASDU. The security processing shall always be performed using device's own extended 64-bit IEEE address and the OutgoingFrameCounter attribute as stored in apsDeviceKeyPairSet attribute of the AIB for the entity indicated by the DstAddress parameter, and those values shall be put into the auxiliary APS header of the frame, even if UseAlias parameter has a value of TRUE. If the security processing fails, the APSDE shall issue the APSDE-DATA.confirm primitive with a status of SECURITY_FAIL.

The APSDE transmits the constructed frame by issuing the NLDE-DATA.request primitive to the NWK layer. When the APSDE has completed all operations related to this transmission request, including transmitting frames as required, any retransmissions, and the receipt or timeout of any acknowledgements, then the APSDE shall issue the APSDE-DATA.confirm primitive (see section 2.2.4.1.2). If one or more NLDE-DATA.confirm primitives failed, then the Status parameter shall be set to that received from the NWK layer. Otherwise, if one or more APS acknowledgements were not correctly received, then the Status parameter shall be set to NO_ACK. If the ASDU was successfully transferred to all intended targets, then the Status parameter shall be set to SUCCESS.

If NWK layer multicast is being used, the NonmemberRadius parameter of the NLDE-DATA.request primitive shall be set to *apsNonmemberRadius*.

The APSDE will ensure that route discovery is always enabled at the network layer by setting the DiscoverRoute parameter of the NLDE-DATA.request primitive to 0x01, each time it is issued.

If the ASDU to be transmitted is larger than will fit in a single frame and fragmentation is not possible, then the ASDU is not transmitted and the APSDE shall issue the APSDE-DATA.confirm primitive with a status of ASDU_TOO_LONG. Fragmentation is not possible if either an acknowledged transmission is not requested, or if the fragmentation permitted flag in the TxOptions field is set to 0, or if the ASDU is too large to be handled by the APSDE.

If the ASDU to be transmitted is larger than will fit in a single frame, an acknowledged transmission is requested, and the fragmentation permitted flag of the TxOptions field is set to 1, and the ASDU is not too large to be handled by the APSDE, then the ASDU shall be fragmented across multiple APDUs, as described in section 2.2.8.4.5. Transmission and security processing where requested, shall be carried out for each individual APDU independently. Note that fragmentation shall not be used unless relevant higher-layer documentation and/or interactions explicitly indicate that fragmentation is permitted for the frame being sent, and that the other end is able to receive the fragmented transmission, both in terms of number of blocks and total transmission size.

2.2.4.1.2 APSDE-DATA.confirm

The primitive reports the results of a request to transfer a data PDU (ASDU) from a local NHLE to one or more peer NHLEs.

2.2.4.1.2.1 Semantics of the Service Primitive

This semantics of this primitive are as follows:

```
APSDE-DATA.confirm        {
                          DstAddrMode,
                          DstAddress,
                          DstEndpoint,
                          SrcEndpoint,
                          Status,
                          TxTime
                          }
```

Table 2.3 specifies the parameters for the APSDE-DATA.confirm primitive.

Table 2.3 APSDE-DATA.confirm Parameters

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x00 – 0xff	The addressing mode for the destination address used in this primitive and of the APDU to be transferred. This parameter can take one of the non-reserved values from the following list: 0x00 = DstAddress and DstEndpoint not present 0x01 = 16-bit group address for DstAddress; DstEndpoint not present 0x02 = 16-bit address for DstAddress and DstEndpoint present 0x03= 64-bit extended address for DstAddress and DstEndpoint present 0x04 – 0xff = reserved
DstAddress	Address	As specified by the DstAddr-Mode parameter	The individual device address or group address of the entity to which the ASDU is being transferred.
DstEndpoint	Integer	0x00 – 0xff	This parameter shall be present if, and only if, the DstAddrMode parameter has a value of 0x02 or 0x03 and, if present, shall be the number of the individual endpoint of the entity to which the ASDU is being transferred.
SrcEndpoint	Integer	0x00 – 0xfe	The individual endpoint of the entity from which the ASDU is being transferred.

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS, NO_SHORT_ADDRESS, NO_BOUND_DEVICE, SECURITY_FAIL, NO_ACK, ASDU_TOO_LONG or any status values returned from the NLDE-DATA.confirm primitive	The status of the corresponding request.
TxTime	Integer	Implementation specific	A time indication for the transmitted packet based on the local clock, as provided by the NWK layer.

2.2.4.1.2.2 When Generated

This primitive is generated by the local APS sub-layer entity in response to an APSDE-DATA.request primitive. This primitive returns a status of either SUCCESS, indicating that the request to transmit was successful, or an error code of NO_SHORT_ADDRESS, NO_BOUND_DEVICE, SECURITY_FAIL, ASDU_TOO_LONG, or any status values returned from the NLDE-DATA.confirm primitive. The reasons for these status values are fully described in section 2.2.4.1.1.3.

2.2.4.1.2.3 Effect on Receipt

On receipt of this primitive, the next higher layer of the initiating device is notified of the result of its request to transmit. If the transmission attempt was successful, the Status parameter will be set to SUCCESS. Otherwise, the Status parameter will indicate the error.

2.2.4.1.3 APSDE-DATA.indication

This primitive indicates the transfer of a data PDU (ASDU) from the APS sub-layer to the local application entity.

2.2.4.1.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
APSDE-DATA.indication        {
                             DstAddrMode,
                             DstAddress,
                             DstEndpoint,
                             SrcAddrMode,
                             SrcAddress,
                             SrcEndpoint,
                             ProfileId,
                             ClusterId,
                             asduLength,
                             asdu,
                             Status,
                             SecurityStatus,
                             LinkQuality,
                             RxTime
                             }
```

Table 2.4 specifies the parameters for the APSDE-DATA.indication primitive.

Table 2.4 APSDE-DATA.indication Parameters

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x00 - 0xff	The addressing mode for the destination address used in this primitive and of the APDU that has been received. This parameter can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddress; DstEndpoint not present 0x02 = 16-bit address for DstAddress and DstEndpoint present 0x03 = 64-bit extended address for DstAddress and DstEndpoint present. 0x04 = 64-bit extended address for DstAddress, but DstEndpoint NOT present. 0x05 – 0xff = reserved
DstAddress	Address	As specified by the DstAddrMode parameter	The individual device address or group address to which the ASDU is directed.

Name	Type	Valid Range	Description
DstEndpoint	Integer	0x00 – 0xfe	The target endpoint on the local entity to which the ASDU is directed.
SrcAddrMode	Integer	0x00 – 0xff	The addressing mode for the source address used in this primitive and of the APDU that has been received. This parameter can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = reserved 0x02 = 16-bit short address for SrcAddress and SrcEndpoint present 0x03 = 64-bit extended address for SrcAddress and SrcEndpoint present 0x04 = 64-bit extended address for SrcAddress, but SrcEndpoint NOT present. 0x05 – 0xff = reserved
SrcAddress	Address	As specified by the SrcAddrMode parameter	The individual device address of the entity from which the ASDU has been received.
SrcEndpoint	Integer	0x00 – 0xfe	The number of the individual endpoint of the entity from which the ASDU has been received.
ProfileId	Integer	0x0000 - 0xffff	The identifier of the profile from which this frame originated.
ClusterId	Integer	0x0000-0xffff	The identifier of the received object.
asduLength	Integer		The number of octets comprising the ASDU being indicated by the APSDE.
asdu	Set of octets	-	The set of octets comprising the ASDU being indicated by the APSDE.
Status	Enumeration	SUCCESS, DEFRAG_UNSUPPORTED, DEFRAG_DEFERRED or any status returned from the security processing of the frame	The status of the incoming frame processing.
SecurityStatus	Enumeration	UNSECURED, SECURED_NWK_KEY,	UNSECURED if the ASDU was received without any security.

Name	Type	Valid Range	Description
		SECURED_LINK_KEY	SECURED_NWK_KEY if the received ASDU was secured with the NWK key. SECURED_LINK_KEY if the ASDU was secured with a link key.
LinkQuality	Integer	0x00 - 0xff	The link quality indication delivered by the NLDE.
RxTime	Integer	Implementation specific	A time indication for the received packet based on the local clock, as provided by the NWK layer.

2.2.4.1.3.2 When Generated

This primitive is generated by the APS sub-layer and issued to the next higher layer on receipt of an appropriately addressed data frame from the local NWK layer entity or following receipt of an APSDE-DATA. request in which the DstAddrMode parameter was set to 0x00 and the binding table entry has directed the frame to the device itself. If the frame control field of the ASDU header indicates that the frame is secured, security processing shall be done as specified in section 4.2.3.

This primitive is generated by the APS sub-layer entity and issued to the next higher layer entity on receipt of an appropriately addressed data frame from the local network layer entity, via the NLDE-DATA.indication primitive.

If the frame control field of the APDU header indicates that the frame is secured, then security processing must be undertaken as specified in section 4.2.3. If the security processing fails, the APSDE sets the Status parameter to the security error code returned from the security processing.

If the frame is not secured or the security processing was successful, the APSDE must check for the frame being fragmented. If the extended header is included in the APDU header and the fragmentation sub-field of the extended frame control field indicates that the frame is fragmented but this device does not support fragmentation, the APSDE sets the Status parameter to DEFRAG_UNSUPPORTED. If the extended header is included in the APDU header, the fragmentation sub-field of the extended frame control field indicates that the frame is fragmented and the device supports fragmentation, but is not currently able to defragment the frame, the APSDE sets the Status parameter to DEFRAG_DEFERRED.

Under all other circumstances, the APSDE sets the Status parameter to SUCCESS.

If the Status parameter is not set to SUCCESS, the APSDE sets the ASDULength parameter to 0 and the ASDU parameter to the null set of bytes.

The APS sub-layer entity shall attempt to map the source address from the received frame to its corresponding extended 64-bit IEEE address by using the nwkAddressMap attribute of the NIB (see Table 3.43). If a corresponding 64-bit IEEE address was found, the APSDE issues this primitive with the SrcAddrMode parameter set to 0x03 and the SrcAddress parameter set to the corresponding 64-bit IEEE address. If a corresponding 64-bit IEEE address was not found, the APSDE issues this primitive with the SrcAddrMode parameter set to 0x02, and the SrcAddress parameter set to the 16-bit source address as contained in the received frame.

2.2.4.1.3.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the arrival of data at the device.

2.2.4.2 APS Management Service

The APS management entity SAP (APSME-SAP) supports the transport of management commands between the next higher layer and the APSME. Table 2.5 summarizes the primitives supported by the APSME through the APSME-SAP interface. See the following sections for more details on the individual primitives.

Table 2.5 Summary of the Primitives Accessed Through the APSME-SAP

Name	Request	Indication	Response	Confirm
APSME-BIND	2.2.4.3.1			2.2.4.3.2
APSME-UNBIND	2.2.4.3.3			2.2.4.3.4
APSME-GET	2.2.4.4.1			2.2.4.4.2
APSME-SET	2.2.4.4.3			2.2.4.4.4
APSME-ADD-GROUP	2.2.4.5.1			2.2.4.5.2
APSME-REMOVE-GROUP	2.2.4.5.3			2.2.4.5.4
APSME-REMOVE-ALL-GROUPS	2.2.4.5.5			2.2.4.5.6

2.2.4.3 Binding Primitives

This set of primitives defines how the next higher layer of a device can add (commit) a binding record to its local binding table or remove a binding record from its local binding table.

Only a device supporting a binding table or a binding table cache may process these primitives. If any other device receives these primitives from their next higher layer, the primitives should be rejected.

2.2.4.3.1 APSME-BIND.request

This primitive allows the next higher layer to request to bind two devices together, or to bind a device to a group, by creating an entry in its local binding table, if supported.

2.2.4.3.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
APSME-BIND.request          {
                            SrcAddr,
                            SrcEndpoint,
                            ClusterId,
                            DstAddrMode,
                            DstAddr,
                            DstEndpoint
                            }
```

Table 2.6 specifies the parameters for the APSME-BIND.request primitive.

Table 2.6 APSME-BIND.request Parameters

Name	Type	Valid Range	Description
SrcAddr	IEEE address	A valid 64-bit IEEE address	The source IEEE address for the binding entry.
SrcEndpoint	Integer	0x01 – 0xfe	The source endpoint for the binding entry.
ClusterId	Integer	0x0000 – 0xffff	The identifier of the cluster on the source device that is to be bound to the destination device.
DstAddrMode	Integer	0x00 – 0xff	The addressing mode for the destination address used in this primitive. This parameter can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddr and DstEndpoint not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddr and DstEndpoint present 0x04 – 0xff = reserved
DstAddr	Address	As specified by the DstAddrMode parameter	The destination address for the binding entry.
DstEndpoint	Integer	0x01 – 0xff	This parameter will be present only if the DstAddrMode parameter has a value of 0x03 and, if present, will be the destination endpoint for the binding entry.

2.2.4.3.1.2 When Generated

This primitive is generated by the next higher layer and issued to the APS sub-layer in order to instigate a binding operation on a device that supports a binding table.

2.2.4.3.1.3 Effect on Receipt

On receipt of this primitive by a device that is not currently joined to a network, or by a device that does not support a binding table, or if any of the parameters has a value which is out of range, the APSME issues the APSME-BIND.confirm primitive with the Status parameter set to ILLEGAL_REQUEST.

If the APS sub-layer on a device that supports a binding table receives this primitive from the NHLE, the APSME attempts to create the specified entry directly in its binding table. If the entry could be created, the APSME issues the APSME-BIND.confirm primitive with the Status parameter set to SUCCESS. If the entry could not be created due to a lack of capacity in the binding table, the APSME issues the APSME-BIND.confirm primitive with the Status parameter set to TABLE_FULL.

2.2.4.3.2 APSME-BIND.confirm

This primitive allows the next higher layer to be notified of the results of its request to bind two devices together, or to bind a device to a group.

2.2.4.3.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
APSME-BIND.confirm          {
                            Status,
                            SrcAddr,
                            SrcEndpoint,
                            ClusterId,
                            DstAddrMode,
                            DstAddr,
                            DstEndpoint
                            }
```

Table 2.7 specifies the parameters for the APSME-BIND.confirm primitive.

Table 2.7 APSME-BIND.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS, ILLEGAL_REQUEST, TABLE_FULL, NOT_SUPPORTED	The results of the binding request.
SrcAddr	IEEE address	A valid 64-bit IEEE address	The source IEEE address for the binding entry.
SrcEndpoint	Integer	0x01 – 0xfe	The source endpoint for the binding entry.
ClusterId	Integer	0x0000 – 0xffff	The identifier of the cluster on the source device that is to be bound to the destination device.
DstAddrMode	Integer	0x00 – 0xff	The addressing mode for the destination address used in this primitive. This parameter can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddr and DstEndpoint not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddr and DstEndpoint present 0x04 – 0xff = reserved
DstAddr	Address	As specified by the DstAddrMode parameter	The destination address for the binding entry.

Name	Type	Valid Range	Description
DstEndpoint	Integer	0x01 – 0xff	This parameter will be present only if the DstAddrMode parameter has a value of 0x03 and, if present, will be the destination endpoint for the binding entry.

2.2.4.3.2.2 When Generated

This primitive is generated by the APSME and issued to its NHLE in response to an APSME-BIND.request primitive. If the request was successful, the Status parameter will indicate a successful bind request. Otherwise, the Status parameter indicates an error code of NOT_SUPPORTED, ILLEGAL_REQUEST or TABLE_FULL.

2.2.4.3.2.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its bind request. If the bind request was successful, the Status parameter is set to SUCCESS. Otherwise, the Status parameter indicates the error.

2.2.4.3.3 APSME-UNBIND.request

This primitive allows the next higher layer to request to unbind two devices, or to unbind a device from a group, by removing an entry in its local binding table, if supported.

2.2.4.3.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
APSME-UNBIND.request        {
                            SrcAddr,
                            SrcEndpoint,
                            ClusterId,
                            DstAddrMode,
                            DstAddr,
                            DstEndpoint
                            }
```

Table 2.8 specifies the parameters for the APSME-UNBIND.request primitive.

Table 2.8 APSME-UNBIND.request Parameters

Name	Type	Valid Range	Description
SrcAddr	IEEE address	A valid 64-bit IEEE address	The source IEEE address for the binding entry.
SrcEndpoint	Integer	0x01 – 0xfe	The source endpoint for the binding entry.
ClusterId	Integer	0x0000 – 0xffff	The identifier of the cluster on the source device that is bound to the destination device.

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x00 – 0xff	The addressing mode for the destination address used in this primitive. This parameter can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddr and DstEndpoint not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddr and DstEndpoint present 0x04 – 0xff = reserved
DstAddr	Address	As specified by the DstAddrMode parameter	The destination address for the binding entry.
DstEndpoint	Integer	0x01 – 0xff	This parameter will be present only if the DstAddrMode parameter has a value of 0x03 and, if present, will be the destination endpoint for the binding entry.

2.2.4.3.3.2 When Generated

This primitive is generated by the next higher layer and issued to the APS sub-layer in order to instigate an unbind operation on a device that supports a binding table.

2.2.4.3.3.3 Effect on Receipt

On receipt of this primitive by a device that is not currently joined to a network, or by a device that does not support a binding table, or if any of the parameters has a value which is out of range, the APSME issues the APSME-UNBIND.confirm primitive with the Status parameter set to ILLEGAL_REQUEST.

If the APS on a device that supports a binding table receives this primitive from the NHLE, the APSME searches for the specified entry in its binding table. If the entry exists, the APSME removes the entry and issues the APSME-UNBIND.confirm (see section 2.2.4.3.4) primitive with the Status parameter set to SUCCESS. If the entry could not be found, the APSME issues the APSME-UNBIND.confirm primitive with the Status parameter set to INVALID_BINDING.

2.2.4.3.4 APSME-UNBIND.confirm

This primitive allows the next higher layer to be notified of the results of its request to unbind two devices, or to unbind a device from a group.

2.2.4.3.4.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
APSME-UNBIND.confirm        {
                            Status,
                            SrcAddr,
                            SrcEndpoint,
                            ClusterId,
                            DstAddrMode,
                            DstAddr,
                            DstEndpoint
                            }
```

Table 2.9 specifies the parameters for the APSME-UNBIND.confirm primitive.

Table 2.9 APSME-UNBIND.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS, ILLEGAL_REQUEST, INVALID_BINDING	The results of the unbind request.
SrcAddr	IEEE address	A valid 64-bit IEEE address	The source IEEE address for the binding entry.
SrcEndpoint	Integer	0x01 – 0xfe	The source endpoint for the binding entry.
ClusterId	Integer	0x0000 – 0xffff	The identifier of the cluster on the source device that is bound to the destination device.
DstAddrMode	Integer	0x00 – 0xff	The addressing mode for the destination address used in this primitive. This parameter can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddr and DstEndpoint not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddr and DstEndpoint present 0x04 – 0xff = reserved
DstAddr	Address	As specified by the DstAddrMode parameter	The destination address for the binding entry.
DstEndpoint	Integer	0x01 – 0xff	The destination endpoint for the binding

Name	Type	Valid Range	Description
			entry.

2.2.4.3.4.2 When Generated

This primitive is generated by the APSME and issued to its NHLE in response to an APSME-UNBIND. request primitive. If the request was successful, the Status parameter will indicate a successful unbind request. Otherwise, the Status parameter indicates an error code of ILLEGAL_REQUEST, or INVALID_BINDING.

2.2.4.3.4.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its unbind request. If the unbind request was successful, the Status parameter is set to SUCCESS. Otherwise, the Status parameter indicates the error.

2.2.4.4 Information Base Maintenance

This set of primitives defines how the next higher layer of a device can read and write attributes in the AIB.

2.2.4.4.1 APSME-GET.request

This primitive allows the next higher layer to read the value of an attribute from the AIB.

2.2.4.4.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
APSME-GET.request                {
                                 AIBAttribute
                                 }
```

Table 2.10 specifies the parameters for this primitive.

Table 2.10 APSME-GET.request Parameters

Name	Type	Valid Range	Description
AIBAttribute	Integer	See Table 2.24	The identifier of the AIB attribute to read.

2.2.4.4.1.2 When Generated

This primitive is generated by the next higher layer and issued to its APSME in order to read an attribute from the AIB.

2.2.4.4.1.3 Effect on Receipt

On receipt of this primitive, the APSME attempts to retrieve the requested AIB attribute from its database. If the identifier of the AIB attribute is not found in the database, the APSME issues the APSME-GET.confirm primitive with a status of UNSUPPORTED_ATTRIBUTE.

If the requested AIB attribute is successfully retrieved, the APSME issues the APSME-GET.confirm primitive with a status of SUCCESS such that it contains the AIB attribute identifier and value.

2.2.4.4.2 APSME-GET.confirm

This primitive reports the results of an attempt to read the value of an attribute from the AIB.

2.2.4.4.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
APSME-GET.confirm          {
                           Status,
                           AIBAttribute,
                           AIBAttributeLength,
                           AIBAttributeValue
                           }
```

Table 2.11 specifies the parameters for this primitive.

Table 2.11 APSME-GET.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS or UNSUPPORTED_ ATTRIBUTE	The results of the request to read an AIB attribute value.
AIBAttribute	Integer	See Table 2.24	The identifier of the AIB attribute that was read.
AIBAttributeLength	Integer	0x0001 - 0xffff	The length, in octets, of the attribute value being returned.
AIBAttributeValue	Various	Attribute specific (see Table 2.24)	The value of the AIB attribute that was read.

2.2.4.4.2.2 When Generated

This primitive is generated by the APSME and issued to its next higher layer in response to an APSME-GET.request primitive. This primitive returns a status of SUCCESS, indicating that the request to read an AIB attribute was successful, or an error code of UNSUPPORTED_ATTRIBUTE. The reasons for these status values are fully described in section 2.2.4.4.1.3.

2.2.4.4.2.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to read an AIB attribute. If the request to read an AIB attribute was successful, the Status parameter will be set to SUCCESS. Otherwise, the Status parameter indicates the error.

2.2.4.4.3 APSME-SET.request

This primitive allows the next higher layer to write the value of an attribute into the AIB.

2.2.4.4.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

APSME-SET.request	{
	AIBAttribute,
	AIBAttributeLength,
	AIBAttributeValue
	}

Table 2.12 specifies the parameters for this primitive.

Table 2.12 APSME-SET.request Parameters

Name	Type	Valid Range	Description
AIBAttribute	Integer	See Table 2.24	The identifier of the AIB attribute to be written.
AIBAttributeLength	Integer	0x0000 - 0xffff	The length, in octets, of the attribute value being set.
AIBAttributeValue	Various	Attribute specific (see Table 2.24).	The value of the AIB attribute that should be written.

2.2.4.4.3.2 When Generated

This primitive is to be generated by the next higher layer and issued to its APSME in order to write the value of an attribute in the AIB.

2.2.4.4.3.3 Effect on Receipt

On receipt of this primitive, the APSME attempts to write the given value to the indicated AIB attribute in its database. If the AIBAttribute parameter specifies an attribute that is not found in the database, the APSME issues the APSME-SET.confirm primitive with a status of UNSUPPORTED_ATTRIBUTE. If the AIBAttributeValue parameter specifies a value that is outside the valid range for the given attribute, the APSME issues the APSME-SET.confirm primitive with a status of INVALID_PARAMETER.

If the requested AIB attribute is successfully written, the APSME issues the APSME-SET.confirm primitive with a status of SUCCESS.

2.2.4.4.4 APSME-SET.confirm

This primitive reports the results of an attempt to write a value to an AIB attribute.

2.2.4.4.4.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

APSME-SET.confirm	{
	Status,
	AIBAttribute
	}

Table 2.13 specifies the parameters for this primitive.

Table 2.13 APSME-SET.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS, INVALID_PARAMETER or UNSUPPORTED_ATTRIBUTE	The result of the request to write the AIB Attribute.
AIBAttribute	Integer	See Table 2.24	The identifier of the AIB attribute that was written.

2.2.4.4.4.2 When Generated

This primitive is generated by the APSME and issued to its next higher layer in response to an APSME-SET.request primitive. This primitive returns a status of either SUCCESS, indicating that the requested value was written to the indicated AIB attribute, or an error code of INVALID_PARAMETER or UNSUPPORTED_ATTRIBUTE. The reasons for these status values are completely described in section 2.2.4.4.3.3.

2.2.4.4.4.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to write the value of a AIB attribute. If the requested value was written to the indicated AIB attribute, the Status parameter will be set to SUCCESS. Otherwise, the Status parameter indicates the error.

2.2.4.5 Group Management

This set of primitives allows the next higher layer to manage group membership for endpoints on the current device by adding and removing entries in the group table.

2.2.4.5.1 APSME-ADD-GROUP.request

This primitive allows the next higher layer to request that group membership for a particular group be added for a particular endpoint.

2.2.4.5.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
APSME-ADD-GROUP.request          {
                                 GroupAddress,
                                 Endpoint
                                 }
```

Table 2.14 specifies the parameters for this primitive.

Table 2.14 APSME-ADD-GROUP.request Parameters

Name	Type	Valid Range	Description
GroupAddress	16-bit group address	0x0000 - 0xffff	The 16-bit address of the group being added.
Endpoint	Integer	0x01 - 0xfe	The endpoint to which the given group is being added.

2.2.4.5.1.2 When Generated

This primitive is generated by the next higher layer when it wants to add membership in a particular group to an endpoint, so that frames addressed to the group will be delivered to that endpoint in the future.

2.2.4.5.1.3 Effect on Receipt

If, on receipt of this primitive, the GroupAddress parameter is found to be outside the valid range, then the APSME will issue the APSME-ADD-GROUP.confirm primitive to the next higher layer with a status value of INVALID_PARAMETER. Similarly, if the Endpoint parameter has a value which is out of range or else enumerates an endpoint that is not implemented on the current device, the APSME will issue the APSME-ADD-GROUP.confirm primitive with a Status of INVALID_PARAMETER.

After checking the parameters as described above, the APSME will check the group table to see if an entry already exists containing the values given by the GroupAddress and Endpoint parameters. If such an entry already exists in the table then the APSME will issue the APSME-ADD-GROUP.confirm primitive to the next higher layer with a status value of SUCCESS. If there is no such entry and there is space in the table for another entry then the APSME will add a new entry to the group table with the values given by the GroupAddress and Endpoint parameters. After the entry is added to the APS group table, and if the NWK layer is maintaining a group table, then the APSME ensures that the corresponding NWK group table is consistent by issuing the NLME-SET.request primitive, for the *nwkGroupIDTable* attribute, with the list of group addresses contained in the group table of the APS sub-layer. Once both tables are consistent, the APSME issues the APSME-ADD-GROUP.confirm primitive to the next higher layer with a status value of SUCCESS. If no entry for the given GroupAddress and Endpoint is present but there is no room in the group table for another entry, then the APSME will issue the APSME-ADD-GROUP.confirm primitive to the next higher layer with a status value of TABLE_FULL.

2.2.4.5.2 APSME-ADD-GROUP.confirm

This primitive allows the next higher layer to be informed of the results of its request to add a group to an endpoint.

2.2.4.5.2.1 Semantics of the Service Primitive

The semantics of the service primitive are as follows:

```
APSME-ADD-GROUP.confirm          {
                                 Status,
                                 GroupAddress,
                                 Endpoint
                                 }
```


Table 2.15 specifies the parameters for this primitive.

Table 2.15 APSME-ADD-GROUP.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS, INVALID_PARAMETER or TABLE_FULL	The status of the request to add a group.
GroupAddress	16-bit group address	0x0000 - 0xffff	The 16-bit address of the group being added.
Endpoint	Integer	0x01 - 0xfe	The endpoint to which the given group is being added.

2.2.4.5.2.2 When Generated

This primitive is generated by the APSME and issued to the next higher layer in response to an APMSE-ADD-GROUP.request primitive. If the APSME-ADD-GROUP.request was successful, then the Status parameter value will be SUCCESS. If one of the parameters of the APMSE-ADD-GROUP.request primitive had an invalid value, then the status value will be set to INVALID_PARAMETER. If the APMSE attempted to add a group table entry but there was no room in the table for another entry, then the status value will be TABLE_FULL.

2.2.4.5.2.3 Effect on Receipt

On receipt of this primitive, the next higher layer is informed of the status of its request to add a group. The Status parameter values will be as described above.

2.2.4.5.3 APSME-REMOVE-GROUP.request

This primitive allows the next higher layer to request that group membership in a particular group for a particular endpoint be removed.

2.2.4.5.3.1 Semantics of the Service Primitive

The semantics of the service primitive are as follows:

```
APSME-REMOVE-GROUP.request      {
                                GroupAddress,
                                Endpoint
                                }
```


Table 2.16 specifies the parameters for this primitive.

Table 2.16 APSME-REMOVE-GROUP.request Parameters

Name	Type	Valid Range	Description
GroupAddress	16-bit group address	0x0000 - 0xffff	The 16-bit address of the group being removed.
Endpoint	Integer	0x01 - 0xfe	The endpoint to which the given group is being removed.

2.2.4.5.3.2 When Generated

This primitive is generated by the next higher layer when it wants to remove membership in a particular group from an endpoint so that frames addressed to the group will no longer be delivered to that endpoint.

2.2.4.5.3.3 Effect on Receipt

If, on receipt of this primitive, the GroupAddress parameter is found to be outside the valid range, then the APSME will issue the APSME-REMOVE-GROUP.confirm primitive to the next higher layer with a status value of INVALID_PARAMETER. Similarly, if the Endpoint parameter has a value which is out of range or else enumerates an endpoint that is not implemented on the current device, the APSME will issue the APSME-REMOVE-GROUP.confirm primitive with a Status of INVALID_PARAMETER.

After checking the parameters as described above, the APSME will check the group table to see if an entry exists containing the values given by the GroupAddress and Endpoint parameters. If such an entry already exists in the table, then that entry will be removed. If the NWK layer is maintaining a group table, then the APSME ensures that the NWK group table is consistent by issuing the NLME-SET.request primitive, for the *nwkGroupIDTable* attribute, with the list of group addresses contained in the group table of the APS sub-layer. Once both tables are consistent, the APSME issues the APSME-REMOVE-GROUP.confirm primitive to the next higher layer with a status value of SUCCESS. If there is no such entry, the APSME will issue the APSME-REMOVE-GROUP.confirm primitive to the next higher layer with a status value of INVALID_GROUP.

2.2.4.5.4 APSME-REMOVE-GROUP.confirm

This primitive allows the next higher layer to be informed of the results of its request to remove a group from an endpoint.

2.2.4.5.4.1 Semantics of the Service Primitive

The semantics of the service primitive are as follows:

```
APSME-REMOVE-GROUP.confirm          {
                                    Status,
                                    GroupAddress,
                                    Endpoint
                                            }
```


Table 2.17 specifies the parameters for this primitive.

Table 2.17 APSME-REMOVE-GROUP.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS, INVALID_ GROUP or INVALID_ PARAMETER	The status of the request to remove a group.
GroupAddress	16-bit group address	0x0000 - 0xffff	The 16-bit address of the group being removed.
Endpoint	Integer	0x01 - 0xfe	The endpoint which is to be removed from the group.

2.2.4.5.4.2 When Generated

This primitive is generated by the APSME and issued to the next higher layer in response to an APMSE-REMOVE-GROUP.request primitive. If the APSME-REMOVE-GROUP.request was successful, the Status parameter value will be SUCCESS. If the APSME-REMOVE-GROUP.request was not successful because an entry containing the values given by the GroupAddress and Endpoint parameters did not exist, then the status value will be INVALID_GROUP. If one of the parameters of the APMSE-REMOVE-GROUP.request primitive had an invalid value, then the status value will be INVALID_PARAMETER.

2.2.4.5.4.3 Effect on Receipt

On receipt of this primitive, the next higher layer is informed of the status of its request to remove a group. The Status parameter values will be as described above.

2.2.4.5.5 APSME-REMOVE-ALL-GROUPS.request

This primitive is generated by the next higher layer when it wants to remove membership in all groups from an endpoint, so that no group-addressed frames will be delivered to that endpoint.

2.2.4.5.5.1 Semantics of the Service Primitive

The semantics of the service primitive are as follows:

```
APSME-REMOVE-ALL-GROUPS.request {
                                Endpoint
                                }
```

Table 2.18 specifies the parameters for this primitive.

Table 2.18 APSME-REMOVE-ALL-GROUPS.request Parameters

Name	Type	Valid Range	Description
Endpoint	Integer	0x01 - 0xfe	The endpoint to which the given group is being removed.

2.2.4.5.5.2 When Generated

This primitive is generated by the next higher layer when it wants to remove membership in all groups from an endpoint so that no group addressed frames will be delivered to that endpoint.

2.2.4.5.5.3 Effect on Receipt

If, on receipt of this primitive, the Endpoint parameter has a value which is out of range or else enumerates an endpoint that is not implemented on the current device the APSME will issue the APSME-REMOVE-ALL-GROUPS.confirm primitive with a Status of INVALID_PARAMETER.

After checking the Endpoint parameter as described above, the APSME will remove all entries related to this endpoint from the group table. The APSME ensures that the corresponding NWK group table is consistent by issuing the NLME-SET.request primitive, for the *nwkGroupIDTable* attribute, with the list of group addresses contained in the group table of the APS sub-layer. Once both tables are consistent, the APSME issues the APSME-REMOVE-ALL-GROUPS.confirm primitive to the next higher layer with a status value of SUCCESS.

2.2.4.5.6 APSME-REMOVE-ALL-GROUPS.confirm

This primitive allows the next higher layer to be informed of the results of its request to remove all groups from an endpoint.

2.2.4.5.6.1 Semantics of the Service Primitive

The semantics of the service primitive are as follows:

```
APSME-REMOVE-ALL-GROUPS.confirm  {
                                  Status,
                                  Endpoint
                                  }
```

Table 2.19 specifies the parameters for this primitive.

Table 2.19 APSME-REMOVE-ALL-GROUPS.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS or INVALID_PARAMETER	The status of the request to remove all groups.
Endpoint	Integer	0x01 - 0xfe	The endpoint which is to be removed from all groups.

2.2.4.5.6.2 When Generated

This primitive is generated by the APSME and issued to the next higher layer in response to an APSME-REMOVE-ALL-GROUPS.request primitive. If the APSME-REMOVE-ALL-GROUPS.request was successful, then the Status parameter value will be SUCCESS. If the Endpoint parameter of the APSME-REMOVE-ALL-GROUPS.request primitive had an invalid value, then the status value will be INVALID_PARAMETER.

2.2.4.5.6.3 Effect on Receipt

On receipt of this primitive, the next higher layer is informed of the status of its request to remove all groups from an endpoint. The Status parameter values will be as described above.

2.2.5 Frame Formats

This section specifies the format of the APS frame (APDU). Each APS frame consists of the following basic components:

- An APS header, which comprises frame control and addressing information.
- An APS payload, of variable length, which contains information specific to the frame type.

The frames in the APS sub-layer are described as a sequence of fields in a specific order. All frame formats in this section are depicted in the order in which they are transmitted by the NWK layer, from left to right, where the leftmost bit is transmitted first in time. Bits within each field are numbered from 0 (leftmost and least significant) to k-1 (rightmost and most significant), where the length of the field is k bits. Fields that are longer than a single octet are sent to the NWK layer in order from the octet containing the lowest-numbered bits to the octet containing the highest-numbered bits.

On transmission, all fields marked as reserved shall be set to zero. On reception, all fields marked as reserved in this version of the specification shall be checked for being equal to zero. If such a reserved field is not equal to zero, no further processing shall be applied to the frame and the frame shall be discarded.

2.2.5.1 General APDU Frame Format

The APS frame format is composed of an APS header and an APS payload. The fields of the APS header appear in a fixed order, however, the addressing fields may not be included in all frames. The general APS frame shall be formatted as illustrated in Figure 2.2.

Figure 2.2 General APS Frame Format

Octets: 1	**0/1**	**0/2**	**0/2**	**0/2**	**0/1**	**1**	**0/ Variable**	**Variable**
Frame control	Destination endpoint	Group address	Cluster identifier	Profile identifier	Source endpoint	APS counter	Extended header	Frame payload
	Addressing fields							
APS header								APS payload

2.2.5.1.1 Frame Control Field

The frame control field is 8-bits in length and contains information defining the frame type, addressing fields, and other control flags. The frame control field shall be formatted as illustrated in Figure 2.3.

Figure 2.3 Format of the Frame Control Field

Bits: 0-1	**2-3**	**4**	**5**	**6**	**7**
Frame type	Delivery mode	Ack. format	Security	Ack. request	Extended header present

2.2.5.1.1.1 Frame Type Sub-Field

The frame type sub-field is two bits in length and shall be set to one of the non-reserved values listed in Table 2.20.

Table 2.20 Values of the Frame Type Sub-Field

Frame Type Value b_1 b_0	**Frame Type Name**
00	Data
01	Command
10	Acknowledgement
11	Inter-PAN APS

2.2.5.1.1.2 Delivery Mode Sub-Field

The delivery mode sub-field is two bits in length and shall be set to one of the non-reserved values from Table 2.21.

Table 2.21 Values of the Delivery Mode Sub-Field

Delivery Mode Value b_1 b_0	**Delivery Mode Name**
00	Normal unicast delivery
01	Reserved
10	Broadcast
11	Group addressing

If the value is 0b00, the frame will be delivered to a given endpoint on the receiving device.

If the value is 0b10, the message is a broadcast. In this case, the message will go to all devices defined for the selected broadcast address in use as defined in section 3.6.5. The destination endpoint field shall be set to a value between 0x01-0xfe (for broadcasts to specific endpoints) or to 0xff (for broadcasts to all active endpoints).

If the value is 0b11, then group addressing is in use and that frame will only be delivered to device endpoints that express group membership in the group identified by the group address field in the APS header. Note that other endpoints on the source device may be members of the group addressed by the outgoing frame. The frame shall be delivered to any member of the group, including other endpoints on the source device that are members of the specified group.

Devices where nwkUseMulticast is set to TRUE, shall never set the delivery mode of an outgoing frame to 0b11. In this case, the delivery mode of the outgoing frame shall be set to 0b10 (broadcast) and the frame shall be sent using an NLDE-DATA.request with the destination address mode set to group addressing.

2.2.5.1.1.3 Ack Format Field

This bit indicates if the destination endpoint, cluster identifier, profile identifier and source endpoint fields shall be present in the acknowledgement frame. This is set to 0 for data frame acknowledgement and 1 for APS command frame acknowledgement.

2.2.5.1.1.4 Security Sub-Field

The Security Services Provider (see Chapter 4) manages the security sub-field.

2.2.5.1.1.5 Acknowledgement Request Sub-Field

The acknowledgement request sub-field is one bit in length and specifies whether the current transmission requires an acknowledgement frame to be sent to the originator on receipt of the frame. If this sub-field is set to 1, the recipient shall construct and send an acknowledgement frame back to the originator after determining that the frame is valid. If this sub-field is set to 0, the recipient shall not send an acknowledgement frame back to the originator.

This sub-field shall be set to 0 for all frames that are broadcast or multicast.

2.2.5.1.1.6 Extended Header Present

The extended header present sub-field is one bit in length and specifies whether the extended header shall be included in the frame. If this sub-field is set to 1, then the extended header shall be included in the frame. Otherwise, it shall not be included in the frame.

2.2.5.1.2 Destination Endpoint Field

The destination endpoint field is 8-bits in length and specifies the endpoint of the final recipient of the frame. This frame shall be included in the frame only if the delivery mode subfield is set to 0b00 (normal unicast delivery), or 0b10 (broadcast delivery). In the case of broadcast delivery, the frame shall be delivered to the destination endpoint specified within the range 0x01-0xfe or to all active endpoints if specified as 0xff.

A destination endpoint value of 0x00 addresses the frame to the ZigBee device object (ZDO), resident in each device. A destination endpoint value of 0x01-0xfe addresses the frame to an application operating on that endpoint. A destination endpoint value of 0xff addresses the frame to all active endpoints except endpoint 0x00.

2.2.5.1.3 Group Address Field

The group address field is 16 bits in length and will only be present if the delivery mode sub-field of the frame control has a value of 0b11. In this case, the destination endpoint shall not be present. If the APS header of a frame contains a group address field, the frame will be delivered to all endpoints for which the group table in the device contains an association between that endpoint and the group identified by the contents of the group address field.

Devices where *nwkUseMulticast* is set to TRUE shall never set the group address field of an outgoing frame.

2.2.5.1.4 Cluster Identifier Field

The cluster identifier field is 16 bits in length and specifies the identifier of the cluster to which the frame relates and which shall be made available for filtering and interpretation of messages at each device that takes delivery of the frame. This field shall be present only for data or acknowledgement frames.

2.2.5.1.5 Profile Identifier Field

The profile identifier is two octets in length and specifies the ZigBee profile identifier for which the frame is intended and shall be used during the filtering of messages at each device that takes delivery of the frame. This field shall be present only for data or acknowledgement frames.

2.2.5.1.6 Source Endpoint Field

The source endpoint field is eight-bits in length and specifies the endpoint of the initial originator of the frame. A source endpoint value of 0x00 indicates that the frame originated from the ZigBee device object (ZDO) resident in each device. A source endpoint value of 0x01-0xfe indicates that the frame originated from an application operating on that endpoint.

2.2.5.1.7 APS Counter

This field is eight bits in length and is used as described in section 2.2.8.4.2 to prevent the reception of duplicate frames. This value shall be incremented by one for each new transmission.

2.2.5.1.8 Extended Header Sub-Frame

The extended header sub-frame contains further sub-fields and shall be formatted as illustrated in Figure 2.4.

Figure 2.4 Format of the Extended Header Sub-Frame

Octets: 1	**0/1**	**0/1**
Extended frame control	Block number	ACK bitfield

2.2.5.1.8.1 Extended Frame Control Field

The extended frame control field is eight-bits in length and contains information defining the use of fragmentation. The extended frame control field shall be formatted as illustrated in Figure 2.5.

Figure 2.5 Format of the Extended Frame Control Field

Bits: 0-1	**2-7**
Fragmentation	Reserved

The fragmentation sub-field is two bits in length and shall be set to one of the non-reserved values listed in Table 2.22.

Table 2.22 Values of the Fragmentation Sub-Field

Fragmentation Value $b_1 b_0$	**Description**
00	Transmission is not fragmented.
01	Frame is first fragment of a fragmented transmission.
10	Frame is part of a fragmented transmission but not the first part.
11	Reserved

2.2.5.1.8.2 Block Number

The block number field is one octet in length and is used for fragmentation control as follows: If the fragmentation sub-field is set to indicate that the transmission is not fragmented then the block number field shall not be included in the sub-frame. If the fragmentation sub-field is set to 01, then the block number field shall be included in the sub-frame and shall indicate the number of blocks in the fragmented transmission. If the fragmentation sub-field is set to 10, then the block number field shall be included in the sub-frame and shall indicate which block number of the transmission the current frame represents, taking the value 0x01 for the second fragment, 0x02 for the third, etc.

2.2.5.1.8.3 Ack Bitfield

The ack bitfield field is one octet in length and is used in an APS acknowledgement as described in section 2.2.8.4.5.2 to indicate which blocks of a fragmented ASDU have been successfully received. This field is only present if the frame type sub-field indicates an acknowledgement and the fragmentation sub-field indicates a fragmented transmission.

2.2.5.1.9 Frame Payload Field

The frame payload field has a variable length and contains information specific to individual frame types.

2.2.5.2 Format of Individual Frame Types

There are three defined frame types: data, APS command, and acknowledgement. Each of these frame types is discussed in the following sections.

2.2.5.2.1 Data Frame Format

The data frame shall be formatted as illustrated in Figure 2.6.

Figure 2.6 Data Frame Format

Octets: 1	**0/1**	**0/2**	**2**	**2**	**1**	**1**	**0/ Variable**	**Variable**
Frame control	Destination endpoint	Group address	Cluster identifier	Profile Identifier	Source endpoint	APS counter	Extended header	Frame payload
	Addressing fields							
APS header								APS payload

The order of the fields of the data frame shall conform to the order of the general APS frame as illustrated in Figure 2.2.

2.2.5.2.1.1 Data Frame APS Header Fields

The APS header field for a data frame shall contain the frame control, cluster identifier, profile identifier, source endpoint and APS counter fields. The destination endpoint, group address and extended header fields shall be included in a data frame according to the values of the delivery mode and extended header present sub-fields of the frame control field.

In the frame control field, the frame type sub-field shall contain the value that indicates a data frame, as shown in Table 2.20. All other sub-fields shall be set appropriately according to the intended use of the data frame.

2.2.5.2.1.2 Data Payload Field

For an outgoing data frame, the data payload field shall contain part or all of the sequence of octets that the next higher layer has requested the APS data service to transmit. For an incoming data frame, the data payload field shall contain all or part of the sequence of octets that has been received by the APS data service and that is to be delivered to the next higher layer.

2.2.5.2.2 APS Command Frame Format

The APS command frame shall be formatted as illustrated in Figure 2.7.

Figure 2.7 APS Command Frame Format

Octets: 1	**1**	**1**	**Variable**
Frame control	APS counter	APS command identifier	APS command payload
APS header		APS payload	

The order of the fields of the APS command frame shall conform to the order of the general APS frame as illustrated in Figure 2.2.

2.2.5.2.2.1 APS Command Frame APS Header Fields

The APS header field for an APS command frame shall contain the frame control and APS counter fields. In this version of the specification, the APS command frame shall not be fragmented and the extended header field shall not be present.

In the frame control field, the frame type sub-field shall contain the value that indicates an APS command frame, as shown in Table 2.20. The APS Command Payload shall be set appropriately according to the intended use of the APS command frame.

2.2.5.2.2.2 APS Command Identifier Field

The APS command identifier field identifies the APS command being used.

2.2.5.2.2.3 APS Command Payload Field

The APS command payload field of an APS command frame shall contain the APS command itself.

2.2.5.2.3 Acknowledgement Frame Format

The acknowledgement frame shall be formatted as illustrated in Figure 2.8.

Figure 2.8 Acknowledgement Frame Format

Octets: 1	**0/1**	**0/2**	**0/2**	**0/1**	**1**	**0/Variable**
Frame control	Destination endpoint	Cluster identifier	Profile identifier	Source endpoint	APS counter	Extended header
APS header						

The order of the fields of the acknowledgement frame shall conform to the order of the general APS frame as illustrated in Figure 2.2.

2.2.5.2.3.1 Acknowledgement Frame APS Header Fields

If the ack format field is not set in the frame control field, the destination endpoint, cluster identifier, profile identifier and source endpoint shall be present. This is not set for data frame acknowledgement. The extended header field shall be included in a data frame according to the value of the extended header present sub-field of the frame control field.

In the frame control field, the frame type sub-field shall contain the value that indicates an acknowledgement frame, as shown in Table 2.20. The extended header present sub-field shall contain the same value as in the frame to which this frame is an acknowledgement. All other sub-fields shall be set appropriately according to the intended use of the acknowledgement frame.

If the ack format field is set in the frame control field, the frame is an APS command frame acknowledgement and the destination endpoint, cluster identifier, profile identifier and source endpoint fields shall not be included. Alternatively, if an APS data frame is being acknowledged, the source endpoint field shall reflect the value in the destination endpoint field of the frame that is being acknowledged. Similarly, the destination endpoint field shall reflect the value in the source endpoint field of the frame that is being acknowledged. And the Cluster identifier and Profile identifier fields shall contain the same values as in the frame to which this frame is an acknowledgement.

The APS counter field shall contain the same value as the frame to which this frame is an acknowledgment.

Where the extended header is present, the fragmentation sub-field of the extended frame control field shall contain the same value as in the frame to which this frame is an acknowledgement. If fragmentation is in use for this frame, then the block number and ack bitfield fields shall be present. Where present, the block number field shall contain block number to which this frame is an acknowledgement. If fragmentation is in use, the acknowledgement frames shall be issued according to section 2.2.8.4.5.2 and not for each received frame unless the transmission window size is set to request acknowledgement of each frame.

2.2.6 Command Frames

This specification defines no command frames. Refer to section 4.4.9 for a thorough description of the APS command frames and primitives related to security.

2.2.7 Constants and PIB Attributes

2.2.7.1 APS Constants

The constants that define the characteristics of the APS sub-layer are presented in Table 2.23.

Table 2.23 APS Sub-Layer Constants

Constant	Description	Value
apscMaxDescriptorSize	The maximum number of octets contained in a non-complex descriptor.	64
apscMaxFrameRetries	The maximum number of retries allowed after a transmission failure.	3

Constant	Description	Value
apscAckWaitDuration	The maximum number of seconds to wait for an acknowledgement to a transmitted frame.	0.05 * (2**nwkcMaxDepth*) + (security encrypt/decrypt delay), where the (security encrypt/decrypt delay) = 0.1 (assume 0.05 per encrypt or decrypt cycle)
apscMinDuplicateRejectionTableSize	The minimum required size of the APS duplicate rejection table.	1
apscMinHeaderOverhead	The minimum number of octets added by the APS sub-layer to an ASDU.	0x0C
apsParentAnnounceBaseTimer	The base amount of delay before each broadcast parent announce is sent.	10
apsParentAnnounceJitterMax	The max amount of jitter that is added to the apsParentAnnounceBaseTimer before each broadcast parent announce is sent.	10

2.2.7.2 APS Information Base

The APS information base comprises the attributes required to manage the APS layer of a device. The attributes of the AIB are listed in Table 2.24. The security-related AIB attributes are described in section 4.4.10.

Table 2.24 APS IB Attributes

Attribute	Identifier	Type	Range	Description	Default
apsBindingTable	0xc1	Set	Variable	The current set of binding table entries in the device (see section 2.2.8.2.1).	Null set
apsDesignatedCoordinator	0xc2	Boolean	TRUE or FALSE	TRUE if the device should become the ZigBee Coordinator on startup, FALSE if otherwise.	FALSE
apsChannelMask	0xc3	IEEE 802.15.4 channel mask	Any legal mask for the PHY	The mask of allowable channels for this device to use for network operations.	All channels

Attribute	**Identifier**	**Type**	Range	**Description**	**Default**
apsUseExtended-PANID	0xc4	64-bit extended address	0x0000000000000000 to 0xfffffffffffffffe	The 64-bit address of a network to form or to join.	0x0000000000000000
apsGroupTable	0x0c5	Set	Variable	The current set of group table entries (see Table 2.25).	Null set
apsNonmember Radius	0xc6	Integer	0x00-0x07	The value to be used for the NonmemberRadius parameter when using NWK layer multicast.	2
apsUseInsecure-Join	0xc8	Boolean	TRUE or FALSE	A flag controlling the use of insecure join at startup.	FALSE
apsInter-frameDelay	0xc9	Integer	0x00 to 0xff (may be restricted by application profile)	Fragmentation parameter—the standard delay, in milliseconds, between sending two blocks of a fragmented transmission (see section 2.2.8.4.5).	Set by application profile
apsLastChannel Energy	0xca	Integer	0x00 - 0xff	The energy measurement for the channel energy scan performed on the previous channel just before a channel change (in accordance with [B1]).	Null set
apsLastChannel FailureRate	xcb	Integer	0-100 (decimal)	The latest percentage of transmission network transmission failures for the previous channel just before a channel change (in percentage of failed transmissions to the total number of transmissions attempted)	Null set

Attribute	**Identifier**	**Type**	Range	**Description**	**Default**
apsChannelTimer	0xcc	Integer	1-24 (decimal)	A countdown timer (in hours) indicating the time to the next permitted frequency agility channel change. A value of NULL indicates the channel has not been changed previously.	Null set
apsMaxWindow Size	0xcd	See Table 2.26	Variable	A table with the active endpoints and their respective *apsMaxWindowSize* where fragmentation is used (active endpoints not supporting fragmentations shall be omitted from the list).	Null set
apsParentAnnounceTimer	0xce	Integer	0 to apsParentAnnounceBaseTimer + apsParentAnnounceJitterMax	The value of the current countdown timer before the next Parent_annce is sent.	0

Table 2.25 Group Table Entry Format

Group ID	**Endpoint List**
16-bit group address	List of endpoints on this device which are members of the group.

Table 2.26 *apsMaxWindowSize* by Endpoint Number

Endpoint Number	**apsMaxWindowSize for the Endpoint Number**
0x01 - 0xff	Value of 1-8

2.2.8 Functional Description

2.2.8.1 Persistent Data

The APS is required to maintain a minimum set of data in persistent memory. This data set shall persist over power fail, device reset, or other processing events. The following data shall be maintained in persistent memory within APS:

- *apsBindingTable* (if supported on the device)
- *apsDesignatedCoordinator* (if supported on the device)
- *apsChannelMask*
- *apsUseExtendedPANID*
- *apsUseInsecureJoin*
- *apsGroupTable* (if supported on the device)
- Binding Table Cache (if the device is designated as a primary or backup binding table cache, see section 2.4.2.4)
- Discovery Cache (if the device is designated as a primary discovery cache, see section 2.4.2.1)
- Node Descriptor, Power Descriptor plus the Simple Descriptor(s) for each active endpoint on the device
- Network manager address

The method by which these data are made to persist is beyond the scope of this specification.

2.2.8.2 Binding

The APS may maintain a binding table, which allows ZigBee devices to establish a designated destination for frames from a given source endpoint and with a given cluster ID. Each designated destination shall represent either a specific endpoint on a specific device, or a group address.

2.2.8.2.1 Binding Table Implementation

A device designated as containing a binding table shall be able to support a binding table of implementation-specific length. The binding table shall implement the following mapping:

$$(a_s, e_s, c_s) = \{(a_{d1}|, e_{d1}|), (a_{d2}|, e_{d2}|) \ldots (a_{dn}|, e_{dn}|)\}$$

Where:

a_s	=	the address of the device as the source of the binding link
e_s	=	the endpoint identifier of the device as the source of the binding link
c_s	=	the cluster identifier used in the binding link
a_{di}	=	the i*th* destination address or destination group address associated with the binding link
e_{di}	=	the i*th* optional destination endpoint identifier associated with the binding link Note that e_{di} will only be present when a_{di} is a device address.

2.2.8.2.2 Binding

The APSME-BIND.request or APSME-UNBIND.request primitives initiate the procedure for creating or removing a binding link. Only a device supporting a binding table cache, or a device that wishes to store source bindings, shall initiate this procedure. If this procedure is initiated by another type of device, then the APSME shall issue the APSME-BIND.confirm or APSME-UNBIND.confirm primitive with the Status parameter set to ILLEGAL_REQUEST.

When this procedure is initiated, the APSME shall first extract the address and endpoint for both the source and destination of the binding link. If the DstAddrMode parameter has a value of 0x01, indicating group addressing, then only the source address is treated in the way just described. The 16-bit group address is used directly as a destination address and, in this case, no destination endpoint is specified. With this information, the APSME shall either create a new entry or remove the corresponding entry from its binding table, depending on whether the bind or unbind procedure, respectively, was initiated.

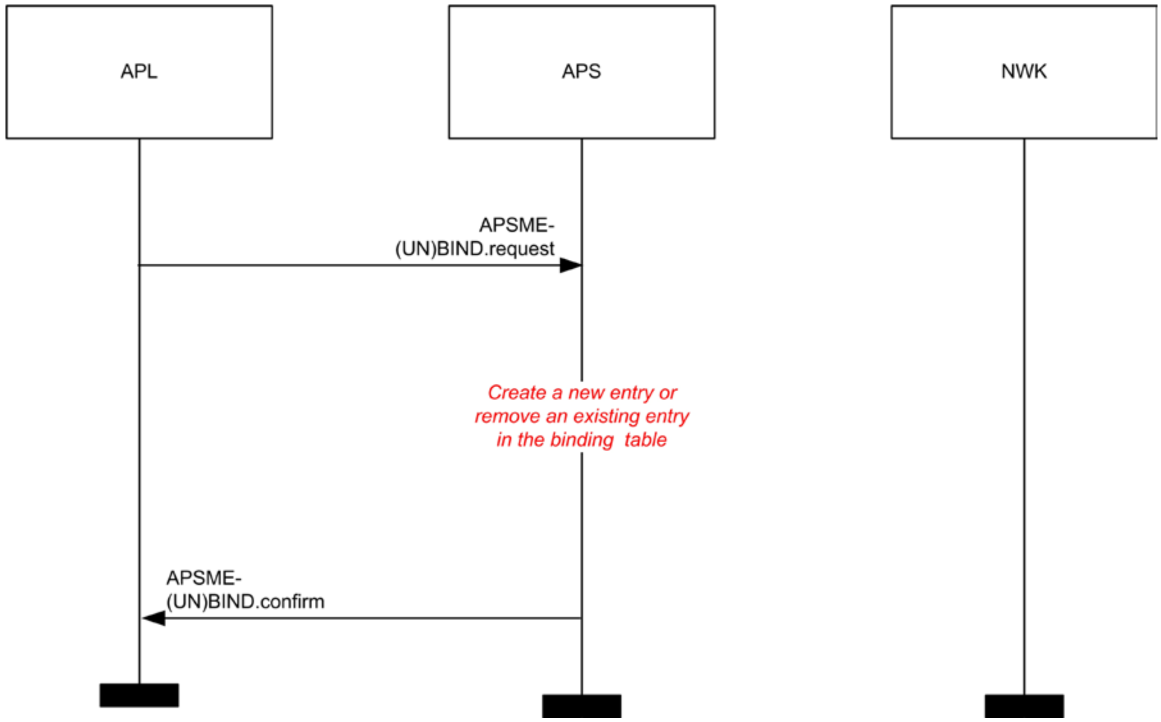
If a bind operation was requested, the APSME shall create a new entry in the binding table. The device shall only create a new entry in the binding table if it has the capacity to do so. If the binding table does not have capacity, then the APSME shall issue the APSME-BIND.confirm primitive with the Status parameter set to TABLE_FULL.

If an unbind operation was requested, the APSME shall search the binding table for an existing entry that matches the information contained in the initiation request. If an entry is not found, the APSME shall terminate the procedure and notify the NHLE of the invalid binding. This is achieved by issuing the APSME-UNBIND.confirm primitive with the Status parameter set to INVALID_BINDING. If an entry is found, the APSME shall remove the entry in the binding table.

If the binding link is successfully created or removed, the APSME shall notify the NHLE of the results of the binding attempt and the success of the procedure. This is achieved by issuing the APSME-BIND.confirm or APSME-UNBIND.confirm primitive, respectively, with the binding results and the Status parameter set to SUCCESS.

The procedure for a successful binding is illustrated in the MSC shown in Figure 2.9.

Figure 2.9. Binding on a Device Supporting a Binding Table



2.2.8.3 Group Addressing

The APS sub-layer shall maintain a group table, which allows endpoints to be associated with groups and allows group-addressed frames to be delivered selectively to those endpoints that are associated in the table with a particular group.

The list of group addresses in the APS sub-layer group table shall be kept consistent with the list of group IDs in the NWK layer group table, stored in the *nwkGroupIDTable* attribute.

2.2.8.3.1 The Group Table

For purposes of this discussion, the group table shall be viewed as a set of associations between groups and endpoints as follows:

$$\{(g_1 - ep_{11}, ep_{12} \ldots ep_{1n}), (g_2 - ep_{21}, ep_{22} \ldots ep_{2m}) \ldots (g_i - ep_{i1}, ep_{i2} \ldots ep_{ik})\}$$

where:

g_i	=	the i^{th} group represented in the table
ep_{ij}	=	the j^{th} endpoint associated with the i^{th} group

Implementers of this specification are free to implement the group table in any manner that is convenient and efficient, as long as it represents the associations just described.

2.2.8.4 Transmission, Reception, and Acknowledgement

This section describes the fundamental procedures for transmission, reception, and acknowledgement.

2.2.8.4.1 Transmission

Only those devices that are currently part of a network shall send frames from the APS sub-layer. If any other device receives a request to transmit a frame, it shall discard the frame and notify the instigating layer of the error. An APSDE-DATA.confirm primitive with a status of CHANNEL_ACCESS_FAILURE indicates that the attempt at transmission of the frame was unsuccessful due to the channel being busy.

All frames handled by or generated within the APS sub-layer shall be constructed according to the general frame format specified in section 2.2.5.1 and transmitted using the NWK layer data service.

Transmissions employing delivery modes 0b00 (Normal Unicast) and 0b10 (Broadcast) shall include both the source endpoint and destination endpoint fields. Group addressed transmissions, having a delivery mode sub-field value of 0b11 shall contain a source endpoint field and group address field, but no destination endpoint field. Note that other endpoints on the source device are legal group members and possible destinations for group-addressed frames.

For all devices where the transmission is due to a binding table entry stored on the source device, the APSDE of the source device shall determine whether the binding table entry contains a unicast destination device address or a destination group address. In the case where a binding table entry contains a unicast destination device address and this destination device address is that of the source device itself, the APSDE shall issue an APSDE-DATA.indication primitive to the next higher layer and shall not transmit a frame. Otherwise, the APSDE shall transmit the frame to the 16-bit NWK address corresponding to the destination address indicated by the binding table entry, and the delivery mode sub-field of the frame control field shall be set to 0b00. In the case where the binding table entry contains a destination group address and nwkUseMulticast is FALSE, the delivery mode sub-field of the frame control field shall have a value of 0b11, the destination group address shall be placed in the APS header, and the destination endpoint shall be omitted. The frame shall then be broadcast using the NLDE-DATA.request primitive and employing a broadcast address of 0xfffd. In the case where the binding table entry contains a destination group address and nwkUseMulticast is TRUE, the delivery mode sub-field of the frame control field shall have a value of 0b10 and the destination endpoint shall have a value of 0xff. The frame shall then be multicast using the NLDE-DATA.request primitive and employing the group address supplied in the binding table entry.

If security is required, the frame shall be processed as described in section 4.4.

If fragmentation is required, and is permitted for this frame, then the frame shall be processed as described in section 2.2.8.4.5.

When the frame is constructed and ready for transmission, it shall be passed to the NWK data service with suitable destination and source addresses. In addition, the APS layer shall ensure that route discovery is enabled at the network layer. An APDU transmission is initiated by issuing the NLDE-DATA.request primitive to the NWK layer and the results of the transmission returned via the NLDE-DATA.confirm primitive.

2.2.8.4.2 Reception and Rejection

The APS sub-layer shall be able to filter frames arriving via the NWK layer data service and only present the frames that are of interest to the NHLE.

If the APSDE receives a secured frame, it shall process the frame as described in section 4.4 to remove the security.

If the APSDE receives a frame containing the destination endpoint field, then the APSDE shall pass it directly to the NHLE at the destination endpoint supplied, unless it is part of an incomplete fragmented transmission or it is determined to have been a duplicate of a frame that has been passed up previously. Subject to the same incomplete fragmented transmission and duplicate frame detection, if the destination endpoint is set to the broadcast endpoint (0xff) and the DstAddrMode parameter of the received NLDE-DATA.indication primitive was not 0x01, then the APSDE shall also present the frame to all non-reserved endpoints (0x01-0xfe) supported by the NHLE.

If the APSDE of a device receives a transmission with the delivery mode sub-field of the frame control field set to 0b11, indicating group addressing, it shall deliver the frame to each endpoint on the device that is associated in the group table with the 16-bit group address found in the group address field of the APS header. Similarly, if the APSDE of a device receives a NLDE-DATA.indication primitive where the DstAddrMode parameter has a value of 0x01, also indicating group addressing, it shall deliver the frame to each endpoint on the device that is associated in the group table with the 16-bit group address given as the value of the DstAddr parameter. In either case, it shall search the group table and, for each endpoint associated with the given group address, it shall issue the NLDE-DATA.indication primitive to the next higher layer with a value of the DstEndpoint parameter equal to the number of the associated endpoint. All other parameters of the NLDE-DATA.indication primitive shall remain the same for all instances of the primitive issued.

The APSDE shall maintain a duplicate rejection table to include at least source address, APS counter, and timing information, such that frames transmitted according to this specification and received more than once are identified as duplicates and only delivered to the NHLE once. The size of this table shall be at least *apscMinDuplicateRejectionTableSize*.

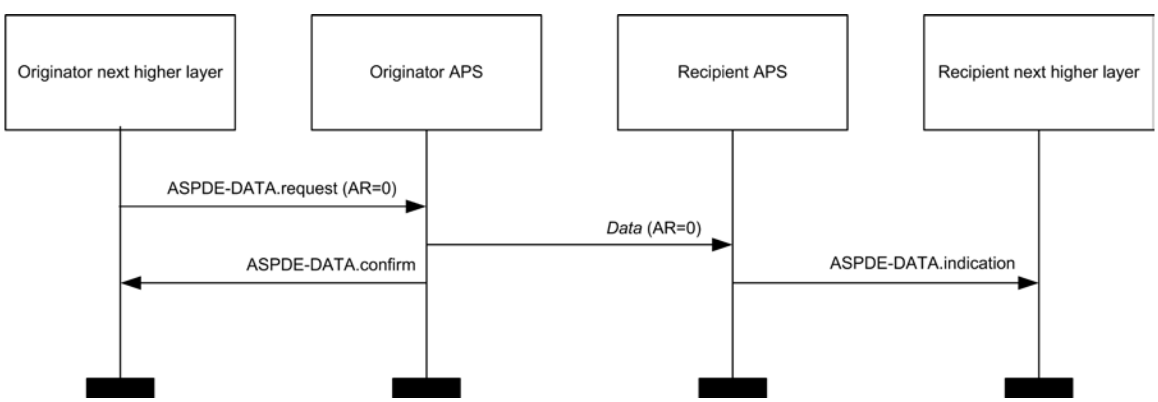
2.2.8.4.3 Use of Acknowledgements

A data or APS command frame shall be sent with its acknowledgement request sub-field set appropriately for the frame. An acknowledgement frame shall always be sent with the acknowledgement request sub-field set to 0. Similarly, any frame that is broadcast or multicast shall be sent with its acknowledgement request sub-field set to 0.

2.2.8.4.3.1 No Acknowledgement

A frame that is received by its intended recipient with its acknowledgement request (AR) sub-field set to 0 shall not be acknowledged. The originating device shall assume that the transmission of the frame was successful. Figure 2.10 shows the scenario for transmitting a single frame of data from an originator to a recipient without requiring an acknowledgement. In this case, the originator transmits the data frame with the AR sub-field equal to 0.

Figure 2.10 Successful Data Transmission Without an Acknowledgement



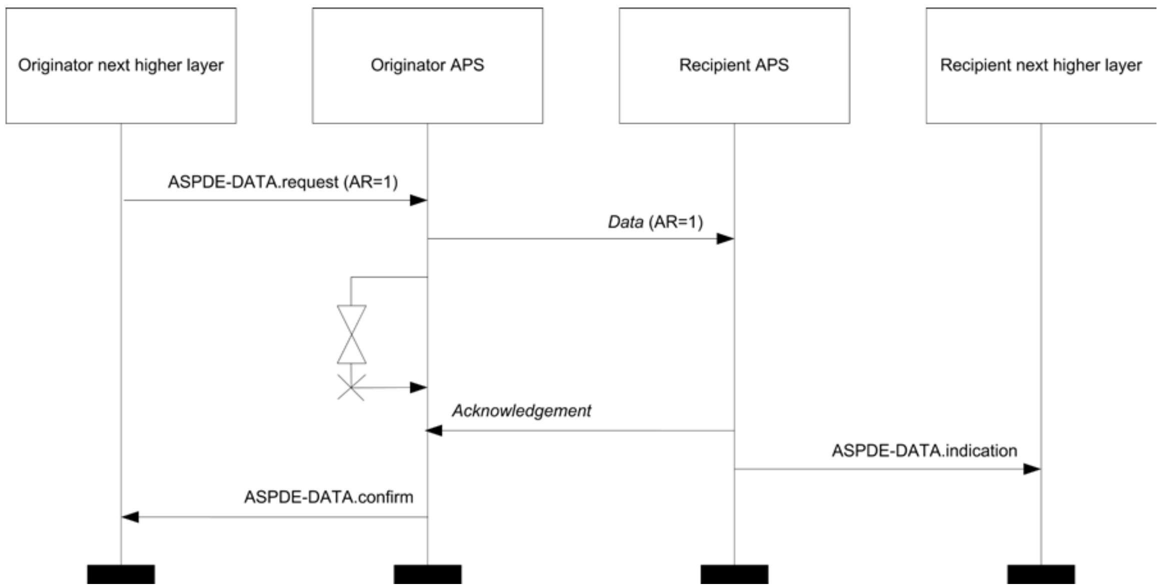
2.2.8.4.3.2 Acknowledgement

A frame that is received by its intended recipient with its acknowledgement request (AR) sub-field set to 1 shall be acknowledged. If the intended recipient correctly receives the frame, it shall generate and send an acknowledgement frame to the originator of the frame that is being acknowledged.

The transmission of an acknowledgement frame shall commence when the APS sub-layer determines that the frame is valid.

Figure 2.11 shows the scenario for transmitting a single frame of data from an originator to a recipient with an acknowledgement. In this case, the originator indicates to the recipient that it requires an acknowledgement by transmitting the data frame with the AR sub-field set to 1.

Figure 2.11 Successful Data Transmission with an Acknowledgement



2.2.8.4.4 Retransmissions

A device that sends a frame with its acknowledgement request sub-field set to 0 shall assume that the transmission was successfully received and shall hence not perform the retransmission procedure.

A device that sends a frame with its acknowledgement request sub-field set to 1 shall wait for a maximum of *apscAckWaitDuration* seconds for the corresponding acknowledgement frame to be received.

If an acknowledgement frame is received within *apscAckWaitDuration* seconds, containing the same cluster identifier and APS counter as the original frame and has a source endpoint equal to the destination endpoint to which the original frame was transmitted, the transmission shall be considered successful and no further action shall be taken by the device. If an acknowledgement is not received within *apscAckWaitDuration* seconds, or an acknowledgement is received within *apscAckWaitDuration* seconds but contains an unexpected cluster identifier or APS counter or has a source endpoint that is not equal to the destination endpoint to which the original frame was transmitted, the device shall conclude that the single transmission attempt has failed.

If a single transmission attempt has failed, the device shall repeat the process of transmitting the frame and waiting for the acknowledgement, up to a maximum of *apscMaxFrameRetries* times. If an acknowledgement is still not received after *apscMaxFrameRetries* retransmissions, the APS sub-layer shall assume the transmission has failed and notify the next higher layer of the failure.

Retransmissions of a secured frame shall use a frame counter greater than the original frame.

2.2.8.4.5 Fragmented Transmissions

Where an ASDU is too large to be transmitted within a single MAC data frame, an acknowledged unicast transmission was requested, and fragmentation is permitted for this frame, the ASDU shall be fragmented into a number of smaller byte strings, here referred to as “blocks.” Each block is transmitted in a separate frame.

A “transmission window” is used to arrange an orderly transaction. The window size is set by the stack profile, and may be set as high as eight blocks. The protocol below arranges that all blocks in a transmission window must be received and acknowledged before the window can move on. An acknowledgement is sent when all blocks in the transmission window have been successfully received or, according to the protocol below, to request retransmission of one or more unreceived blocks.

Transactions not using APS acknowledgements may not be fragmented. Multicast and broadcast transmissions are not permitted to use fragmentation.

2.2.8.4.5.1 Transmission

All blocks in a fragmented transmission shall have the same APS Counter value. The extended header sub-frame shall be included in the frame. The fragmentation sub-field of the extended frame control field shall be set to 0b01 for the first block and 0b10 for all subsequent blocks of the fragmented transmission. The block number field shall indicate the total number of blocks in the transmission in the first block, shall take the value 0x01 in the second block, and thereafter shall be incremented for each subsequent block.

A transmission window shall be maintained, initially covering blocks 0 to (*apscMaxWindowSize-1*), or the total number of blocks if this is less.

If security is required, then each frame shall be processed independently, as described in clause 4. Following transmission of each block, the APS shall start a timer. If there are more unacknowledged blocks to send in the current transmission window, then after a delay of *apsInterframeDelay* milliseconds the next block shall be passed to the NWK data service. Otherwise, the timer shall be set for *apscAckWaitDuration* seconds.

A retryCounter parameter shall be maintained and is reset to zero for each new transaction. If an *apscAckWaitDuration* timer expires, then the block with the lowest unacknowledged block number shall be passed to the NWK data service again, and the retryCounter parameter shall be incremented. If the retryCounter parameter reaches the value *apscMaxFrameRetries*, the transaction shall be deemed to have failed, and an APSDE-DATA.confirm primitive returned to the NHLE with a status value of NO_ACK.

On receipt of an acknowledgement frame with matching values in the APS counter, block number, and addressing fields, outgoing blocks are acknowledged as described in the section below. If at least one previously unacknowledged block is acknowledged, then the timer shall be stopped and the retryCounter parameter reset. If all blocks in the current transmission window have been acknowledged, then the transmission window shall be increased by *apscMaxWindowSize*. If all blocks have now been transmitted and acknowledged, then the transaction is complete, and an APSDE-DATA.confirm primitive shall be returned to the NHLE with a status value of SUCCESS. Otherwise, the block with the lowest unacknowledged block number shall be passed to the NWK data service.

2.2.8.4.5.2 Reception and Rejection, and Acknowledgements

If the fields required for a fragmentation-enabled transmission are not present in the frame it shall be rejected. Also, any frames with parameters that fall outside the bounds of this protocol shall be rejected.

If an incoming fragmented transaction is already in progress but the addressing and APS counter fields do not match those of the received frame, then the received frame may optionally be rejected or handled independently as a further transaction.

If no transaction is in progress and a fragmented frame is received, then reassembly shall be attempted. Initially the receive window shall be from 0 to (*apscMaxWindowSize-1*).

If a transaction is initiated with APS counter and source address field values matching a previously received transaction, then the new transaction may be rejected as a duplicate.

Upon receipt of the first received block (not necessarily block 0) in the first window, or when an acknowledgement is generated, the receiver shall set a timer for *apscAckWaitDuration*.

If the receive window does not move forward within any (*apscAckWaitDuration* + *apscAckWaitDuration* * *apscMaxFrameRetries*) time period, the transaction shall be deemed to have failed. The receiver may send an acknowledgement to the sender with the block or blocks missed.

If all blocks in the current receive window have been received and a block is received with a block number higher than the current receive window, then the receive window shall be increased by *apsMaxWindowSize* blocks.

Additionally an APS acknowledgement shall be generated for the window if any one of the following circumstances occurs: (1) the last block in the entire fragmented transmission is received, (2) the last block in the window is received, (3) a block is received and all subsequent blocks in the window have been previously received and acknowledged. If a block is received with its block number value outside of the current window, then an acknowledgement shall NOT be generated.

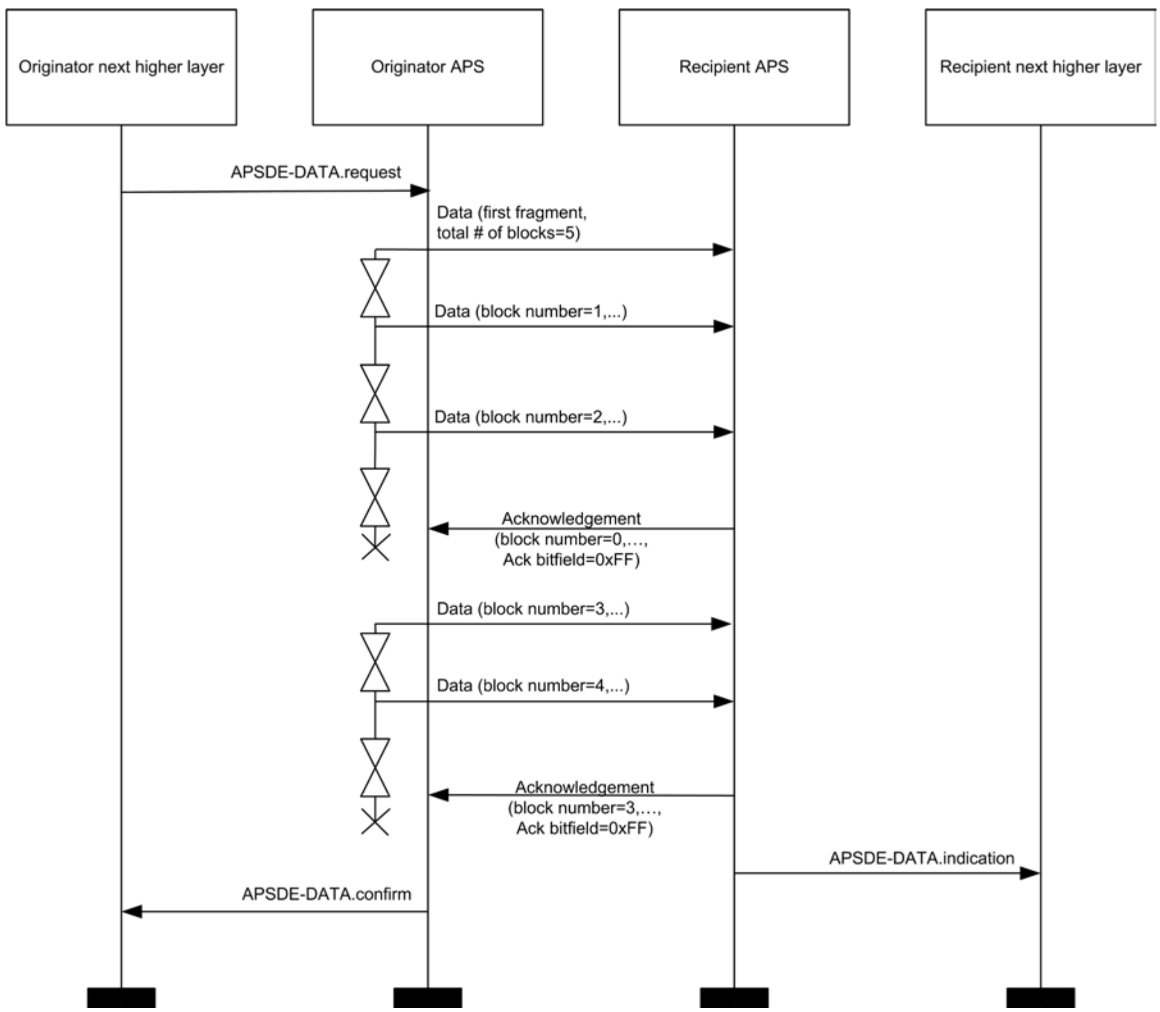
Once all blocks in the transaction have been received, the APS shall issue an APSDE-DATA.indication primitive containing the reassembled message, and the transaction shall be deemed to be complete. A period of persistence of *apscAckWaitDuration* seconds is encouraged in order to facilitate any retransmission of the final acknowledgement.

Where generated, the acknowledgement is formatted according to the acknowledgement frame format specified in section 2.2.5.2.3. The APS counter field shall reflect the value in the corresponding field of the frame(s) being acknowledged. The block number field shall contain the value of the lowest block number in the current receive window, using the value 0 as the value of the first block.

The first bit of the ack bitfield shall be set to 1 if the first fragment in the current receive window has been correctly received, and 0 otherwise. Subsequent bits shall be set similarly, with values corresponding to subsequent fragments in the current receive window. If *apsMaxWindowSize* is less than 8, then the remaining bits shall be set to 1.

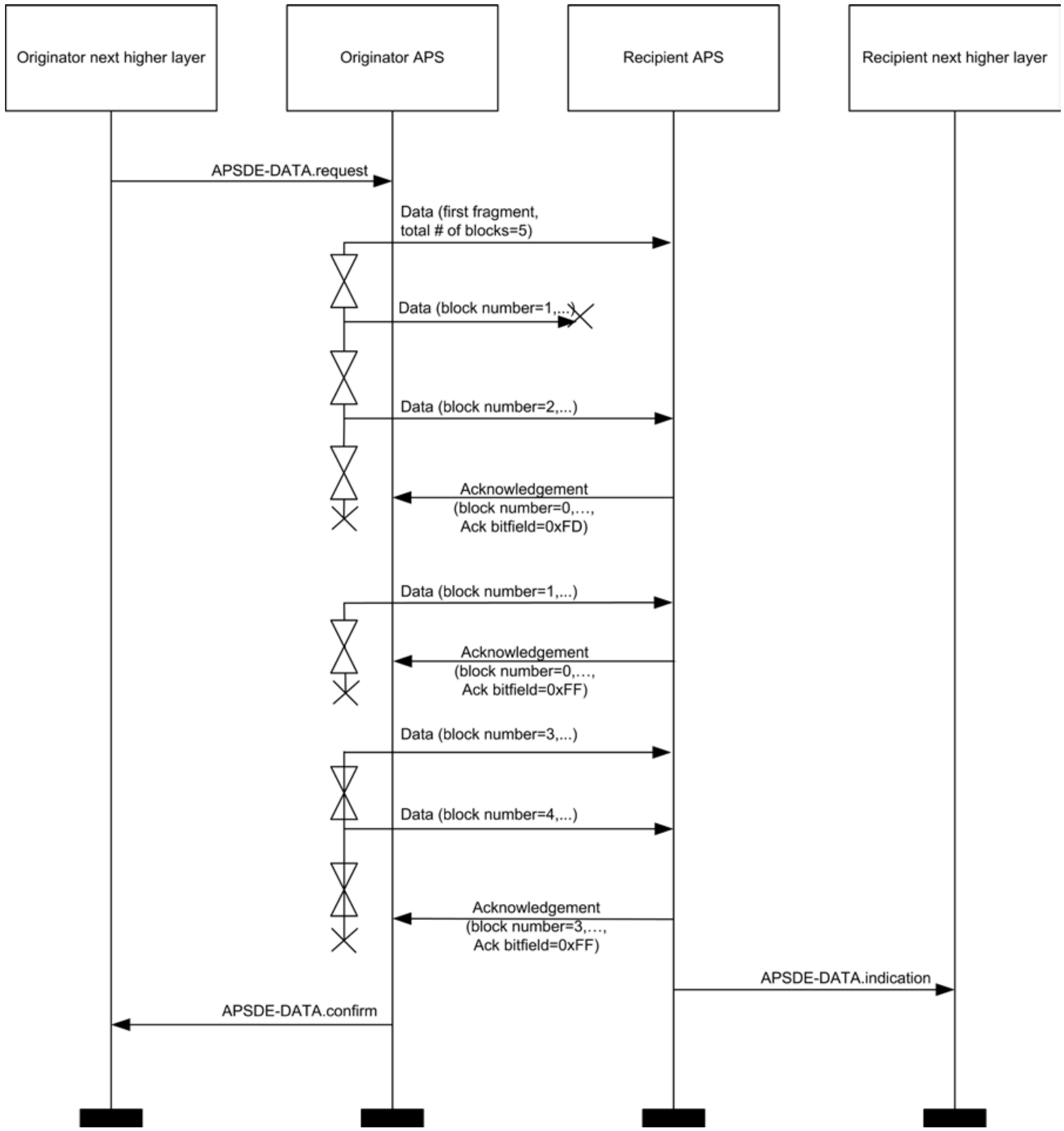
The process is illustrated in the following diagrams. In Figure 2.12, the transmission is successful immediately. (These examples assume that *apscMaxWindowSize* takes the value 3).

Figure 2.12 Successful Data Transmission with Fragmentation



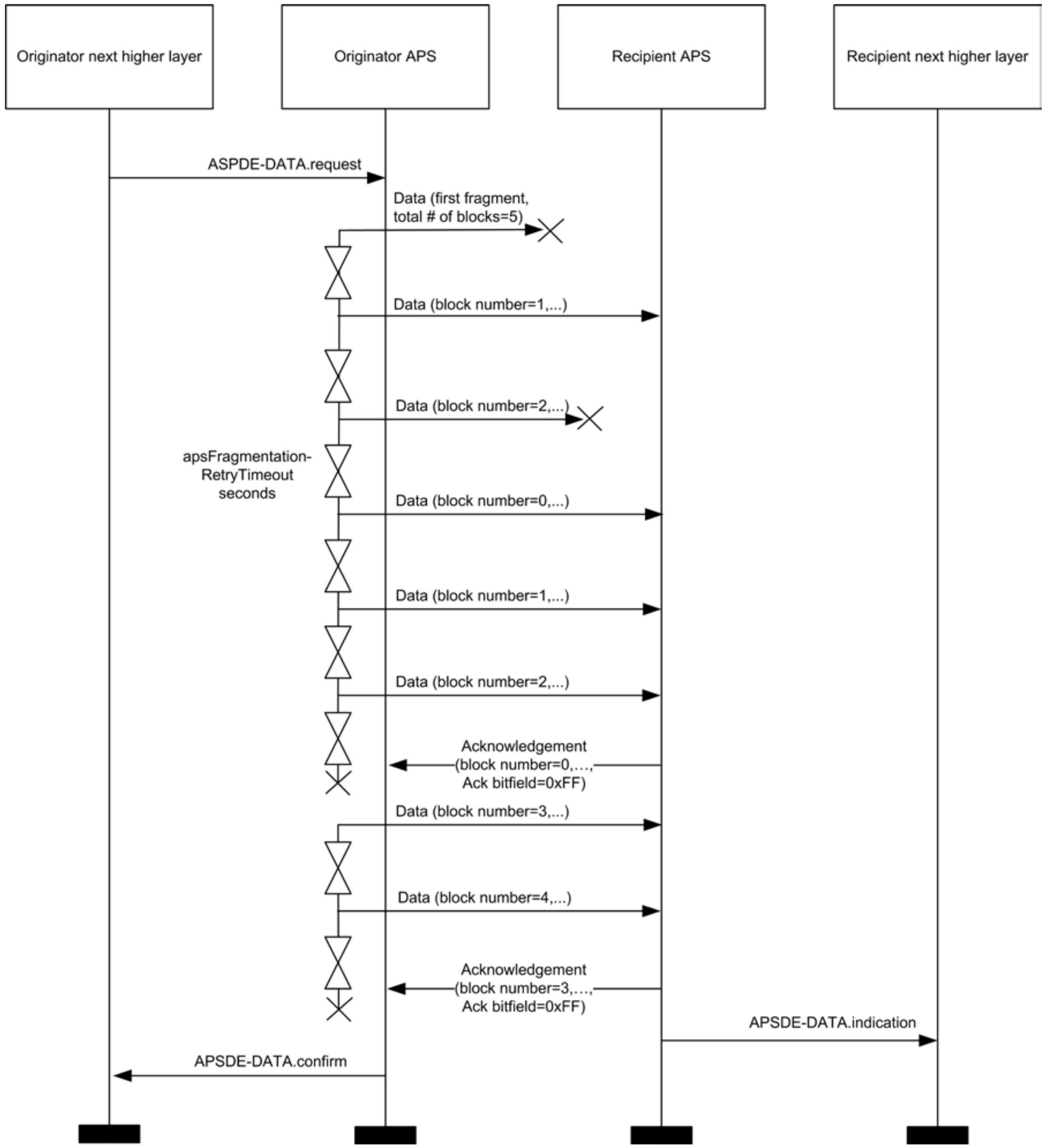
In Figure 2.13, a single frame is lost during transit across the network, and is retransmitted.

Figure 2.13 Fragmented Data Transmission with a Single Retransmission



In Figure 2.14, multiple frames are lost in the network, including a frame which has the highest block number in the window. Slightly more traffic is required in this case, but the source backs off and gives the network a chance to recover, and the ASDU is delivered successfully.

Figure 2.14 Fragmented Data Transmission with Multiple Retransmissions



2.2.9 APS Sub-Layer Status Values

Application support (APS) sub-layer confirm primitives often include a parameter that reports on the status of the request to which the confirmation applies. Values for APS sub-layer Status parameters appear in Table 2.27.

2411 **Table 2.27 APS Sub-layer Status Values**

Name	Value	Description
SUCCESS	0x00	A request has been executed successfully.
ASDU_TOO_LONG	0xa0	A transmit request failed since the ASDU is too large and fragmentation is not supported.
DEFRAG_DEFERRED	0xa1	A received fragmented frame could not be defragmented at the current time.
DEFRAG_UNSUPPORTED	0xa2	A received fragmented frame could not be defragmented since the device does not support fragmentation.
ILLEGAL_REQUEST	0xa3	A parameter value was out of range.
INVALID_BINDING	0xa4	An APSME-UNBIND.request failed due to the requested binding link not existing in the binding table.
INVALID_GROUP	0xa5	An APSME-REMOVE-GROUP.request has been issued with a group identifier that does not appear in the group table.
INVALID_PARAMETER	0xa6	A parameter value was invalid or out of range.
NO_ACK	0xa7	An APSDE-DATA.request requesting acknowledged trans-mission failed due to no acknowledgement being received.
NO_BOUND_DEVICE	0xa8	An APSDE-DATA.request with a destination addressing mode set to 0x00 failed due to there being no devices bound to this device.
NO_SHORT_ADDRESS	0xa9	An APSDE-DATA.request with a destination addressing mode set to 0x03 failed due to no corresponding short address found in the address map table.
NOT_SUPPORTED	0xaa	An APSDE-DATA.request with a destination addressing mode set to 0x00 failed due to a binding table not being supported on the device.
SECURED_LINK_KEY	0xab	An ASDU was received that was secured using a link key.
SECURED_NWK_KEY	0xac	An ASDU was received that was secured using a network key.

Name	Value	Description
SECURITY_FAIL	0xad	An APSDE-DATA.request requesting security has resulted in an error during the corresponding security processing.
TABLE_FULL	0xae	An APSME-BIND.request or APSME.ADD-GROUP.request issued when the binding or group tables, respectively, were full.
UNSECURED	0xaf	An ASDU was received without any security.
UNSUPPORTED_ATTRIBUTE	0xb0	An APSME-GET.request or APSME-SET.request has been issued with an unknown attribute identifier.

2.3 The ZigBee Application Framework

2.3.1 Creating a ZigBee Profile

The key to communicating between devices on a ZigBee network is agreement on a profile.

An example of a profile would be home automation. This ZigBee profile permits a series of device types to exchange control messages to form a wireless home automation application. These devices are designed to exchange well-known messages to effect control such as turning a lamp on or off, sending a light sensor measurement to a lighting controller, or sending an alert message if an occupancy sensor detects movement.

An example of another type of profile is the device profile that defines common actions between ZigBee devices. To illustrate, wireless networks rely on the ability for autonomous devices to join a network and discover other devices and services on devices within the network. Device and service discovery are features supported within the device profile.

2.3.1.1 Getting a Profile Identifier from the ZigBee Alliance

ZigBee defines profiles in two separate classes: manufacturer-specific and public. The exact definition and criteria for these classes are an administrative issue within the ZigBee Alliance and outside the scope of this document. For the purposes of this technical specification, the only criterion is for profile identifiers to be unique. To that end, every profile effort must start with a request to the ZigBee Alliance for allocation of a profile identifier. Once the profile identifier is obtained, that profile identifier permits the profile designer to define the following:

- Device descriptions
- Cluster identifiers

The application of profile identifiers to market space is a key criterion for issuance of a profile identifier from the ZigBee Alliance. The profile needs to cover a broad enough range of devices to permit interoperability to occur between devices, without being overly broad and resulting in a shortage of cluster identifiers to describe their interfaces. Conversely, the profile cannot be defined to be too narrowly, resulting in many devices described by individual profile identifiers, resulting in a waste of the profile identifier addressing space and interoperability issues in describing how the devices are interfaced. Policy groups within the ZigBee Alliance will establish criteria on how profiles are to be defined and to help requestors tailor their profile identifier requests.

2.3.1.2 Defining Device Descriptions and Clusters

The profile identifier is the main enumeration feature within the ZigBee protocol. Each unique profile identifier defines an associated enumeration of device descriptions and cluster identifiers. For example, for profile identifier "1", there exists a pool of device descriptions described by a 16-bit value (meaning there are 65,536 possible device descriptions within each profile) and a pool of cluster identifiers described by a 16-bit value (meaning there are 65,536 possible cluster identifiers within each profile). Each cluster identifier also supports a pool of attributes described by a 16-bit value. As such, each profile identifier has up to 65,536 cluster identifiers and each of those cluster identifiers contains up to 65,536 attributes. It is the responsibility of the profile developer to define and allocate device descriptions, cluster identifiers, and attributes within their allocated profile identifier. Note that the definition of device descriptions, cluster identifiers, and attribute identifiers must be undertaken with care to ensure efficient creation of simple descriptors and simplified processing when exchanging messages.

For public profile identifiers defined within the ZigBee Alliance, a cluster library has been created which provides a common definition and enumeration of clusters and their attributes. The cluster library is designed to sponsor re-use of cluster and attribute definitions across application profiles. By convention, when public profiles employ the cluster library, they will share a common enumeration and definition of cluster and attribute identifiers.

Device descriptions and cluster identifiers must be accompanied by knowledge of the profile identifier to be processed. Prior to any messages being directed to a device, it is assumed by the ZigBee protocol that service discovery has been employed to determine profile support on devices and endpoints. Likewise, the binding process assumes similar service discovery and profile matching has occurred, since the resulting match is distilled to source address, source endpoint, cluster identifier, destination address, and destination endpoint.

2.3.1.3 Deploying the Profile on Endpoints

A single ZigBee device may contain support for many profiles, provide for subsets of various cluster identifiers defined within those profiles, and may support multiple device descriptions. This capability is defined using a hierarchy of addressing within the device as follows:

- **Device:** The entire device is supported by a single radio with a unique IEEE and NWK address.
- **Endpoints:** This is an 8-bit field that describes different applications that are supported by a single radio. Endpoint 0x00 is used to address the device profile, which each ZigBee device must employ, endpoint 0xff is used to address all active endpoints (the broadcast endpoint). Consequently, a single physical ZigBee radio can support up to 254 applications on endpoints 0x01-0xfe. Note that endpoints 0xf1-0xfe can only be used for ZigBee Alliance approved applications.

It is an application decision as to how to deploy applications on a device endpoint and which endpoints to advertise. The only requirement is that simple descriptors be created for each endpoint and those descriptors made available for service discovery.

2.3.1.4 Enabling Service Discovery

Once a device is created to support specific profiles and made consistent with cluster identifier usage for device descriptions within those profiles, the applications can be deployed. To do this, each application is assigned to individual endpoints and each described using simple descriptors (an endpoint can support only a single application profile). It is via the simple descriptors and other service discovery mechanisms described in the ZigBee device profile that service discovery is enabled, binding of devices is supported, and application messaging between complementary devices is facilitated.

One important point is that service discovery is made on the basis of profile identifier, input cluster identifier list, and output cluster identifier list (device description is notably missing). The device description is simply a convention for specifying mandatory and optional cluster identifier support within devices of that type for the indicated profile. Additionally, it is expected that the device description enumeration would be employed within PDAs or other assisted binding devices to provide external descriptions of device capabilities.

2.3.1.5 Mixing Standard and Proprietary Profiles

As an example, a ZigBee device could be developed to ZigBee public profile identifier “XX.” If a manufacturer wanted to deploy a ZigBee device supporting public profile “XX” and also provide manufacturer specific extensions, these extensions could be added to the manufacturer’s implementation of public profile “XX” if manufacturer extensions are supported within the definition of profile “XX.” Alternatively, if manufacturer extensions are not supported or the type of desired manufacturer extensions aren’t supported in profile “XX,” the manufacturer may deploy the extensions in a separate manufacturer-specific profile identifier advertised on a separate endpoint within the same physical device. In either case, devices that support the profile identifier “XX” but not containing the manufacturer extensions, would only advertise support for the base features of public profile identifier “XX” and could not respond to or create messages using the manufacturer extensions.

2.3.1.6 Enabling Backward Compatibility

In the previous example, a device is created using ZigBee public profile identifier “XX.” If the ZigBee Alliance were to update this public profile at a later time to add new features, the revisions could either be incorporated directly into public profile identifier “XX” if such extensions are supported via the definition of the profile, or could be introduced into a new public profile with a new profile identifier (say “XY”). Assuming extensibility is not supported in public profile “XX,” devices manufactured with just profile identifier “XX” could still be compatible with newer devices manufactured later by having the newer devices advertise support for both profile identifier “XX” and profile identifier “XY.” In this manner, the newer device may communicate with older devices using profile identifier “XX”; however, it may also communicate with newer devices using profile identifier “XY” from within the same application. The service discovery feature within ZigBee enables devices on the network to determine the level of support.

It is the goal of the ZigBee Alliance to provide extensibility, both for manufacturer extensions to public profiles as well as future enhancements to public profiles. That goal includes maintaining those extensions and enhancements within the same profile identifier whenever possible. This section illustrates that the profile definition features within ZigBee permit deployment of manufacturer extensions and feature enhancements, whether the goal of profile extensibility is achievable or not. The subject of profile extensibility, both for manufacturer extensions and feature enhancements, is beyond the scope of this document and addressed in other Alliance documents.

2.3.2 ZigBee Descriptors

ZigBee devices describe themselves using descriptor data structures. The actual data contained in these descriptors is defined in the individual device descriptions. There are five descriptors: node, node power, simple, complex, and user, shown in Table 2.28.

Table 2.28 ZigBee Descriptors

Descriptor Name	Status	Description
Node	M	Type and capabilities of the node.
Node power	M	Node power characteristics.
Simple	M	Device descriptions contained in node.
Complex	O	Further information about the device descriptions.
User	O	User-definable descriptor.

2.3.2.1 Transmission of Descriptors

The node, node power, simple, and user descriptors shall be transmitted in the order that they appear in their respective tables, i.e., the field at the top of the table is transmitted first and the field at the bottom of the table is transmitted last. Each individual field shall follow the transmission order specified in section 1.2.1.4.

Each descriptor shall be less than or equal to *apscMaxDescriptorSize* unless provision has been made to enable transmission of discovery information without the mandatory use of fragmentation.

In the case of the Simple Descriptor (see 2.3.2.5), transmission primitives exist which permit the descriptor to extend beyond *apscMaxDescriptorSize* (see 2.4.3.1.22 and 2.4.4.2.20). When extended transmission primitives are employed, the standard transmission primitives (see 2.4.3.1.5 and 2.4.4.2.5) require transmission of an abbreviated Simple Descriptor, and the Node Descriptor of the device shall indicate availability of extended transmission primitives (see 2.3.2.3.12).

The complex descriptor shall be formatted and transmitted as illustrated in Figure 2.15.

Figure 2.15 Format of the Complex Descriptor

Octets: 1	**Variable**	**…**	**Variable**
Field count	Field 1	…	Field *n*

Each field included in the complex descriptor shall be formatted as illustrated in Figure 2.16.

Figure 2.16 Format of an Individual Complex Descriptor Field

Octets: 1	**Variable**
Compressed XML tag	Field data

2.3.2.1.1 Field Count Field

The field count field is one octet in length and specifies the number of fields included in the Complex Descriptor, each formatted as illustrated in Figure 2.16.

2.3.2.1.1.1 Compressed XML Tag Field

The compressed XML tag field is one octet in length and specifies the XML tag for the current field. The compressed XML tags for the complex descriptor are listed in Table 2.41.

2.3.2.1.1.2 Field Data Field

The field data field has a variable length and contains the information specific to the current field, as indicated by the compressed XML tag field.

2.3.2.2 Discovery via Descriptors

Descriptor information is queried in the ZDO management entity device and service discovery, using the ZigBee device profile request primitive addressed to endpoint 0. For details of the discovery operation, see section 2.4.2.1. Information is returned via the ZigBee device profile indication primitive.

The node, node power, complex, and user descriptors apply to the complete node. The simple descriptor must be specified for each endpoint defined in the node. If a node contains multiple subunits, these will be on separate endpoints and the specific descriptors for these endpoints are read by including the relevant endpoint number in the ZigBee device profile primitive.

2.3.2.3 Node Descriptor

The node descriptor contains information about the capabilities of the ZigBee node and is mandatory for each node. There shall be only one node descriptor in a node.

The fields of the node descriptor are shown in Table 2.29 in their order of transmission.

Table 2.29 Fields of the Node Descriptor

Field Name	Length (Bits)
Logical type	3
Complex descriptor available	1
User descriptor available	1
Reserved	3
APS flags	3
Frequency band	5
MAC capability flags	8
Manufacturer code	16
Maximum buffer size	8
Maximum incoming transfer size	16
Server mask	16
Maximum outgoing transfer size	16
Descriptor capability field	8

2.3.2.3.1 Logical Type Field

The logical type field of the node descriptor is three bits in length and specifies the device type of the ZigBee node. The logical type field shall be set to one of the non-reserved values listed in Table 2.30.

Table 2.30 Values of the Logical Type Field

Logical Type Value $b_2b_1b_0$	Description
000	ZigBee coordinator
001	ZigBee router
010	ZigBee end device
011-111	Reserved

2.3.2.3.2 Complex Descriptor Available Field

The complex descriptor available field of the node descriptor is one bit in length and specifies whether a complex descriptor is available on this device. If this field is set to 1, a complex descriptor is available. If this field is set to 0, a complex descriptor is not available.

2.3.2.3.3 User Descriptor Available Field

The user descriptor available field of the node descriptor is one bit in length and specifies whether a user descriptor is available on this device. If this field is set to 1, a user descriptor is available. If this field is set to 0, a user descriptor is not available.

2.3.2.3.4 APS Flags Field

The APS flags field of the node descriptor is three bits in length and specifies the application support sub-layer capabilities of the node.

This field is currently not supported and shall be set to zero.

2.3.2.3.5 Frequency Band Field

The frequency band field of the node descriptor is five bits in length and specifies the frequency bands that are supported by the underlying IEEE 802.15.4 radio utilized by the node. For each frequency band supported by the underlying IEEE 802.15.4 radio, the corresponding bit of the frequency band field, as listed in Table 2.31, shall be set to 1. All other bits shall be set to 0.

Table 2.31 Values of the Frequency Band Field

Frequency Band Field Bit Number	Supported Frequency Band
0	868 – 868.6 MHz
1	Reserved
2	902 – 928 MHz
3	2400 – 2483.5 MHz
4	Reserved

2.3.2.3.6 MAC Capability Flags Field

The MAC capability flags field is eight bits in length and specifies the node capabilities, as required by the IEEE 802.15.4-2003 MAC sub-layer [B1]. The MAC capability flags field shall be formatted as illustrated in Figure 2.17.

Figure 2.17 Format of the MAC Capability Flags Field

Bits: 0	1	2	3	4-5	6	7
Alternate PAN coordinator	Device type	Power source	Receiver on when idle	Reserved	Security capability	Allocate address

The alternate PAN coordinator sub-field is one bit in length and shall be set to 1 if this node is capable of becoming a PAN coordinator. Otherwise, the alternative PAN coordinator sub-field shall be set to 0.

The device type sub-field is one bit in length and shall be set to 1 if this node is a full function device (FFD). Otherwise, the device type sub-field shall be set to 0, indicating a reduced function device (RFD).

The power source sub-field is one bit in length and shall be set to 1 if the current power source is mains power. Otherwise, the power source sub-field shall be set to 0. This information is derived from the node current power source field of the node power descriptor.

The receiver on when idle sub-field is one bit in length and shall be set to 1 if the device does not disable its receiver to conserve power during idle periods. Otherwise, the receiver on when idle sub-field shall be set to 0 (see also section 2.3.2.4.)

The security capability sub-field is one bit in length and shall be set to 1 if the device is capable of sending and receiving frames secured using the security suite specified in [B1]. Otherwise, the security capability sub-field shall be set to 0.

The allocate address sub-field is one bit in length and shall be set to 0 or 1.

2.3.2.3.7 Manufacturer Code Field

The manufacturer code field of the node descriptor is sixteen bits in length and specifies a manufacturer code that is allocated by the ZigBee Alliance, relating the manufacturer to the device.

2.3.2.3.8 Maximum Buffer Size Field

The maximum buffer size field of the node descriptor is eight bits in length, with a valid range of 0x00-0x7f. This field specifies the maximum size, in octets, of the network sub-layer data unit (NSDU) for this node. This is the maximum size of data or commands passed to or from the application by the application support sub-layer, before any fragmentation or re-assembly.

This field can be used as a high-level indication for network management.

2.3.2.3.9 Maximum Incoming Transfer Size Field

The maximum transfer size field of the node descriptor is sixteen bits in length, with a valid range of 0x0000-0x7fff. This field specifies the maximum size, in octets, of the application sub-layer data unit (ASDU) that can be transferred to this node in one single message transfer. This value can exceed the value of the node maximum buffer size field (see section 2.3.2.3.8) through the use of fragmentation.

2.3.2.3.10 Server Mask Field

The server mask field of the node descriptor is sixteen bits in length, with bit settings signifying the system server capabilities of this node. It is used to facilitate discovery of particular system servers by other nodes on the system. The bit settings are defined in Table 2.32.

Table 2.32 Server Mask Bit Assignments

Bit Number	Assignment
0	Primary Trust Center
1	Backup Trust Center
2	Primary Binding Table Cache
3	Backup Binding Table Cache
4	Primary Discovery Cache
5	Backup Discovery Cache
6	Network Manager
7 – 8	Reserved
9 – 15	Stack Compliance Revision

2.3.2.3.10.1 Stack Compliance Revision

These bits indicate the revision of the ZigBee Pro Core specification that the running stack is implemnted to. Prior to revision 21 of the specification these bits were reserved and thus set to 0. A stack that is compliant to revision 21 would set these bits to 21 (0010101b). A stack shall indicate the revision of the specification it is compliant to by setting these bits.

2.3.2.3.11 Maximum Outgoing Transfer Size Field

The maximum transfer size field of the node descriptor is sixteen bits in length, with a valid range of 0x0000-0x7fff. This field specifies the maximum size, in octets, of the application sub-layer data unit (ASDU) that can be transferred from this node in one single message transfer. This value can exceed the value of the node maximum buffer size field (see section 2.3.2.3.8) through the use of fragmentation.

2.3.2.3.12 Descriptor Capability Field

The descriptor capability field of the node descriptor is eight bits in length, with bit settings signifying the descriptor capabilities of this node. It is used to facilitate discovery of particular features of the descriptor fields by other nodes on the system. The bit settings are defined in Table 2.33.

Table 2.33 Descriptor Capability Bit Assignments

Bit Number	Assignment
0	Extended Active Endpoint List Available
1	Extended Simple Descriptor List Available

Bit Number	Assignment
2–7	Reserved

2.3.2.4 Node Power Descriptor

The node power descriptor gives a dynamic indication of the power status of the node and is mandatory for each node. There shall be only one node power descriptor in a node.

The fields of the node power descriptor are shown in Table 2.34 in the order of their transmission.

Table 2.34 Fields of the Node Power Descriptor

Field Name	Length (Bits)
Current power mode	4
Available power sources	4
Current power source	4
Current power source level	4

2.3.2.4.1 Current Power Mode Field

The current power mode field of the node power descriptor is four bits in length and specifies the current sleep/power-saving mode of the node. The current power mode field shall be set to one of the non-reserved values listed in Table 2.35.

Table 2.35 Values of the Current Power Mode Field

Current Power Mode Value $b_3b_2b_1b_0$	Description
0000	Receiver synchronized with the receiver on when idle subfield of the node descriptor.
0001	Receiver comes on periodically as defined by the node power descriptor.
0010	Receiver comes on when stimulated, for example, by a user pressing a button.
0011-1111	Reserved.

2.3.2.4.2 Available Power Sources Field

The available power sources field of the node power descriptor is four bits in length and specifies the power sources available on this node. For each power source supported on this node, the corresponding bit of the available power sources field, as listed in Table 2.36, shall be set to 1. All other bits shall be set to 0.

Table 2.36 Values of the Available Power Sources Field

Available Power Sources Field Bit Number	Supported Power Source
0	Constant (mains) power
1	Rechargeable battery
2	Disposable battery
3	Reserved

2.3.2.4.3 Current Power Source Field

The current power source field of the node power descriptor is four bits in length and specifies the current power source being utilized by the node. For the current power source selected, the corresponding bit of the current power source field, as listed in Table 2.37, shall be set to 1. All other bits shall be set to 0.

Table 2.37 Values of the Current Power Sources Field

Current Power Source Field Bit Number	Current Power Source
0	Constant (mains) power
1	Rechargeable battery
2	Disposable battery
3	Reserved

2.3.2.4.4 Current Power Source Level Field

The current power source level field of the node power descriptor is four bits in length and specifies the level of charge of the power source. The current power source level field shall be set to one of the non-reserved values listed in Table 2.38.

Table 2.38 Values of the Current Power Source Level Field

Current Power Source Level Field $b_3b_2b_1b_0$	**Charge Level**
0000	Critical
0100	33%
1000	66%
1100	100%
All other values	Reserved

2.3.2.5 Simple Descriptor

The simple descriptor contains information specific to each endpoint contained in this node. The simple descriptor is mandatory for each endpoint present in the node.

The fields of the simple descriptor are shown in Table 2.39 in their order of transmission. As this descriptor needs to be transmitted over air, the overall length of the simple descriptor shall be less than or equal to *apscMaxDescriptorSize*.

Table 2.39 Fields of the Simple Descriptor

Field Name	**Length (Bits)**
Endpoint	8
Application profile identifier	16
Application device identifier	16
Application device version	4
Reserved	4
Application input cluster count	8
Application input cluster list	16**i* (where *i* is the value of the application input cluster count)
Application output cluster count	8
Application output cluster list	16**o* (where *o* is the value of the application output cluster count)

2.3.2.5.1 Endpoint Field

The endpoint field of the simple descriptor is eight bits in length and specifies the endpoint within the node to which this description refers. Applications shall only use endpoints 1-254. Endpoints 241-254 shall be used only with the approval of the ZigBee Alliance. The Green Power cluster, if implemented, shall use endpoint 242.

2.3.2.5.2 Application Profile Identifier Field

The application profile identifier field of the simple descriptor is sixteen bits in length and specifies the profile that is supported on this endpoint. Profile identifiers shall be obtained from the ZigBee Alliance.

2.3.2.5.3 Application Device Identifier Field

The application device identifier field of the simple descriptor is sixteen bits in length and specifies the device description supported on this endpoint. Device description identifiers shall be obtained from the ZigBee Alliance.

2.3.2.5.4 Application Device Version Field

The application device version field of the simple descriptor is four bits in length and specifies the version of the device description supported on this endpoint. The application device version field shall be set to one of the non-reserved values listed in Table 2.40.

Table 2.40 Values of the Application Device Version Field

Application Device Version Value $b_3b_2b_1b_0$	Description
0000-1111	Specific values to be set by the application profile described by the application profile identifier in this descriptor. Default shall be 0000 unless otherwise defined by the application profile.

2.3.2.5.5 Application Input Cluster Count Field

The application input cluster count field of the simple descriptor is eight bits in length and specifies the number of input clusters, supported on this endpoint that will appear in the application input cluster list field. If the value of this field is zero, the application input cluster list field shall not be included.

2.3.2.5.6 Application Input Cluster List

The application input cluster list of the simple descriptor is 16**i* bits in length, where *i* is the value of the application input cluster count field. This field specifies the list of input clusters supported on this endpoint, for use during the service discovery and binding procedures.

The application input cluster list field shall be included only if the value of the application input cluster count field is greater than zero.

2.3.2.5.7 Application Output Cluster Count Field

The application output cluster count field of the simple descriptor is eight bits in length and specifies the number of output clusters, supported on this endpoint that will appear in the application output cluster list field. If the value of this field is zero, the application output cluster list field shall not be included.

2.3.2.5.8 Application Output Cluster List

The application output cluster list of the simple descriptor is 16**o* bits in length, where *o* is the value of the application output cluster count field. This field specifies the list of output clusters supported on this endpoint, for use during the service discovery and binding procedures.

The application output cluster list field shall be included only if the value of the application output cluster count field is greater than zero.

2.3.2.6 Complex Descriptor

The complex descriptor contains extended information for each of the device descriptions contained in this node. The use of the complex descriptor is optional.

Due to the extended and complex nature of the data in this descriptor, it is presented in XML form using compressed XML tags. Each field of the descriptor, shown in Table 2.41, can therefore be transmitted in any order. As this descriptor needs to be transmitted over air, the overall length of the complex descriptor shall be less than or equal to *apscMaxDescriptorSize*.

Table 2.41 Fields of the Complex Descriptor

Field Name	**XML Tag**	**Compressed XML Tag Value x_1x_0**	**Data Type**
Reserved	-	00	-
Language and character set	<languageChar>	01	See section 2.3.2.6.1
Manufacturer name	<manufacturerName>	02	Character string
Model name	<modelName>	03	Character string
Serial number	<serialNumber>	04	Character string
Device URL	<deviceURL>	05	Character string
Icon	<icon>	06	Octet string
Icon URL	<outliner>	07	Character string
Reserved	-	08 – ff	-

2.3.2.6.1 Language and Character Set Field

The language and character set field is three octets in length and specifies the language and character set used by the character strings in the complex descriptor. The format of the language and character set field is illustrated in Figure 2.18.

Figure 2.18 Format of the Language and Character Set Field

Octets: 2	**1**
ISO 639-1 language code	Character set identifier

The ISO 639-1 language code sub-field is two octets in length and specifies the language used for character strings, as defined in [B5].

The character set identifier sub-field is one octet in length and specifies the encoding used by the characters in the character set. This sub-field shall be set to one of the non-reserved values listed in Table 2.42.

Table 2.42 Values of the Character Set Identifier Sub-Field

Character Set Identifier Value	Bits Per Character	Description
0x00	8	ISO 646, ASCII character set. Each character is fitted into the least significant 7 bits of an octet with the most significant bit set to zero (see also [B6]).
0x01 – 0xff	-	Reserved.

If the language and character sets have not been specified, the language shall default to English (language code = “EN”) and the character set to ISO 646.

2.3.2.6.2 Manufacturer Name Field

The manufacturer name field has a variable length and contains a character string representing the name of the manufacturer of the device.

2.3.2.6.3 Model Name Field

The model name field has a variable length and contains a character string representing the name of the manufacturer’s model of the device.

2.3.2.6.4 Serial Number Field

The serial number field has a variable length and contains a character string representing the manufacturer’s serial number of the device.

2.3.2.6.5 Device URL Field

The device URL field has a variable length and contains a character string representing the URL through which more information relating to the device can be obtained.

2.3.2.6.6 Icon Field

The icon field has a variable length and contains an octet string which carries the data for an icon that can represent the device on a computer, gateway, or PDA. The format of the icon shall be a 32-by-32-pixel PNG image.

2.3.2.6.7 Icon URL Field

The icon URL field has a variable length and contains a character string representing the URL through which the icon for the device can be obtained.

2.3.2.7 User Descriptor

The user descriptor contains information that allows the user to identify the device using a user-friendly character string, such as “Bedroom TV” or “Stairs light”. The use of the user descriptor is optional. This descriptor contains a single field, which uses the ASCII character set, and shall contain a maximum of 16 characters.

The fields of the user descriptor are shown in Table 2.43 in the order of their transmission.

Table 2.43 Fields of the User Descriptor

Field Name	Length (Octets)
User description	16

2.3.3 Functional Description

2.3.3.1 Reception and Rejection

The application framework shall be able to filter frames arriving via the APS sub-layer data service and only present the frames that are of interest to the applications implemented on each active endpoint.

The application framework receives data from the APS sub-layer via the APSDE-DATA.indication primitive and is targeted at a specific endpoint (DstEndpoint parameter) and a specific profile (ProfileId parameter).

If the application framework receives a frame for an inactive endpoint, the frame shall be discarded. Otherwise, if the profile identifier passes the Profile Id Endpoint Matching Rules (see section 2.3.3.2), the application framework shall pass the payload of the received frame to the application implemented on the specified endpoint.

2.3.3.2 Profile ID Endpoint Matching Rules

Table 2.44 below details the matching of incoming APS datagrams or ZDO discovery messages are matched.

Table 2.44 Profile ID Endpoint Matching Rules

Incoming Message	APS or ZDO Profile ID	Destination Endpoint SimpleDescriptor ZDO	Legacy	Common	ZSE	GW	MSP	GP	ZLL
ZDO	0x0000	ZDO	x	x	x	x	x	x	x
Legacy	0x0101 – 0x0103, 0x0105 – 0x0108	x	Legacy	Legacy	x	x	x	x	x
Common (HA)	0x0104	x	x	Common	x	x	x	x	x
ZSE	0x0109	x	x	x	ZSE	x	x	x	x
Gateway (GW)	0x7F02	x	x	x	x	GW	x	x	x
MSP	0x8000 – 0xFF00, 0x7F01	x	x	x	x	x	MSP	x	x
GreenPower (GP)	0xA1E0	x	x	x	x	x	x	GP	x
ZLL	0xC05E	x	x	ZLL	x	x	x	x	ZLL
Wildcard	0xFFFF	ZDO	Legacy	Common	ZSE	GW	x	x	ZLL

2.3.3.2.1 Profile ID Endpoint Matching Rules for Incoming Messages

To apply Profile ID Endpoint matching rules for an incoming message, perform the following:

(1) Starting on the Left side of the table, find the row that matches the profile ID of the incoming message.

(2) If no match is found, then the message shall be dropped and no further processing shall take place.

(3) Lookup the Simple Descriptor of the local destination Endpoint.

(4) If no Simple Descriptor exists for the local destination endpoint, the message shall be dropped and no further processing shall be done.

(5) If a Simple Descriptor exists, follow across the selected row in the table to where the Profile ID at the top of the column matches the Profile ID of the simple descriptor of the destination endpoint.

(6) If an ‘x‘ appears in the selected row, then the message shall be dropped and no further processing shall take place.

(7) If a value other than ‘x‘ appears in the selected row, then the message shall be processed. The value in the cell indicates the Profile ID that shall be used for any response message generated. If a range of values exist (for example Legacy), then the exact value for the Profile ID on the incoming message may be used on any outgoing message generated.

For ZDO messages, the Profile ID Endpoint matching may be applied twice. The first time the rules will be applied on the message as a normal incoming APS datagram. For certain ZDO messages, the rules will be applied again to determine if the contents of the ZDO message match.

2.3.3.2.2 Profile ID Endpoint Matching Rules for ZDO Contents

To apply Profile ID Endpoint matching rules on the contents of ZDO discovery messages, perform the following:

(1) Starting on the left side of the table, find the row that matches the profile ID within the payload of the ZDO message (do not consider the Profile ID of the incoming ZDO message, which is always 0x0000).

(2) If no match is found, then there is no match for the discovery. Do the following:

  (a) Return an empty list of endpoints to the ZDO for processing. A response may be generated according to the rules of ZDO discovery. No further match processing on the message shall take place.

(3) If a match is found, lookup the Simple Descriptor for all local endpoints. For each simple descriptor, perform the following:

  (a) Follow the previously selected row across the table and find the column with a Profile ID that matches the Simple Descriptor.

  (b) If a column with a matching Profile ID does not exist, then there is no match. Continue processing on the next local endpoint.

  (c) If the Profile ID at the top of the column matches, examine the contents of the cell.

  (d) If an X is found in the cell, then there is no match. Continue processing on the next local endpoint.

  (e) If a value other than X is found in the table, then a match exists. Add the endpoint and the associated Profile ID of the simple descriptor to the list of matches.

(4) Once all endpoints have been analyzed, return the list of matching endpoints and the associated Profile IDs for each endpoint to the ZDO for processing.

2.4 The ZigBee Device Profile

2.4.1 Scope

This ZigBee Application Layer Specification describes how general ZigBee device features such as Binding, Device Discovery, and Service Discovery are implemented within ZigBee Device Objects. The ZigBee Device Profile operates like any ZigBee profile by defining clusters. Unlike application specific profiles, the clusters within the ZigBee Device Profile define capabilities supported in all ZigBee devices. As with any profile document, this document details the mandatory and/or optional clusters.

2.4.2 Device Profile Overview

The Device Profile supports four key inter-device communication functions within the ZigBee protocol. These functions are explained in the following sections:

- Device and Service Discovery Overview
- End Device Bind Overview
- Bind and Unbind Overview
- Binding Table Management Overview
- Network Management Overview

2.4.2.1 Device and Service Discovery Overview

Device and Service Discovery are distributed operations where individual devices or designated discovery cache devices respond to discovery requests. The “device address of interest” field enables responses from either the device itself or a discovery cache device. In selected cases where both the discovery cache device or the device’s parent and the “device address of interest” device respond, the response from the “device address of interest” shall be used.

The following capabilities exist for device and service discovery:

- **Device Discovery:** Provides the ability for a device to determine the identity of other devices on the PAN. Device Discovery is supported for both the 64-bit IEEE address and the 16-bit Network address.
    - Device Discovery messages can be used in one of two ways:
        - **Broadcast addressed:** All devices on the network shall respond according to the Logical Device Type and the matching criteria. ZigBee End Devices shall respond with just their address. ZigBee Coordinators and ZigBee Routers with associated devices shall respond with their address as the first entry followed by the addresses of their associated devices depending on the type of request. The responding devices shall employ APS acknowledged service on the unicast responses.
        - **Unicast addressed:** Only the specified device responds. A ZigBee End Device shall respond only with its address. A ZigBee Coordinator or Router shall reply with its own address and the address of each associated child device. Inclusion of the associated child devices allows the requestor to determine the network topology underlying the specified device.
- **Service Discovery:** Provides the ability for a device to determine services offered by other devices on the PAN.
    - Service Discovery messages can be used in one of two ways:
        - **Broadcast addressed:** Due to the volume of information that could be returned, only the individual device or the primary discovery cache shall respond with the matching criteria established in the request. The primary discovery cache shall only respond in this case if it holds cached discovery information for the NWKAddrOfInterest from the request. The responding devices shall also employ APS acknowledged service on the unicast responses.

- **Unicast addressed:** Only the specified device shall respond. In the case of a ZigBee Coordinator or ZigBee Router, these devices shall cache the Service Discovery information for sleeping associated devices and respond on their behalf.

  - Service Discovery is supported with the following query types:
    - **Active Endpoint:** This command permits an enquiring device to determine the active endpoints. An active endpoint is one with an application supporting a single profile, described by a Simple Descriptor. The command shall be unicast addressed.
    - **Match Simple Descriptor:** This command permits enquiring devices to supply a Profile ID (and, optionally, lists of input and/or output Cluster IDs) and ask for a return of the identity of an endpoint on the destination device which matches the supplied criteria. This command may be broadcast to all devices for which macRxOnWhenIdle = TRUE, or unicast addressed. For broadcast addressed requests, the responding device shall employ APS acknowledged service on the unicast responses.
    - **Simple Descriptor:** This command permits an enquiring device to return the Simple Descriptor for the supplied endpoint. This command shall be unicast addressed.
    - **Node Descriptor:** This command permits an enquiring device to return the Node Descriptor from the specified device. This command shall be unicast addressed.
    - **Power Descriptor:** This command permits an enquiring device to return the Power Descriptor from the specified device. This command shall be unicast addressed.
    - **Complex Descriptor:** This optional command permits an enquiring device to return the Complex Descriptor from the specified device. This command shall be unicast addressed.
    - **User Descriptor:** This optional command permits an enquiring device to return the User Descriptor from the specified device. This command shall be unicast addressed.

2.4.2.2 End Device Bind Overview

The following capabilities exist for end device bind:

- **End Device Bind:**
  - Provides the ability for an application to support a simplified method of binding where user intervention is employed to identify command/control device pairs. Typical usage would be where a user is asked to push buttons on two devices for installation purposes. Using this mechanism a second time allows the user to remove the binding table entry.

2.4.2.3 Bind and Unbind Overview

The following capabilities exist for directly configuring binding table entries:

- **Bind:** provides the ability for creation of a Binding Table entry that maps control messages to their intended destination.
- **Unbind:** provides the ability to remove Binding Table entries.

2.4.2.4 Binding Table Management Overview

The following capabilities exist for management of binding tables:

- Registering devices that implement source binding:
  - Provides the ability for a source device to instruct its primary binding table cache to hold its own binding table.
- Replacing a device with another wherever it occurs in the binding table:
  - Provides the ability to replace one device for another, by replacing all instances of its address in the binding table.
- Backing up a binding table entry:

  - Provides the ability for a primary binding table cache to send details of a newly created entry to the backup binding table cache (after receiving a bind request).
- Removing a backup binding table entry:
  - Provides the ability for a primary binding table cache to request that a specific entry be removed from the backup binding table cache (after receiving an unbind request).
- Backing up of the entire binding table:
  - Provides the ability for a primary binding table cache to request backup of its entire binding table, using the backup binding table cache.
- Restoring the entire binding table:
  - Provides the ability for a primary binding table cache to request restoration of its entire binding table, using the backup binding table cache.
- Backing up the Primary Binding Table Cache:
  - Provides the ability for a primary binding table cache to request backup of its entire source devices address table (which contains the addresses of any source device containing its own binding table).
- Restoring the Primary Binding Table Cache:
  - Provides the ability for a primary binding table cache to request restoration of its entire source devices address table (which contains the addresses of any source device containing its own binding table).

2.4.2.5 Network Management Overview

The following capabilities exist for network management:

- Provides the ability to retrieve management information from the devices including:
    - Network discovery results
    - Link quality to neighbor nodes
    - Routing table contents
    - Binding table contents
    - Discovery cache contents
    - Energy detection scan results
- Provides the ability to set management information controls including:
    - Network leave
    - Network direct join
    - Permit joining
    - Network update and fault notification

2.4.2.6 Device Descriptions for the Device Profile

The ZigBee Device Profile utilizes a single Device Description. Each cluster specified as Mandatory shall be present in all ZigBee devices. The response behavior to some messages is logical device type specific. The support for optional clusters is not dependent on the logical device type.

2.4.2.7 Configuration and Roles

The Device Profile assumes a client/server topology. A device making Device Discovery, Service Discovery, Binding or Network Management requests does so via a client role. A device which services these requests and responds does so via a server role. The client and server roles are non-exclusive in that a given device may supply both client and server roles.

Since many client requests and server responses are public and accessible to application objects other than ZigBee Device Objects, the Transaction Sequence number in the Application Framework header shall be the same on client requests and their associated server responses.

The Device Profile describes devices in one of two configurations:

- **Client:** A client issues requests to the server via Device Profile messages.
- **Server:** A server issues responses to the client that initiated the Device Profile message.

2.4.2.8 Transmission of ZDP Commands

All ZDP commands shall be transmitted via the APS data service and shall be formatted according to the ZDP frame structure, as illustrated in Figure 2.19.

Figure 2.19 Format of the ZDP Frame

Octets: 1	**Variable**
Transaction sequence number	Transaction data

2.4.2.8.1 Transaction Sequence Number Field

The transaction sequence number field is eight bits in length and specifies an identification number for the ZDP transaction so that a response command frame can be related to the request frame. The application object itself shall maintain an eight-bit counter that is copied into this field and incremented by one for each command sent. When a value of 0xff is reached, the next command shall restart the counter with a value of 0x00.

If a device sends a ZDP request command that requires a response, the target device shall respond with the relevant ZDP response command and include the transaction sequence number contained in the original request command.

The transaction sequence number field can be used by a controlling device, which may have issued multiple commands, so that it can match the incoming responses to the relevant command.

2.4.2.8.2 Transaction Data Field

The transaction data field has a variable length and contains the data for the individual ZDP transaction. The format and length of this field is dependent on the command being transmitted, as defined in sections 2.4.3 and 2.4.4.

2.4.3 Client Services

The Device Profile Client Services support the transport of device and service discovery requests, end device binding requests, bind requests, unbind requests, and network management requests from client to server. Additionally, Client Services support receipt of responses to these requests from the server.

2.4.3.1 Device and Service Discovery Client Services

Table 2.45 lists the commands supported by Device Profile, Device, and Service Discovery Client Services. Each of these commands will be discussed in the following sections.

Table 2.45 Device and Service Discovery Client Services Commands

Device and Service Discovery Client Services	Client Transmission	Server Processing
NWK_addr_req	O	M
IEEE_addr_req	O	M
Node_Desc_req	M	M
Power_Desc_req	O	M
Simple_Desc_req	O	M
Active_EP_req	O	M
Match_Desc_req	O	M
Complex_Desc_req	O	O
User_Desc_req	O	O

Device and Service Discovery Client Services	Client Transmission	Server Processing
Discovery_Cache_req	O	O
Device_annce	O	M
Parent_annce	M	M
Parent_annce_rsp	M	M
User_Desc_set	O	O
System_Server_Discover_req	O	O
Discovery_store_req	O	O
Node_Desc_store_req	O	O
Power_Desc_store_req	O	O
Active_EP_store_req	O	O
Simple_Desc_store_req	O	O
Remove_node_cache_req	O	O
Find_node_cache_req	O	O
Extended_Simple_Desc_req	O	O
Extended_Active_EP_req	O	O

2.4.3.1.1 NWK_addr_req

The NWK_addr_req command (ClusterID=0x0000) shall be formatted as illustrated in Figure 2.20.

Figure 2.20 Format of the NWK_addr_req Command

Octets: 8	1	1
IEEEAddress	RequestType	StartIndex

Table 2.46 specifies the fields of the NWK_addr_req Command Frame.

Table 2.46 Fields of the NWK_addr_req Command

Name	Type	Valid Range	Description
IEEEAddr	IEEE Address	A valid 64-bit IEEE address	The IEEE address to be matched by the Remote Device
RequestType	Integer	0x00-0xff	Request type for this command: 0x00 – Single device response 0x01 – Extended response 0x02-0xFF – reserved
StartIndex	Integer	0x00-0xff	If the Request type for this command is Extended response, the StartIndex provides the starting index for the requested elements of the associated devices list

2.4.3.1.1.1 When Generated

The NWK_addr_req is generated from a Local Device wishing to inquire as to the 16-bit address of the Remote Device based on its known IEEE address. The destination addressing on this command shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.

2.4.3.1.1.2 Effect on Receipt

Upon receipt, a Remote Device shall compare the IEEEAddr to its *nwkIeeeAddress* in the NIB or any IEEE address held in its *nwkNeighborTable* where the Device Type field of the entry is 0x02 (End Device). If there is no match and the request was unicast, a NWK_addr_resp command shall be generated and sent back to the local device with the Status field set to DEVICE_NOT_FOUND, the IEEEAddrRemoteDev field set to the IEEE address of the request; the NWKAddrRemoteDev field set to the NWK address of this device; and the NumAssocDev, StartIndex, and NWKAddrAssocDevList fields shall not be included in the frame. If there is no match and the command was received as a broadcast, the request shall be discarded and no response generated.

If a match is detected between the contained IEEEAddr and the receiving device's *nwkIeeeAddress* or one held in the receiving device's *nwkNeighborTable*, the RequestType shall be used to create a response. If the RequestType is one of the reserved values, a NWK_addr_resp command shall be generated and sent back to the local device with the Status field set to INV_REQUESTTYPE; the IEEEAddrRemoteDev field set to the IEEE address of the request; the NWKAddrRemoteDev field set to the network address corresponding to the IEEE address in the request; the NumAssocDev, StartIndex, and NWKAddrAssocDevList fields shall not be included in the frame.

If the RequestType is single device response, a NWK_addr_resp command shall be generated and sent back to the local device with the Status field set to SUCCESS, the IEEEAddrRemoteDev field set to the IEEE address of the request; the NWKAddrRemoteDev field set to the NWK address of the discovered device; and the NumAssocDev, StartIndex, and NWKAddrAssocDevList fields shall not be included in the frame.

If the RequestType was Extended response and the Remote Device is either the ZigBee coordinator or router, a NWK_addr_resp command shall be generated and sent back to the local device with the Status field set to SUCCESS, the IEEEAddrRemoteDev field set to the IEEE address of the device itself, and the NWKAddrRemoteDev field set to the NWK address of the device itself. The Remote Device shall also supply a list of all 16-bit NWK addresses in the NWKAddrAssocDevList field, starting with the entry StartIndex and continuing with whole entries until the maximum APS packet length is reached, for all devices in its *nwkNeighborTable* where the Device Type is 0x02 (End Device). It shall then set the NumAssocDev field to the number of entries in the NWKAddrAssocDevList field.

2.4.3.1.2 IEEE_addr_req

The IEEE_addr_req command (ClusterID=0x0001) shall be formatted as illustrated in Figure 2.21.

Figure 2.21 Format of the IEEE_addr_req_Command Frame

Octets: 2	**1**	**1**
NWKAddrOfInterest	RequestType	StartIndex

Table 2.47 specifies the fields of the IEEE_addr_req command frame.

Table 2.47 Fields of the IEEE_addr_req Command

Name	Type	Valid Range	Description
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address that is used for IEEE address mapping.
RequestType	Integer	0x00-0xff	Request type for this command: 0x00 – Single device response 0x01 – Extended response 0x02-0xff – reserved
StartIndex	Integer	0x00-0xff	If the Request type for this command is Extended response, the StartIndex provides the starting index for the requested elements of the associated devices list.

2.4.3.1.2.1 When Generated

The IEEE_addr_req is generated from a Local Device wishing to inquire as to the 64-bit IEEE address of the Remote Device based on their known 16-bit address. The destination addressing on this command shall be unicast.

2.4.3.1.2.2 Effect on Receipt

Upon receipt a Remote Device shall compare the NWKAddrOfInterest to its local *nwkNetworkAddress* value in the NIB, or compare any Network address field held in its *nwkNeighborTable* that also has the Device Type field set to 0x02 (End Device). If there is no match, an IEEE_addr_resp command shall be generated and sent back to the local device with the Status field set to DEVICE_NOT_FOUND; the IEEEAddrRemoteDev field set to the IEEE address of this device; the NWKAddrRemoteDev field set to the NWK address of the request; and the NumAssocDev, StartIndex, and NWKAddrAssocDevList fields shall not be included in the frame.

If a match is detected between the contained NWKAddrOfInterest and the receiving device's *nwkNetworkAddress* or one held in the *nwkNeighborTable*, the RequestType shall be used to create a response. If the RequestType is one of the reserved values, an IEEE_addr_resp command shall be generated and sent back to the local device with the Status field set to INV_REQUESTTYPE, the IEEEAddrRemoteDev field set to the IEEE address of this device, the NWKAddrRemoteDev field set to the network address of this device and the NumAssocDev, StartIndex, and NWKAddrAssocDevList fields shall not be included in the frame.

If the RequestType is single device response, an IEEE_addr_resp command shall be generated and sent back to the local device with the Status field set to SUCCESS, the IEEEAddrRemoteDev field set to the IEEE address of the discovered device, the NWKAddrRemoteDev field set to the NWK address of the request and the NumAssocDev, StartIndex, and NWKAddrAssocDevList fields shall not be included in the frame.

If the RequestType indicates an Extended Response and the Remote Device is the ZigBee coordinator or router with associated devices, an IEEE_addr_resp command shall be generated and sent back to the local device with the Status field set to SUCCESS, the IEEEAddrRemoteDev field set to the IEEE address of the device itself, and the NWKAddrRemoteDev field set to the NWK address of the device itself. The Remote Device shall also supply a list of all 16-bit network addresses in the NWKAddrAssocDevList field, starting with the entry StartIndex and continuing with whole entries until the maximum APS packet length is reached, for each entry in the *nwkNeighborTable* where the Device Type field is set to 0x02 (End Device). It shall then set the NumAssocDev field to the number of entries in the NWKAddrAssocDevList field.

2.4.3.1.3 Node_Desc_req

The Node_Desc_req_command (ClusterID=0x0002) shall be formatted as illustrated in Figure 2.22.

Figure 2.22 Format of the Node_Desc_req Command Frame

Octets: 2
NWKAddrOfInterest

Table 2.48 specifies the fields for the Node_Desc_req command frame.

Table 2.48 Fields of the Node_Desc_req Command

Name	**Type**	**Valid Range**	**Description**
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request

2.4.3.1.3.1 When Generated

The Node_Desc_req command is generated from a local device wishing to inquire as to the node descriptor of a remote device. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.

The local device shall generate the Node_Desc_req command using the format illustrated in Table 2.48. The NWKAddrOfInterest field shall contain the network address of the remote device for which the node descriptor is required.

2.4.3.1.3.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate a Node_Desc_rsp command in response, according to the description in section 2.4.4.2.3.1.

2.4.3.1.4 Power_Desc_req

The Power_Desc_req command (ClusterID=0x0003) shall be formatted as illustrated in Figure 2.23.

Figure 2.23 Format of the Power_Desc_req Command Frame

Octets: 2
NWKAddrOfInterest

Table 2.49 specifies the fields of the Power_Desc_req command frame.

Table 2.49 Fields of the Power_Desc_req Command

Name	Type	Valid Range	Description
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.

2.4.3.1.4.1 When Generated

The Power_Desc_req command is generated from a local device wishing to inquire as to the power descriptor of a remote device. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.

The local device shall generate the Power_Desc_req command using the format illustrated in Table 2.49. The NWKAddrOfInterest field shall contain the network address of the remote device for which the power descriptor is required.

2.4.3.1.4.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate a Power_Desc_rsp command in response according to the description in section 2.4.4.2.4.1.

2.4.3.1.5 Simple_Desc_req

The Simple_Desc_req command (ClusterID=0x0004) shall be formatted as illustrated in Figure 2.24.

Figure 2.24 Format of the Simple_Desc_req Command Frame

Octets: 2	1
NWKAddrOfInterest	EndPoint

Table 2.50 specifies the fields of the Simple_Desc_req command frame.

Table 2.50 Fields of the Simple_Desc_req Command

Name	Type	Valid Range	Description
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request
Endpoint	8 bits	1–254	The endpoint on the destination

2.4.3.1.5.1 When Generated

The Simple_Desc_req command is generated from a local device wishing to inquire as to the simple descriptor of a remote device on a specified endpoint. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.

The local device shall generate the Simple_Desc_req command using the format illustrated in Table 2.50. The NWKAddrOfInterest field shall contain the network address of the remote device for which the simple descriptor is required and the endpoint field shall contain the endpoint identifier from which to obtain the required simple descriptor.

2.4.3.1.5.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate a Simple_Desc_rsp command in response, according to the description in section 2.4.4.2.5.1.

2.4.3.1.6 Active_EP_req

The Active_EP_req command (ClusterID=0x0005) shall be formatted as illustrated in Figure 2.25.

Figure 2.25 Format of the Active_EP_req Command Frame

Octets: 2
NWKAddrOfInterest

Table 2.51 specifies the fields of the Active_EP_req command frame.

Table 2.51 Fields of the Active_EP_req Command

Name	Type	Valid Range	Description
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.

2.4.3.1.6.1 When Generated

The Active_EP_req command is generated from a local device wishing to acquire the list of endpoints on a remote device with simple descriptors. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.

The local device shall generate the Active_EP_req command using the format illustrated in Table 2.51. The NWKAddrOfInterest field shall contain the network address of the remote device for which the active endpoint list is required.

2.4.3.1.6.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate an Active_EP_rsp command in response, according to the description in section 2.4.4.2.6.1.

2.4.3.1.7 Match_Desc_req

The Match_Desc_req command (ClusterID=0x0006) shall be formatted as illustrated in Figure 2.26.

Figure 2.26 Format of the Match_Desc_req Command Frame

Octets: 2	**2**	**1**	**Variable**	**1**	**Variable**
NWKAddrOfInterest	ProfileID	NumInClusters	InClusterList	NumOutClusters	OutClusterList

Table 2.52 specifies the fields of the Match_Desc_req command frame.

Table 2.52 Fields of the Match_Desc_req Command

Name	**Type**	**Valid Range**	**Description**
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
ProfileID	Integer	0x0000-0xffff	Profile ID to be matched at the destination.
NumInClusters	Integer	0x00-0xff	The number of Input Clusters provided for matching within the InClusterList.
InClusterList	2 bytes * NumInClusters		List of Input ClusterIDs to be used for matching; the InClusterList is the desired list to be matched by the Remote Device (the elements of the InClusterList are the supported output clusters of the Local Device).
NumOutClusters	Integer	0x00-0xff	The number of Output Clusters provided for matching within OutClusterList.
OutClusterList	2 bytes * NumOutClusters		List of Output ClusterIDs to be used for matching; the OutClusterList is the desired list to be matched by the Remote Device (the elements of the OutClusterList are the supported input clusters of the Local Device).

2.4.3.1.7.1 When Generated

The Match_Desc_req command is generated from a local device wishing to find remote devices supporting a specific simple descriptor match criterion. This command shall either be broadcast to all devices for which macRxOnWhenIdle = TRUE, or unicast. If the command is unicast, it shall be directed either to the remote device itself or to an alternative device that contains the discovery information of the remote device.

The local device shall generate the Match_Desc_req command using the format illustrated in Table 2.52. The NWKAddrOfInterest field shall contain the network address indicating a broadcast to all devices for which macRxOnWhenIdle = TRUE (0xfffd) if the command is to be broadcast, or the network address of the remote device for which the match is required.

The remaining fields shall contain the required criterion for which the simple descriptor match is requested. The ProfileID field shall contain the identifier of the profile for which the match is being sought or the wildcard profile ID of 0xFFFF.

The NumInClusters field shall contain the number of elements in the InClusterList field. If the value of this field is 0, the InClusterList field shall not be included. If the value of the NumInClusters field is not equal to 0, the InClusterList field shall contain the list of input cluster identifiers for which the match is being sought.

The NumOutClusters field shall contain the number of elements in the OutClusterList field. If the value of this field is 0, the OutClusterList field shall not be included. If the value of the NumOutClusters field is not equal to 0, the OutClusterList field shall contain the list of output cluster identifiers for which the match is being sought.

2.4.3.1.7.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate a Match_Desc_rsp command in response, according to the description in section 2.4.4.2.7.1.

2.4.3.1.8 Complex_Desc_req

The Complex_Desc_req command (ClusterID=0x0010) shall be formatted as illustrated in Figure 2.27.

Figure 2.27 Format of the Complex_Desc_req Command Frame

Octets: 2
NWKAddrOfInterest

Table 2.53 specifies the fields of the Complex_Desc_req command frame.

Table 2.53 Fields of the Complex_Desc_req Command

Name	Type	Valid Range	Description
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request

2.4.3.1.8.1 When Generated

The Complex_Desc_req command is generated from a local device wishing to inquire as to the complex descriptor of a remote device. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.

The local device shall generate the Complex_Desc_req command using the format illustrated in Table 2.53. The NWKAddrOfInterest field shall contain the network address of the remote device for which the complex descriptor is required.

2.4.3.1.8.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate a Complex_Desc_rsp command in response, according to the description in section 2.4.4.2.8.1.

2.4.3.1.9 User_Desc_req

The User_Desc_req (ClusterID=0x0011) command shall be formatted as illustrated in Figure 2.28.

Figure 2.28 Format of the User_Desc_req Command Frame

Octets: 2
NWKAddrOfInterest

Table 2.54 specifies the fields of the User_Desc_req command frame.

Table 2.54 Fields of the User_Desc_req Command

Name	Type	Valid Range	Description
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.

2.4.3.1.9.1 When Generated

The User_Desc_req command is generated from a local device wishing to inquire as to the user descriptor of a remote device. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.

The local device shall generate the User_Desc_req command using the format illustrated in Table 2.54. The NWKAddrOfInterest field shall contain the network address of the remote device for which the user descriptor is required.

2.4.3.1.9.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate a User_Desc_rsp command in response, according to the description in section 2.4.4.2.9.1.

2.4.3.1.10 Discovery_Cache_req

The Discovery_Cache_req command (ClusterID=0x0012) shall be formatted as illustrated in Figure 2.29.

Figure 2.29 Format of the Discovery_Cache_req Command Frame

Octets: 2	8
NWKAddr	IEEEAddr

Table 2.55 specifies the parameters for the Discovery_Cache_req command frame.

Table 2.55 Fields of the Discovery_Cache_req Command

Name	Type	Valid Range	Description
NWKAddr	Device Address	16-bit NWK address	NWK address for the Local Device.
IEEEAddr	Device Address	64-bit IEEE address	IEEE address for the Local Device.

2.4.3.1.10.1 When Generated

The Discovery_Cache_req is provided to enable devices on the network to locate a Primary Discovery Cache device on the network. The destination addressing on this primitive shall be broadcast to all devices for which macRxOnWhenIdle = TRUE.

2.4.3.1.10.2 Effect on Receipt

Upon receipt, if the Remote Device does not support the Discovery_Cache_req, the request shall be dropped and no further processing performed. If the Discovery_Cache_req is supported, the Remote Device shall create a unicast Discovery_Cache_rsp message to the source indicated by the Discovery_Cache_req and include a SUCCESS status.

2.4.3.1.11 Device_annce

The Device_annce command (ClusterID=0x0013) shall be formatted as illustrated in Figure 2.30.

Figure 2.30 Format of the Device_annce Command Frame

Octets: 2	8	1
NWKAddr	IEEEAddr	Capability

Table 2.56 specifies the fields of the Device_annce command frame.

Table 2.56 Fields of the Device_annce Command

Name	Type	Valid Range	Description
NWKAddr	Device Address	16-bit NWK address	NWK address for the Local Device
IEEEAddr	Device Address	64-bit IEEE address	IEEE address for the Local Device
Capability	Bitmap	See Figure 2.17	Capability of the local device

2.4.3.1.11.1 When Generated

The Device_annce is provided to enable ZigBee devices on the network to notify other ZigBee devices that the device has joined or re-joined the network, identifying the device’s 64-bit IEEE address and new 16-bit NWK address, and informing the Remote Devices of the capability of the ZigBee device. This command shall be invoked for all ZigBee end devices upon join or rejoin. This command may also be invoked by ZigBee routers upon join or rejoin as part of NWK address conflict resolution. The destination addressing on this primitive is broadcast to all devices for which macRxOnWhenIdle = TRUE.

2.4.3.1.11.2 Effect on Receipt

Upon receipt, the Remote Device shall use the IEEEAddr in the message to find a match with any other IEEE address held in the Remote Device. If a match is detected, the Remote Device shall update the nwkAddressMap attribute of the NIB with the updated NWKAddr corresponding to the IEEEAddr received.

The Remote Device shall also use the NWKAddr in the message to find a match with any other 16-bit NWK address held in the Remote Device, even if the IEEEAddr field in the message carries the value of 0xffffffffffffffff. If a match is detected for a device with an IEEE address other than that indicated in the IEEEAddr field received, then this entry shall be marked as not having a known valid 16-bit NWK address.

2.4.3.1.12 Parent_annce

The Parent_annce command (ClusterID = 0x001F) shall be formatted as illustrated in Figure 2.31.

Figure 2.31 Format of the Parent Annce Message

Octets: 1	**Variable**	**…**	**Variable**
NumberOfChildren	ChildInfo[0]	…	ChildInfo[n]

Table 2.57specifies the contents of the ChildInfo structure.

Table 2.57 - Format of the ChildInfo Structure

Name	Type	Description
Extended Address	64-bit IEEE address	The IEEE address of the child bound to the parent.

2.4.3.1.12.1 When Generated

The Parent_annce is provided to enable ZigBee routers (including the coordinator) on the network to notify other ZigBee routers about all the end devices known to the local device. This command provides a means to resolve conflicts more quickly than aging out the child, when multiple routers purport to be the active parent of a particular end-device. The command may be broadcast from one router to all routers and the coordinator using the broadcast address 0xFFFC or unicast from one router to another router.

This message must be generated if all the following conditions are met:

1. The router or coordinator device has rebooted.
2. The router or coordinator is operating in the joined and authenticated state.

The message generated under the above circumstances must be broadcast. Before broadcasting a Parent_annce message, the device shall start a countdown timer, *apsParentAnnounceTimer* equal to apsParentAnnounceBaseTimer + a random value from 0 to apsParentAnnounceJitterMax.

When the timer expires, a router shall examine its neighbor table for all devices. The router shall construct, but not yet send, an empty Parent_annce message and set NumberOfChildren to 0. For each end device in the neighbor table, it shall do the following.

1. If the Neighbor Table entry indicates a Device Type not equal to End Device (0x02), do not process this entry. Continue to the next one.
2. Incorporate end device information into the Parent_annce message by doing the following:
    a. Append a ChildInfo structure to the message.
    b. Increment NumberOfChildren by 1.
3. Note: The value of Keepalive Received for the Neighbor Table Entry is not considered.

After processing all entries in the neighbor table, if the NumberOfChildren is greater than 0, then it shall send the message to the all routers broadcast address (0xFFFC). If NumberOfChildren is 0, it shall discard the previously constructed Parent_annce message and not send it.

If the device has more ChildInfo entries than fit in a single message, it shall send additional messages. Each additional message needed shall trigger the device to calculate and start a new apsParentAnnounceTimer equal to apsParentAnnounceBaseTimer + a random value from 0 to apsParentAnnounceJitterMax. The local device shall wait until that timer expires before sending each additional message. . The NumberOfChildren for each message shall be set according to the number of ChildInfo entries contained within the message.

If the device must send multiple Parent_annce message but receives a keepalive from an end device before it has sent the Parent_Annce message, it shall not include that device in the message.

2.4.3.1.12.2 Effect on receipt

If the message is received by an end device, it shall be dropped. No further processing shall be done.

Upon receipt of a broadcast Parent_annce, if the local device has a non-zero value for its apsParentAnnounceTimer it shall immediately re-calculate a new value and start a new countdown. The apsParentAnnounceTimer shall be set to apsParentAnnounceBaseTimer + a random value from 0 to apsParentAnnounceJitterMax. It shall continue processing the message.

A router shall construct, but not yet send, an empty Parent_Annce_Rsp message with NumberOfChildren set to 0. It shall examine each Extended Address present in the message and search its Neighbor Table for an Extended Address entry that matches. For each match, process as follows:

1. If the Device Type is Zigbee End Device (0x02) and the Keepalive Received value is TRUE, do the following:
    a. It shall append to the Parent_annce_rsp frame the ChildInfo structure.
    b. Increment the NumberOfChildren by 1.
2. If the Device Type is not ZigBee End Device (0x02) or the Keepalive Received value is FALSE, do not process any further. Continue to the next entry.

If the NumberOfChildren field value is 0, the local device shall discard the previously constructed Parent_Annce_rsp. No response message shall be sent.

If the NumberOfChildren field in the Parent_Annce_rsp is greater than 0, it shall unicast the message to the sender of the Parent_Annce message.

If the device has more ChildInfo entries than fit in a single message, it shall send additional messages. These messages do not have to be jittered or delayed since they are unicast to a single device. Each Parent_annce_rsp shall set the NumberOfChildren field to the number of entries contained within the message.

2.4.3.1.13 User_Desc_set

The User_Desc_set command (ClusterID=0x0014) shall be formatted as illustrated in Figure 2.32.

Figure 2.32 Format of the User_Desc_set Command Frame

Octets: 2	1	Various
NWKAddrOfInterest	Length	UserDescriptor

Table 2.58 specifies the fields of the User_Desc_set command frame.

Table 2.58 Fields of the User_Desc_set Command

Name	Type	Valid Range	Description
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
Length	Integer	0x00 - 0x10	Length of the User Descriptor in bytes.
UserDescription	User Descriptor		The user description to configure; if the ASCII character string to be entered here is less than 16 characters in length, it shall be padded with space characters (0x20) to make a total length of 16 characters. Characters with codes 0x00-0x1f are not permitted.

2.4.3.1.13.1 When Generated

The User_Desc_set command is generated from a local device wishing to configure the user descriptor on a remote device. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.

The local device shall generate the User_Desc_set command using the format illustrated in Table 2.58. The NWKAddrOfInterest field shall contain the network address of the remote device for which the user descriptor is to be configured and the UserDescription field shall contain the ASCII character string that is to be configured in the user descriptor. Characters with ASCII codes numbered 0x00 through 0x1f are not permitted to be included in this string.

2.4.3.1.13.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate a User_Desc_conf command in response, according to the description in section 2.4.4.2.11.1.

2.4.3.1.14 System_Server_Discovery_req

The System_Server_Discovery_req command (ClusterID=0x0015) shall be formatted as illustrated in Figure 2.33.

Figure 2.33 Format of the System_Server_Discovery_req Command Frame

Octets: 2
ServerMask

Table 2.59 specifies the fields of the System_Server_Discovery_req command frame.

Table 2.59 Fields of the System_Server_Discovery_req Command

Name	Type	Valid Range	Description
ServerMask	Bitmap	16 bits	See Table 2.32 for bit assignments

2.4.3.1.14.1 When Generated

The System_Server_Discovery_req is generated from a Local Device wishing to discover the location of a particular system server or servers as indicated by the ServerMask parameter. The destination addressing on this request is 'broadcast to all devices for which macRxOnWhenIdle = TRUE.'

2.4.3.1.14.2 Effect on Receipt

Upon receipt, remote devices shall compare the ServerMask parameter to the Server Mask field in their own Node descriptor. If no bits are found to match, no action is taken. If any matching bits are found, the remote device shall send a System_Server_Discovery_rsp back to the originator using unicast transmission (with acknowledgement request) and indicating the matching bits.

2.4.3.1.15 Discovery_store_req

The Discovery_Store_req command (ClusterID=0x0016) shall be formatted as illustrated in Figure 2.34.

Figure 2.34 Format of the Discovery_Store_req Command Frame

Octets: 2	8	1	1	1	1	Variable
NWKAddr	IEEEAddr	NodeDescSize	PowerDescSize	ActiveEPSize	Simple DescCount	Simple DescSizeList

Table 2.60 specifies the fields of the Discovery_store_req command frame.

Table 2.60 Fields of the Discovery_store_req Command

Name	Type	Valid Range	Description
NWKAddr	Device Address	16-bit NWK Address	NWK Address for the Local Device.
IEEEAddr	Device Address	64-bit IEEE Address	IEEE Address for the Local Device.

Name	Type	Valid Range	Description
NodeDescSize	Integer	0x00-0xff	Size in bytes of the Node Descriptor for the Local Device.
PowerDescSize	Integer	0x00 - 0xff	Size in bytes of the Power Descriptor for the Local Device.
ActiveEPSize	Integer	0x00 - 0xff	Size in bytes of the ActiveEPCount and ActiveEPList fields of the Active_EP_rsp for the Local Device.
SimpleDescCount	Integer	0x00 - 0xff	Number of Simple Descriptors supported by the Local Device (should be the same value as the ActiveEPSize).
SimpleDescSizeList	Array of bytes		List of bytes of SimpleDescCount length, each of which represents the size in bytes of the Simple Descriptor for each Active Endpoint on the Local Device.

2.4.3.1.15.1 When Generated

The Discovery_store_req is provided to enable ZigBee end devices on the network to request storage of their discovery cache information on a Primary Discovery Cache device. Included in the request is the amount of storage space the Local Device requires.

The destination addressing on this request is unicast.

2.4.3.1.15.2 Effect on Receipt

Upon receipt, the Remote Device shall determine whether it is a Primary Discovery Cache device. If it is not a Primary Discovery Cache device, the Remote Device shall return a status of NOT_SUPPORTED. Next, the Remote Device shall determine whether it has storage for the requested discovery cache size determined by summing the sizes of the NWKAddr and IEEEAddr plus the NodeDescSize, PowerDescSize, ActiveEPSize, and the sizes from the SimpleDescSizeList. If sufficient space exists, the Local Device shall be provided a SUCCESS status. Otherwise, the Local Device shall return INSUFFICIENT_SPACE. If a SUCCESS status is returned, the Remote Device shall reserve the storage requested for the upload of the discovery information from the Local Device. Additionally, if the Local Device supplies an IEEEAddr which matches a previously stored entry, but the NWKAddr differs from the previous entry, the Remote Device shall remove the previous entry and discovery cache information in favor of the newly registered data.

2.4.3.1.16 Node_Desc_store_req

The Node_Desc_store_req command (ClusterID=0x0017) shall be formatted as illustrated in Figure 2.35.

Figure 2.35 Format of the Node_Desc_store_req Command Frame

Octets: 2	8	Variable
NWKAddr	IEEEAddr	NodeDescriptor

Table 2.61 specifies the fields of the Node_Desc_store_req command frame.

Table 2.61 Fields of the Node_Desc_store_req Command

Name	Type	Valid Range	Description
NWKAddr	Device Address	16-bit NWK Address	NWK Address for the Local Device
IEEEAddr	Device Address	64-bit IEEE Address	IEEE address for the Local Device
NodeDescriptor	Node Descriptor		See the Node Descriptor format in section 2.3.2.3

2.4.3.1.16.1 When Generated

The Node_Desc_store_req is provided to enable ZigBee end devices on the network to request storage of their Node Descriptor on a Primary Discovery Cache device which has previously received a SUCCESS status from a Discovery_store_req to the same Primary Discovery Cache device. Included in this request is the Node Descriptor the Local Device wishes to cache.

2.4.3.1.16.2 Effect on Receipt

Upon receipt, the Remote Device shall determine whether it is a Primary Discovery Cache device. If it is not a Primary Discovery Cache device, the Remote Device shall return a status of NOT_SUPPORTED. Next, the Remote Device shall determine whether it has previously processed a Discovery_store_req for the Local Device and returned a status of SUCCESS. If a previous Discovery_store_req has not been processed with a SUCCESS status, the Remote Device shall return NOT_PERMITTED. Next, the Remote Device shall determine if enough space is available to store the Node Descriptor for the Local Device. If not, the Remote Device shall return INSUFFICIENT_SPACE. Finally, the Remote Device shall store the Node Descriptor for the Local Device and return SUCCESS. If the request returned a status of SUCCESS and the NWKAddr and IEEEAddr in the request referred to addresses already held in the Primary Discovery Cache, the descriptor in this request shall overwrite the previously held entry.

2.4.3.1.17 Power_Desc_store_req

The Power_Desc_store_req command (ClusterID=0x0018) shall be formatted as illustrated in Figure 2.36.

Figure 2.36 Format of the Power_Desc_store_req Command Frame

Octets: 2	8	Variable
NWKAddr	IEEEAddr	PowerDescriptor

Table 2.62 specifies the fields of the Power_Desc_store_req command frame.

Table 2.62 Fields of the Power_Desc_store_req Command

Name	Type	Valid Range	Description
NWKAddr	Device Address	16-bit NWK Address	NWK Address for the Local Device.
IEEEAddr	Device Address	64-bit Address	IEEE address for the Local Device.

Name	Type	Valid Range	Description
PowerDescriptor	Power Descriptor		See the Power Descriptor format in section 2.3.2.4; This field shall only be included in the frame if the status field is equal to SUCCESS.

2.4.3.1.17.1 When Generated

The Power_Desc_store_req is provided to enable ZigBee end devices on the network to request storage of their Power Descriptor on a Primary Discovery Cache device which has previously received a SUCCESS status from a Discovery_store_req to the same Primary Discovery Cache device. Included in this request is the Power Descriptor the Local Device wishes to cache.

2.4.3.1.17.2 Effect on Receipt

Upon receipt, the Remote Device shall determine whether it is a Primary Discovery Cache device. If it is not a Primary Discovery Cache device, the Remote Device shall return a status of NOT_SUPPORTED. Next, the Remote Device shall determine whether it has previously processed a Discovery_store_req for the Local Device and returned a status of SUCCESS. If a previous Discovery_store_req has not been processed with a SUCCESS status, the Remote Device shall return NOT_PERMITTED. Next, the Remote Device shall determine if enough space is available to store the Power Descriptor for the Local Device. If not, the Remote Device shall return INSUFFICIENT_SPACE. Finally, the Remote Device shall store the Power Descriptor for the Local Device and return SUCCESS. If the request returned a status of SUCCESS, and the NWKAddr and IEEEAddr in the request referred to addresses already held in the Primary Discovery Cache, the descriptor in this request shall overwrite the previously held entry.

2.4.3.1.18 Active_EP_store_req

The Active_EP_store_req command (ClusterID=0x0019) shall be formatted as illustrated in Figure 2.37.

Figure 2.37 Format of the Active_EP_store_req Command Frame

Octets: 2	8	1	Variable
NWKAddr	IEEEAddr	ActiveEPCount	ActiveEPList

Table 2.63 specifies the fields of the Active_EP_store_req command frame.

Table 2.63 Fields of the Active_EP_store_req Command

Name	Type	Valid Range	Description
NWKAddr	Device Address	16-bit NWK Address	NWK Address for the Local Device.
IEEEAddr	Device Address	64-bit IEEE Address	IEEE Address for the Local Device.
ActiveEPCount	Integer	0x00-0xff	The count of active endpoints on the Local Device.

Name	Type	Valid Range	Description
ActiveEPList			List of bytes, each of which represents an 8-bit endpoint number.

2.4.3.1.18.1 When Generated

The Active_EP_store_req is provided to enable ZigBee end devices on the network to request storage of their list of Active Endpoints on a Primary Discovery Cache device which has previously received a SUCCESS status from a Discovery_store_req to the same Primary Discovery Cache device. Included in this request is the count of Active Endpoints the Local Device wishes to cache and the endpoint list itself.

2.4.3.1.18.2 Effect on Receipt

Upon receipt, the Remote Device shall determine whether it is a Primary Discovery Cache device. If it is not a Primary Discovery Cache device, the Remote Device shall return a status of NOT_SUPPORTED. Next, the Remote Device shall determine whether it has previously processed a Discovery_store_req for the Local Device and returned a status of SUCCESS. If a previous Discovery_store_req has not been processed with a SUCCESS status, the Remote Device shall return NOT_PERMITTED. Next, the Remote Device shall determine if enough space is available to store the Active Endpoint count and list for the Local Device. If not, the Remote Device shall return INSUFFICIENT_SPACE. Finally, the Remote Device shall store the Active Endpoint count and list for the Local Device and return SUCCESS. If the request returned a status of

SUCCESS, and the NWKAddr and the IEEEAddr in the request referred to addresses already held in the Primary Discovery Cache, the descriptor in this request shall overwrite the previously held entry.

2.4.3.1.19 Simple_Desc_store_req

The Simple_Desc_store_req command (ClusterID=0x001a) shall be formatted as illustrated in Figure 2.38.

Figure 2.38 Format of the Simple_Desc_store_req Command Frame

Octets: 2	8	1	Variable
NWKAddr	IEEEAddr	Length	SimpleDescriptor

Table 2.64 specifies the fields of the Simple_Desc_store_req command frame.

Table 2.64 Fields of the Simple_Desc_store_req Command

Name	Type	Valid Range	Description
NWKAddr	Device Address	16-bit NWK Address	NWK Address for the Local Device.
IEEEAddr	Device Address	64-bit IEEE Address	IEEE Address for the Local Device.
Length	Device Address	0x00 - 0xff	The length in bytes of the Simple Descriptor to follow.

Name	Type	Valid Range	Description
SimpleDescriptor	Simple Descriptor		See the Simple Descriptor format in section 2.3.2.5.

2.4.3.1.19.1 When Generated

The Simple_desc_store_req is provided to enable ZigBee end devices on the network to request storage of their list of Simple Descriptors on a Primary Discovery Cache device which has previously received a SUCCESS status from a Discovery_store_req to the same Primary Discovery Cache device. Note that each Simple Descriptor for every active endpoint on the Local Device must be individually uploaded to the Primary Discovery Cache device via this command to enable cached discovery. Included in this request is the length of the Simple Descriptor the Local Device wishes to cache and the Simple Descriptor itself. The endpoint is a field within the Simple Descriptor and is accessed by the Remote Device to manage the discovery cache information for the Local Device.

2.4.3.1.19.2 Effect on Receipt

Upon receipt, the Remote Device shall determine whether it is a Primary Discovery Cache device. If it is not a Primary Discovery Cache device, the Remote Device shall return a status of NOT_SUPPORTED. Next, the Remote Device shall determine whether it has previously processed a Discovery_store_req for the Local Device and returned a status of SUCCESS. If a previous Discovery_store_req has not been processed with a SUCCESS status, the Remote Device shall return NOT_PERMITTED. Next, the Remote Device shall determine if enough space is available to store the Simple Descriptor for the Local Device. If not, the Remote Device shall return INSUFFICIENT_SPACE. Finally, the Remote Device shall store the Simple Descriptor for the Local Device and return SUCCESS. If the request returned a status of SUCCESS and the NWKAddr and the IEEEAddr in the request referred to addresses already held in the Primary Discovery Cache, the descriptor in this request shall overwrite the previously held entry.

2.4.3.1.20 Remove_node_cache_req

The Remove_node_cache_req command (ClusterID=0x001b) shall be formatted as illustrated in Figure 2.39.

Figure 2.39 Format of the Remove_node_cache_req Command Frame

Octets: 2	8
NWKAddr	IEEEAddr

Table 2.65 specifies the fields of the Remove_node_cache_req command frame.

Table 2.65 Fields of the Remove_node_cache_req Command

Name	Type	Valid Range	Description
NWKAddr	Device Address	16-bit NWK Address	NWK Address for the device of interest.
IEEEAddr	Device Address	64-bit IEEE Address	IEEE Address for the device of interest.

2.4.3.1.20.1 When Generated

The Remove_node_cache_req is provided to enable ZigBee devices on the network to request removal of discovery cache information for a specified ZigBee end device from a Primary Discovery Cache device. The effect of a successful Remove_node_cache_req is to undo a previously successful Discovery_store_req and additionally remove any cache information stored on behalf of the specified ZigBee end device on the Primary Discovery Cache device.

2.4.3.1.20.2 Effect on Receipt

Upon receipt, the Remote Device shall determine whether it is a Primary Discovery Cache device. If it is not a Primary Discovery Cache device, the Remote Device shall return a status of NOT_SUPPORTED. Next, the Remote Device shall determine whether it has previously processed a Discovery_store_req for the indicated device and returned a status of SUCCESS. If a previous Discovery_store_req has not been processed with a SUCCESS status, the Remote Device shall return DEVICE_NOT_FOUND. Finally, the Remote Device shall remove all cached discovery information for the device of interest and return SUCCESS to the Local Device.

2.4.3.1.21 Find_node_cache_req

The Find_node_cache_req command (ClusterID=0x001c) shall be formatted as illustrated in Figure 2.40.

Figure 2.40 Format of the Find_node_cache Command Frame

Octets: 2	8
NWKAddr	IEEEAddr

Table 2.66 specifies the fields of the Find_node_cache_req command frame.

Table 2.66 Fields of the Find_node_cache_req Command Frame

Name	Type	Valid Range	Description
NWKAddr	Device Address	16-bit NWK Address	NWK Address for the device of interest.
IEEEAddr	Device Address	64-bit IEEE Address	IEEE Address for the device of interest.

2.4.3.1.21.1 When Generated

The Find_node_cache_req is provided to enable ZigBee devices on the network to broadcast to all devices for which macRxOnWhenIdle = TRUE a request to find a device on the network that holds discovery information for the device of interest, as specified in the request parameters. The effect of a successful Find_node_cache_req is to have the Primary Discovery Cache device, holding discovery information for the device of interest, unicast a Find_node_cache_rsp back to the Local Device. Note that, like the NWK_addr_req, only the device meeting this criteria shall respond to the request generated by Find_node_cache_req.

2.4.3.1.21.2 Effect on Receipt

Upon receipt, the Remote Device shall determine whether it is the device of interest or a Primary Discovery Cache device, and if so, if it holds discovery cache information for the device of interest. If it is not the device of interest or a Primary Discovery Cache device, and does not hold discovery cache information for the device of interest, the Remote Device shall cease processing the request and not supply a response. If the Remote Device is the device of interest, or a Primary Discovery Cache device, and, if the device holds discovery information for the indicated device of interest, the Remote Device shall return the NWKAddr and IEEEaddr for the device of interest.

2.4.3.1.22 Extended_Simple_Desc_req

The Extended_Simple_Desc_req command (ClusterID=0x001d) shall be formatted as illustrated in Figure 2.41.

Figure 2.41 Format of the Extended_Simple_Desc_req Command Frame

Octets: 2	1	1
NWKAddrOfInterest	EndPoint	StartIndex

Table 2.67 specifies the fields of the Extended_Simple_Desc_req command frame.

Table 2.67 Fields of the Extended_Simple_Desc_req Command

Name	Type	Valid Range	Description
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
Endpoint	8 bits	1-254	The endpoint on the destination.
StartIndex	8 bits	0x00-0xff	Starting index within the cluster list of the response represented by an ordered list of the Application Input Cluster List and Application Output Cluster List.

2.4.3.1.22.1 When Generated

The Extended_Simple_Desc_req command is generated from a local device wishing to inquire as to the simple descriptor of a remote device on a specified endpoint. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device. The Extended_Simple_Desc_req is intended for use with devices which employ a larger number of application input or output clusters than can be described by the Simple_Desc_req.

The local device shall generate the Extended_Simple_Desc_req command using the format illustrated in Table 2.67. The NWKAddrOfInterest field shall contain the network address of the remote device for which the simple descriptor is required and the endpoint field shall contain the endpoint identifier from which to obtain the required simple descriptor. The StartIndex is the first entry requested in the Application Input Cluster List and Application Output Cluster List sequence within the resulting response.

2.4.3.1.22.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate an Extended_Simple_Desc_rsp command in response, according to the description in section 2.4.4.2.20.1.

The results in the Extended_Simple_Desc_rsp shall include the elements described in Table 2.111 with a selectable set of the application input cluster and application output cluster lists starting with the entry StartIndex and continuing with whole entries until the maximum APS packet length is reached, along with a status of SUCCESS.

2.4.3.1.23 Extended_Active_EP_req

The Extended_Active_EP_req command (ClusterID=0x001e) shall be formatted as illustrated in Figure 2.42.

Figure 2.42 Format of the Extended_Active_EP_req Command Frame

Octets: 2	1
NWKAddrOfInterest	StartIndex

Table 2.68 specifies the fields of the Extended_Active_EP_req command frame.

Table 2.68 Fields of the Extended_Active_EP_req Command

Name	Type	Valid Range	Description
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
StartIndex	8 bits	0x00-0xff	Starting index within the Active Endpoint list in the response.

2.4.3.1.23.1 When Generated

The Extended_Active_EP_req command is generated from a local device wishing to acquire the list of endpoints on a remote device with simple descriptors. This command shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device. The Extended_Active_EP_req is used for devices which support more active endpoints than can be returned by a single Active_EP_req.

The local device shall generate the Extended_Active_EP_req command using the format illustrated in Table 2.68. The NWKAddrOfInterest field shall contain the network address of the remote device for which the active endpoint list is required. The StartIndex field shall be set in the request to enable retrieval of lists of active endpoints from devices whose list exceeds the size of a single ASDU and where fragmentation is not supported.

2.4.3.1.23.2 Effect on Receipt

Upon receipt of this command, the recipient device shall process the command and generate an Extended_Active_EP_rsp command in response, according to the description in section 2.4.4.2.21.1.

The results in the Extended_Active_EP_rsp shall include the elements described in Table 2.68 with a selectable set of the list of active endpoints on the remote device starting with the entry StartIndex and continuing with whole entries until the maximum APS packet length is reached or the application input and output cluster lists is exhausted, along with a status of SUCCESS.

2.4.3.2 End Device Bind, Bind, Unbind, and Bind Management Client Services Primitives

Table 2.69 lists the primitives supported by Device Profile: End Device Bind, Bind and Unbind Client Services. Each of these commands will be discussed in the following sections.

Table 2.69 End Device Bind, Bind, Unbind, and Bind Management Client Service Commands

End Device Bind, Bind and Unbind Client Services	Client Transmission	Server Processing
End_Device_Bind_req	O	O
Bind_req	O	O
Unbind_req	O	O
Bind_Register_req	O	O
Replace_Device_req	O	O
Store_Bkup_Bind_Entry_req	O	O
Remove_Bkup_Bind_Entry_req	O	O
Backup_Bind_Table_req	O	O
Recover_Bind_Table_req	O	O
Backup_Source_Bind_req	O	O
Recover_Source_Bind_req	O	O

2.4.3.2.1 End_Device_Bind_req

The End_Device_Bind_req command (ClusterID=0x0020) shall be formatted as illustrated in Figure 2.43.

Figure 2.43 Format of the End_Device_Bind_req Command Frame

Octets: 2	8	1	2	1	Variable	1	Variable
Binding Target	SrcIEEE Address	Src Endpoint	Profile ID	Num InClusters	InCluster List	Num OutClusters	OutCluster List

Table 2.70 specifies the fields of the End_Device_Bind_req command frame.

Table 2.70 Fields of the End_Device_Bind_req Command

Name	Type	Valid Range	Description
BindingTarget	Device Address	16-bit NWK Address	The address of the target for the binding. This can be either the primary binding cache device or the short address of the local device.
SrcIEEEAddress	IEEE Address	A valid 64-bit IEEE Address	The IEEE address of the device generating the request.
SrcEndpoint	8 bits	1-254	The endpoint on the device generating the request.
ProfileID	Integer	0x0000-0xffff	ProfileID which is to be matched between two End_Device_Bind_req received at the ZigBee Coordinator within the timeout value pre-configured in the ZigBee Coordinator.
NumInClusters	Integer	0x00-0xff	The number of Input Clusters provided for end device binding within the InClusterList.
InClusterList	2 bytes * NumInClusters		List of Input ClusterIDs to be used for matching. The InClusterList is the desired list to be matched by the ZigBee coordinator with the Remote Device's output clusters (the elements of the InClusterList are supported input clusters of the Local Device).
NumOutClusters	Integer	0x00-0xff	The number of Output Clusters provided for matching within OutClusterList.
OutClusterList	2 bytes * NumOutClusters		List of Output ClusterIDs to be used for matching. The OutClusterList is the desired list to be matched by the ZigBee coordinator with the Remote Device's input clusters (the elements of the OutClusterList are supported output clusters of the Local Device).

2.4.3.2.1.1 When Generated

The End_Device_Bind_req is generated from a Local Device wishing to perform End Device Bind with a Remote Device. The End_Device_Bind_req is generated, typically based on some user action like a button press. The destination addressing on this command shall be unicast, and the destination address shall be that of the ZigBee Coordinator.

2.4.3.2.1.2 Effect on Receipt

On receipt of this command, the ZigBee coordinator shall first check that the supplied endpoint is within the specified range. If the supplied endpoint does not fall within the specified range, the ZigBee coordinator shall return an End_Device_Bind_rsp with a status of INVALID_EP.

If the supplied endpoint is within the specified range, the ZigBee Coordinator shall retain the End_Device_Bind_req for a pre-configured timeout duration awaiting a second End_Device_Bind_req. If the second request does not appear within the timeout period, the ZigBee Coordinator shall generate a TIMEOUT status and return it with the End_Device_Bind_rsp to the originating Local Device. Assuming the second End_Device_Bind_req is received within the timeout period, it shall be matched with the first request on the basis of the ProfileID, InClusterList and OutClusterList.

If no match of the ProfileID is detected by using the Profile Id Endpoint Matching Rules (see section 2.3.3.2), or if the ProfileID matches but none of the InClusterList or OutClusterList elements match, a status of NO_MATCH shall be supplied to both Local Devices via End_Device_Bind_rsp to each device. If a match of Profile ID and at least one input or output clusterID is detected, an End_Device_Bind_rsp with status SUCCESS shall be issued to each Local Device which generated the End_Device_Bind_req.

In order to facilitate a toggle action, the ZigBee Coordinator shall then issue an Unbind_req command to the BindingTarget, specifying any one of the matched ClusterID values. If the returned status value is NO_ENTRY, the ZigBee Coordinator shall issue a Bind_req command for each matched ClusterID value. Otherwise, the ZigBee Coordinator shall conclude that the binding records are instead to be removed and shall issue an Unbind_req command for any further matched ClusterID values.

The initial Unbind_req and any subsequent Bind_reqs or Unbind_reqs containing the matched clusters shall be directed to one of the BindingTargets specified by the generating devices. The BindingTarget is selected on an individual basis for each matched cluster, as the Binding Target selected by the generating device having that cluster as an output cluster. The SrcAddress field shall contain the 64-bit IEEE address of that same generating device and the SrcEndp field shall contain its endpoint. The DstAddress field shall contain the 64-bit IEEE address of the generating device having the matched cluster in its input cluster list and the DstEndp field shall contain its endpoint.

2.4.3.2.2 Bind_req

The Bind_req command (ClusterID=0x0021) shall be formatted as illustrated in Figure 2.44.

Figure 2.44 Format of the Bind_req Command Frame

Octets: 8	**1**	**2**	**1**	**2/8**	**0/1**
SrcAddress	SrcEndp	ClusterID	DstAddrMode	DstAddress	DstEndp

Table 2.71 specifies the fields of the Bind_req command frame.

Table 2.71 Fields of the Bind_req Command

Name	**Type**	**Valid Range**	**Description**
SrcAddress	IEEE Address	A valid 64-bit IEEE address	The IEEE address for the source.
SrcEndp	Integer	0x01-0xfe	The source endpoint for the binding entry.
ClusterID	Integer	0x0000-0xffff	The identifier of the cluster on the source device that is bound to the destination.

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x00-0xff	The addressing mode for the destination address used in this command. This field can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddress and DstEndp not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddress and DstEndp present 0x04 – 0xff = reserved
DstAddress	Address	As specified by the DstAddrMode field	The destination address for the binding entry.
DstEndp	Integer	0x01-0xfe	This field shall be present only if the DstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.

2.4.3.2.2.1 When Generated

The Bind_req is generated from a Local Device wishing to create a Binding Table entry for the source and destination addresses contained as parameters. The destination addressing on this command shall be unicast only, and the destination address shall be that of a Primary binding table cache or to the SrcAddress itself. The Binding Manager is optionally supported on the source device (unless that device is also the ZigBee Coordinator) so that device shall issue a NOT_SUPPORTED status to the Bind_req if not supported.

2.4.3.2.2.2 Effect on Receipt

Upon receipt, a Remote Device (a Primary binding table cache or the device designated by SrcAddress) shall create a Binding Table entry based on the parameters supplied in the Bind_req if the Binding Manager is supported. If the remote device is a primary binding table cache, the following additional processing is required. First, the primary cache shall check its table of devices holding their own source bindings for the device in SrcAddress and, if it is found, shall issue another Bind_req to that device with the same entry. Second, the primary cache shall check if there is a backup binding table cache and, if so, shall issue a Store_Bkup_Binding_Entry_req command to backup the new entry. The Remote Device shall then respond with SUCCESS if the entry has been created by the Binding Manager; otherwise, the Remote Device shall respond with NOT_SUPPORTED.

2.4.3.2.3 Unbind_req

The Unbind_req command (ClusterID=0x0022) shall be formatted as illustrated in Figure 2.45.

Figure 2.45 Format of the Unbind_req Command Frame

Octets: 8	1	2	1	2/8	0/1
SrcAddress	SrcEndp	ClusterID	DstAddrMode	DstAddress	DstEndp

Table 2.72 specifies the fields of the Unbind_req command frame.

Table 2.72 Fields of the Unbind_req Command

Name	Type	Valid Range	Description
SrcAddress	IEEE Address	A valid 64-bit IEEE address	The IEEE address for the source
SrcEndp	Integer	0x01-0xfe	The source endpoint for the binding entry
ClusterID	Integer	0x0000-0xffff	The identifier of the cluster on the source device that is bound to the destination.
DstAddrMode	Integer	0x00-0xff	The addressing mode for the destination address used in this command. This field can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddress and DstEndp not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddress and DstEndp present 0x04 – 0xff = reserved
DstAddress	Address	As specified by the DstAddrMode field	The destination address for the binding entry.
DstEndp	Integer	0x01-00xfe	This field shall be present only if the DstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.

2.4.3.2.3.1 When Generated

The Unbind_req is generated from a Local Device wishing to remove a Binding Table entry for the source and destination addresses contained as parameters. The destination addressing on this command shall be unicast only and the destination address must be that of the Primary binding table cache or the SrcAddress.

2.4.3.2.3.2 Effect on Receipt

The Remote Device shall evaluate whether this request is supported. If the request is not supported, a Status of NOT_SUPPORTED shall be returned. If the request is supported, the Remote Device (a Primary binding table cache or the SrcAddress) shall remove a Binding Table entry based on the parameters supplied in the Unbind_req. If the Remote Device is a primary binding table cache, the following additional processing is required. First, the primary cache shall check its table of devices holding their own source bindings for the device in SrcAddress and, if it is found, shall issue another Unbind_req to that device with the same entry. Second, the primary cache shall check if there is a backup binding table cache and, if so, shall issue a Remove_Bkup_Bind_Entry_req command to remove the backup of this entry. If a Binding Table entry for the SrcAddress, SrcEndp, ClusterID, DstAddress, DstEndp contained as parameters does not exist, the Remote Device shall respond with NO_ENTRY. Otherwise, the Remote Device shall delete the indicated Binding Table entry and respond with SUCCESS.

2.4.3.2.4 Bind_Register_req

The Bind_Register_req command (ClusterID=0x0023) shall be formatted as illustrated in Figure 2.46.

Figure 2.46 Format of the Bind_Register_req Command Frame

Octets: 8
NodeAddress

Table 2.73 specifies the fields for the Bind_Register_req command frame.

Table 2.73 Fields of the Bind_Register_req Command

Name	**Type**	**Valid Range**	**Description**
NodeAddress	IEEE Address	A valid 64-bit IEEE address	The address of the node wishing to hold its own binding table.

2.4.3.2.4.1 When Generated

The Bind_Register_req is generated from a Local Device and sent to a primary binding table cache device to register that the local device wishes to hold its own binding table entries. The destination addressing mode for this request is unicast.

2.4.3.2.4.2 Effect on Receipt

If the remote device is not a primary binding table cache it shall return a status of NOT_SUPPORTED. Otherwise, the primary binding table cache shall add the NodeAddress given by the parameter to its table of source devices which have chosen to store their own binding table. If this fails, it shall return a status of TABLE_FULL. Otherwise, it returns a status of SUCCESS. If an entry for the NodeAddress already exists in the table of source devices, the behavior will be the same as if it had been newly added. The source device should clear its source binding table before issuing this command to avoid synchronization problems. In the successful case, any existing bind entries from the binding table whose source address is NodeAddress will be sent to the requesting device for inclusion in its source binding table. See Bind_Register_rsp for further details. Subsequent bind entries written to the binding list will cause copies to be written to the source device using Bind_req.

2.4.3.2.5 Replace_Device_req

The Replace_Device_req command (ClusterID=0x0024) shall be formatted as illustrated in Figure 2.47.

Figure 2.47 Format of the Replace_Device_req Command Frame

Octets: 8	**1**	**8**	**1**
OldAddress	OldEndpoint	NewAddress	NewEndpoint

Table 2.74 specifies the fields for the Replace_Device_req command frame.

Table 2.74 Fields of the Replace_Device_req Command

Name	Type	Valid Range	Description
OldAddress	IEEE Address	A valid 64-bit IEEE	The address of the node being replaced.
OldEndpoint	Integer	0x00 - 0xfe	The endpoint being replaced.
NewAddress	IEEE Address	A valid 64-bit IEEE	The replacement address.
NewEndpoint	Integer	0x01 - 0xfe	The replacement endpoint.

2.4.3.2.5.1 When Generated

The Replace_Device_req is intended for use by a special device such as a Commissioning tool and is sent to a primary binding table cache device to change all binding table entries which match OldAddress and OldEndpoint as specified. Note that OldEndpoint = 0 has special meaning and signifies that only the address needs to be matched. The endpoint in the binding table will not be changed in this case and so NewEndpoint is ignored. The processing changes all binding table entries for which the source address is the same as OldAddress and, if OldEndpoint is non-zero, for which the source endpoint is the same as OldEndpoint. It shall also change all binding table entries which have the destination address the same as OldAddress and, if OldEndpoint is non-zero, the destination endpoint the same as OldEndpoint. The destination addressing mode for this request is unicast.

2.4.3.2.5.2 Effect on Receipt

If the remote device is not a primary binding table cache, it shall return a status of NOT_SUPPORTED. The primary binding table cache shall check if the OldAddress parameter is non-zero and, if so, shall search its binding table for entries of source addresses and source endpoint, or destination addresses and destination endpoint, that are set the same as OldAddress and OldEndpoint. It shall change these entries to have NewAddress and NewEndpoint. In the case that OldEndpoint is zero, the primary binding table cache shall search its binding table for entries whose source address or destination address match OldAddress. It shall change these entries to have NewAddress leaving the endpoint value unchanged and ignoring NewEndpoint. It shall then return a response of SUCCESS. The primary binding table cache shall also be responsible for notifying affected devices which are registered as holding their own source binding table of the changes. This will be necessary for each changed binding table entry, where the destination address was changed and the source address appears in the list of source devices which have chosen to store their own binding table. In each of these cases, the amended binding table entry will be sent to the source device using an Unbind_req command for the old entry followed by a Bind_req command for the new one. In the case that the source address of the bind entry has been changed, it will be necessary for the primary binding table cache to send an Unbind_req command to the old source device if it is a source bind device and to send a Bind_req command to the new source bind device if it is a source bind device. The primary binding table cache shall also update the backup binding table cache by means of the Remove_bkup_binding_entry_req command for the old entry and Store_bkup_binding_entry_req for the altered entry.

2.4.3.2.6 Store_Bkup_Bind_Entry_req

The Store_Bkup_Bind_Entry_req command (ClusterID=0x0025) shall be formatted as illustrated in Figure 2.48.

Figure 2.48 Format of the Store_Bkup_Bind_Entry_req Command Frame

Octets: 8	1	2	1	2/8	0/1
SrcAddress	SrcEndp	ClusterID	DstAddrMode	DstAddress	DstEndp

Table 2.75 specifies the fields of the Store_Bkup_Bind_Entry_req command frame.

Table 2.75 Fields of the Store_Bkup_Bind_Entry_req Command

Name	Type	Valid Range	Description
SrcAddress	IEEE Address	A valid 64-bit IEEE address	The IEEE address for the source.
SrcEndpoint	Integer	0x01 - 0xfe	The source endpoint for the binding entry.
ClusterId	Integer	0x0000 - 0xffff	The identifier of the cluster on the source device that is bound to the destination.
DstAddrMode	Integer	0x00-0xff	The addressing mode for the destination address used in this command. This field can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddress and DstEndp not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddress and DstEndp present 0x04 – 0xff = reserved
DstAddress	Address	As specified by the DstAddrMode field	The destination address for the binding entry.
DstEndp	Integer	0x01-0xfe	This field shall be present only if the DstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.

2.4.3.2.6.1 When Generated

The Store_Bkup_Bind_Entry_req is generated from a local primary binding table cache and sent to a remote backup binding table cache device to request backup storage of the entry. It will be generated whenever a new binding table entry has been created by the primary binding table cache. The destination addressing mode for this request is unicast.

2.4.3.2.6.2 Effect on Receipt

If the remote device is not a backup binding table cache it shall return a status of NOT_SUPPORTED. If it is the backup binding table cache, it should maintain the identity of the primary binding table cache from previous discovery. If the contents of the Store_Bkup_Bind_Entry parameters match an existing entry in the binding table cache, then the remote device shall return SUCCESS. Otherwise, the backup binding table cache shall add the binding entry to its binding table and return a status of SUCCESS. If there is no room, it shall return a status of TABLE_FULL.

2.4.3.2.7 Remove_Bkup_Bind_Entry_req

The Remove_Bkup_Bind_Entry_req command (ClusterID=0x0026) shall be formatted as illustrated in Figure 2.49.

Figure 2.49 Format of the Remove_Bkup_Bind_Entry_req Command Frame

Octets: 8	1	2	1	2/8	0/1
SrcAddress	SrcEndp	ClusterID	DstAddrMode	DstAddress	DstEndp

Table 2.76 specifies the fields of the Remove_Bkup_Bind_Entry_req command frame.

Table 2.76 Fields of the Remove_Bkup_Bind_Entry_req Command

Name	Type	Valid Range	Description
SrcAddress	IEEE Address	A valid 64-bit IEEE address	The IEEE address for the source.
SrcEndpoint	Integer	0x01 - 0xfe	The endpoint for the binding entry.
ClusterId	Integer	0x0000 - 0xffff	The identifier of the cluster on the source device that is bound to the destination.
DstAddrMode	Integer	0x00-0xff	The addressing mode for the destination address used in this command. This field can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddress and DstEndp not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddress and DstEndp present 0x04 – 0xff = reserved
DstAddress	Address	As specified by the DstAddrMode field	The destination address for the binding entry.

Name	Type	Valid Range	Description
DstEndp	Integer	0x01-0xfe	This field shall be present only if the DstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.

2.4.3.2.7.1 When Generated

The Remove_Bkup_Bind_Entry_req is generated from a local primary binding table cache and sent to a remote backup binding table cache device to request removal of the entry from backup storage. It will be generated whenever a binding table entry has been unbound by the primary binding table cache. The destination addressing mode for this request is unicast.

2.4.3.2.7.2 Effect on Receipt

If the remote device is not a backup binding table cache, it shall return a status of NOT_SUPPORTED. If it is a backup binding table cache, it should maintain the identity of the primary binding table cache from previous discovery. If it does not recognize the sending device as the primary binding table cache, it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall search its binding table for the entry corresponding to the supplied parameters. If no entry is found, it shall return a status of NO_ENTRY. Otherwise, it shall delete the entry and return a status of SUCCESS.

2.4.3.2.8 Backup_Bind_Table_req

The Backup_Bind_Table_req command (ClusterID=0x0027) shall be formatted as illustrated in Figure 2.50.

Figure 2.50 Format of the Backup_Bind_Table_req Command Frame

Octets: 2	2	2	Variable
BindingTableEntries	StartIndex	BindingTableListCount	BindingTableList

Table 2.77 specifies the fields of the Backup_Bind_Table_req command frame.

Table 2.77 Fields of the Backup_Bind_Table_req Command

Name	Type	Valid Range	Description
BindingTableEntries	Integer	0x0000 - 0xffff	Total number of binding table entries on the primary binding table cache device.
StartIndex	Integer	0x0000 - 0xffff	Starting index within the binding table of entries.
BindingTableListCount	Integer	0x0000 - 0xffff	Number of binding table entries included within BindingTableList.

Name	Type	Valid Range	Description
BindingTableList	List of binding descriptors	The list shall contain the number of elements given by the BindingTableListCount	A list of descriptors beginning with the StartIndex element and continuing for BindingTableListCount of the elements in the primary binding table cache devices's binding table (see Table 2.134 for details.)

2.4.3.2.8.1 When Generated

The Backup_Bind_Table_req is generated from a local primary binding table cache and sent to the remote backup binding table cache device to request backup storage of its entire binding table. The destination addressing mode for this request is unicast.

2.4.3.2.8.2 Effect on Receipt

If the remote device is not a backup binding table cache, it shall return a status of NOT_SUPPORTED. If it is a backup binding table cache, it should maintain the identity of the primary binding table cache from previous discovery. If it does not recognize the sending device as a primary binding table cache, it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall overwrite the binding entries in its binding table starting with StartIndex and continuing for BindingTableListCount entries. If this exceeds its table size, it shall fill in as many entries as possible and return a status of TABLE_FULL. Otherwise, it shall return a status of SUCCESS. The table is effectively truncated to the end of the last entry written by this request. The new size of the table is returned in the response and will be equal to
StartIndex + BindingTableListCount unless TABLE_FULL is being returned it which case it will be the maximum size of the table.

2.4.3.2.9 Recover_Bind_Table_req

The Recover_Bind_Table_req command (ClusterID=0x0028) shall be formatted as illustrated in Figure 2.51.

Figure 2.51 Fields of the Recover_Bind_Table_req Command Frame

Octets: 2
StartIndex

Table 2.78 specifies the fields of the Recover_Bind_Table_req command frame.

Table 2.78 Fields of the Recover_Bind_Table_req Command

Name	Type	Valid Range	Description
StartIndex	Integer	0x0000 - 0xffff	Starting index for the requested elements of the binding table

2.4.3.2.9.1 When Generated

The Recover_Bind_Table_req is generated from a local primary binding table cache and sent to a remote backup binding table cache device when it wants a complete restore of the binding table. The destination addressing mode for this request is unicast.

2.4.3.2.9.2 Effect on Receipt

If the remote device is not the backup binding table cache, it shall return a status of NOT_SUPPORTED. If it does not recognize the sending device as a primary binding table cache it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall prepare a list of binding table entries from its backup beginning with StartIndex. It will fit in as many entries as possible into a Recover_Bind_Table_rsp command and return a status of SUCCESS.

2.4.3.2.10 Backup_Source_Bind_req

The Backup_Source_Bind_req command (ClusterID=0x0029) shall be formatted as illustrated in Figure 2.52.

Figure 2.52 Fields of the Backup_Source_Bind_req Command Frame

Octets: 2	2	2	Variable
SourceTableEntries	StartIndex	SourceTableListCount	SourceTableList

Table 2.79 specifies the fields of the Backup_Source_Bind_req command frame.

Table 2.79 Fields of the Backup_Source_Bind_req Command

Name	Type	Valid Range	Description
SourceTableEntries	Integer	0x0000 - 0xffff	Total number of source table entries on the primary binding table cache device.
StartIndex	Integer	0x0000 - 0xffff	Starting index within the binding table of the entries in SourceTableList.
SourceTableListCount	Integer	0x0000 - 0xffff	Number of source table entries included within SourceTableList.
SourceTableList	List of IEEE Addresses	The list shall contain the number of elements given by the SourceTableListCount	A list of addresses beginning with the StartIndex element and continuing for SourceTableListCount of source addresses in the primary binding table cache device's source table.

2.4.3.2.10.1 When Generated

The Backup_Source_Bind_req is generated from a local primary binding table cache and sent to a remote backup binding table cache device to request backup storage of its entire source table. The destination addressing mode for this request is unicast.

2.4.3.2.10.2 Effect on Receipt

If the remote device is not the backup binding table cache, it shall return a status of NOT_SUPPORTED. If it does not recognize the sending device as a primary binding table cache, it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall overwrite the source entries in its backup source table starting with StartIndex and continuing for SourceTableListCount entries. If this exceeds its table size, it shall return a status of TABLE_FULL. Otherwise, it shall return a status of SUCCESS. The command always truncates the backup table to a number of entries equal to its maximum size or SourceTableEntries, whichever is smaller.

2.4.3.2.11 Recover_Source_Bind_req

The Recover_Source_Bind_req command (ClusterID=0x002a) shall be formatted as illustrated in Figure 2.53.

Figure 2.53 Format of the Recover_Source_Bind_req Command Frame

Octets: 2
StartIndex

Table 2.80 specifies the fields of the Recover_Source_Bind_req command frame.

Table 2.80 Fields of the Recover_Source_Bind_req Command

Name	**Type**	**Valid Range**	**Description**
StartIndex	Integer	0x0000 - 0xffff	Starting index for the requested elements of the binding table

2.4.3.2.11.1 When Generated

The Recover_Source_Bind_req is generated from a local primary binding table cache and sent to the remote backup binding table cache device when it wants a complete restore of the source binding table. The destination addressing mode for this request is unicast.

2.4.3.2.11.2 Effect on Receipt

If the remote device is not the backup binding table cache it shall return a status of NOT_SUPPORTED. If it does not recognize the sending device as a primary binding table cache, it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall prepare a list of source binding table entries from its backup beginning with StartIndex. It will fit in as many entries as possible into a Recover_Source_Bind_rsp command and return a status of SUCCESS.

2.4.3.3 Network Management Client Services

Table 2.81 lists the commands supported by Device Profile: Network Management Client Services. Each of these primitives will be discussed in the following sections.

Table 2.81 Network Management Client Services Commands

Network Management Client Services	Client Transmission	Server Processing
Mgmt_NWK_Disc_req	O	O
Mgmt_Lqi_req	O	M
Mgmt_Rtg_req	O	O
Mgmt_Bind_req	O	M
Mgmt_Leave_req	O	M
Mgmt_Direct_Join_req	O	O
Mgmt_Permit_Joining_req	O	M
Mgmt_Cache_req	O	O
Mgmt_NWK_Update_req	O	O

2.4.3.3.1 Mgmt_NWK_Disc_req

The Mgmt_NWK_Disc_req command (ClusterID=0x0030) shall be formatted as illustrated in Figure 2.54.

Figure 2.54 Format of the Mgmt_NWK_Disc_req Command Frame

Octets: 4	1	1
ScanChannels	ScanDuration	StartIndex

Table 2.82 specifies the fields for the Mgmt_NWK_Disc_req command frame.

Table 2.82 Fields of the Mgmt_NWK_Disc_req Command

Name	Type	Valid Range	Description
ScanChannels	Bitmap	32-bit field	See section 3.2.2.1 for details on NLME-NETWORK-DISCOVERY.request ScanChannels parameter.
ScanDuration	Integer	0x00-0x0e	A value used to calculate the length of time to spend scanning each channel. The time spent scanning each channel is (aBaseSuperframeDuration * (2^n + 1)) symbols, where n is the value of the ScanDuration parameter. For more information on MAC sub-layer scanning (see [B1].

Name	Type	Valid Range	Description
StartIndex	Integer	0x00-0xff	Starting index within the resulting NLME-NETWORK-DISCOVERY.confirm NetworkList to begin reporting for the Mgmt_NWK_Disc_rsp.

2.4.3.3.1.1 When Generated

The Mgmt_NWK_Disc_req is generated from a Local Device requesting that the Remote Device execute a Scan to report back networks in the vicinity of the Local Device. The destination addressing on this command shall be unicast.

2.4.3.3.1.2 Effect on Receipt

The Remote Device shall execute an NLME-NETWORK-DISCOVERY.request using the ScanChannels and ScanDuration parameters supplied with the Mgmt_NWK_Disc_req command. The results of the Scan shall be reported back to the Local Device via the Mgmt_NWK_Disc_rsp command.

If this command is not supported in the Remote Device, the return status provided with the Mgmt_NWK_Disc_rsp shall be NOT_SUPPORTED. If the scan was successful, the Mgmt_NWK_Disc_rsp command shall contain a status of SUCCESS and the results of the scan shall be reported, beginning with the StartIndex element of the NetworkList. If the scan was unsuccessful, the Mgmt_NWK_Disc_rsp command shall contain the error code reported in the NLME-NETWORK-DISCOVERY.confirm primitive.

2.4.3.3.2 Mgmt_Lqi_req

The Mgmt_Lqi_req command (ClusterID=0x0031) shall be formatted as illustrated in Figure 2.55.

Figure 2.55 Format of the Mgmt_Lqi_req Command Frame

Octets: 1
StartIndex

Table 2.83 specifies the fields for the Mgmt_NWK_Disc_req command frame.

Table 2.83 Fields of the Mgmt_Lqi_req Command

Name	Type	Valid Range	Description
StartIndex	Integer	0x00-0xff	Starting Index for the requested elements of the Neighbor Table.

2.4.3.3.2.1 When Generated

The Mgmt_Lqi_req is generated from a Local Device wishing to obtain a neighbor list for the Remote Device along with associated LQI values to each neighbor. The destination addressing on this command shall be unicast only. It may be sent to a coordinator, router, or end device.

2.4.3.3.2.2 Effect on Receipt

Upon receipt, a Remote Device (ZigBee Router or ZigBee Coordinator) shall retrieve the entries of the neighbor table and associated LQI values via the NLME-GET.request primitive (for the *nwkNeighborTable* attribute) and report the resulting neighbor table (obtained via the NLME-GET.confirm primitive) via the Mgmt_Lqi_rsp command.

Prior to revision 21 of this specification, server processing of this command was optional. Additionally end devices were not required to support the command. As a result some devices may return NOT_SUPPORTED.

If this command is not supported in the Remote Device, the return status provided with the Mgmt_Lqi_rsp shall be NOT_SUPPORTED. If the neighbor table was obtained successfully, the Mgmt_Lqi_rsp command shall contain a status of SUCCESS and the neighbor table shall be reported, beginning with the element in the list enumerated as StartIndex. If the neighbor table was not obtained successfully, the Mgmt_Lqi_rsp command shall contain the error code reported in the NLME-GET.confirm primitive.

2.4.3.3.3 Mgmt_Rtg_req

The Mgmt_Rtg_req command (ClusterID=0x0032) shall be formatted as illustrated in Figure 2.56.

Figure 2.56 Format of the Mgmt_Rtg_req Command Frame

Octets: 1
StartIndex

Table 2.84 specifies the fields for the Mgmt_Rtg_req command frame.

Table 2.84 Fields of the Mgmt_Rtg_req Command

Name	Type	Valid Range	Description
StartIndex	Integer	0x00-0xff	Starting Index for the requested elements of the Routing Table.

2.4.3.3.3.1 When Generated

The Mgmt_Rtg_req is generated from a Local Device wishing to retrieve the contents of the Routing Table from the Remote Device. The destination addressing on this command shall be unicast only and the destination address must be that of the ZigBee Router or ZigBee Coordinator.

2.4.3.3.3.2 Effect on Receipt

Upon receipt, a Remote Device (ZigBee Coordinator or ZigBee Router) shall retrieve the entries of the routing table from the NWK layer via the NLME-GET.request primitive (for the *nwkRouteTable* attribute) and report the resulting routing table (obtained via the NLME-GET.confirm primitive) via the Mgmt_Rtg_rsp command.

If the Remote Device does not support this optional management request, it shall return a Status of NOT_SUPPORTED. If the routing table was obtained successfully, the Mgmt_Rtg_req command shall contain a status of SUCCESS and the routing table shall be reported, beginning with the element in the list enumerated as StartIndex. If the routing table was not obtained successfully, the Mgmt_Rtg_rsp command shall contain the error code reported in the NLME-GET.confirm primitive.

2.4.3.3.4 Mgmt_Bind_req

The Mgmt_Bind_req command (ClusterID=0x0033) shall be formatted as illustrated in Figure 2.57.

Figure 2.57 Format of the Mgmt_Bind_req Command Frame

Octets: 1
StartIndex

Table 2.85 specifies the fields for the Mgmt_Bind_req command frame.

Table 2.85 Fields of the Mgmt_Bind_req Command

Name	Type	Valid Range	Description
StartIndex	Integer	0x00-0xff	Starting Index for the requested elements of the Binding Table.

2.4.3.3.4.1 When Generated

The Mgmt_Bind_req is generated from a Local Device wishing to retrieve the contents of the Binding Table from the Remote Device. The destination addressing on this command shall be unicast only and the destination address must be that of a Primary binding table cache or source device holding its own binding table.

2.4.3.3.4.2 Effect on Receipt

Upon receipt, a Remote Device shall retrieve the entries of the binding table from the APS sub-layer via the APSME-GET.request primitive (for the *apsBindingTable* attribute) and report the resulting binding table (obtained via the APSME-GET.confirm primitive) via the Mgmt_Bind_rsp command.

If the Remote Device does not support this optional management request, it shall return a status of NOT_SUPPORTED. If the binding table was obtained successfully, the Mgmt_Bind_rsp command shall contain a status of SUCCESS and the binding table shall be reported, beginning with the element in the list enumerated as StartIndex. If the binding table was not obtained successfully, the Mgmt_Bind_rsp command shall contain the error code reported in the APSME-GET.confirm primitive.

2.4.3.3.5 Mgmt_Leave_req

The Mgmt_Leave_req command (ClusterID=0x0034) shall be formatted as illustrated in Figure 2.58.

Figure 2.58 Format of the Mgmt_Leave_req Command Frame

Bits: 64	6	1	1
Device Address	Reserved	Remove Children	Rejoin

Table 2.86 specifies the fields for the Mgmt_Leave_req command frame.

Table 2.86 Fields of the Mgmt_Leave_req Command

Name	Type	Valid Range	Description
DeviceAddress	Device Address	An extended 64-bit, IEEE address	See section 3.2.2.16 for details on the Device Address parameter within NLME-LEAVE.request. For DeviceAddress of NULL, a value of 0x0000000000000000 shall be used.
Remove Children	Bit	0 or 1	This field has a value of 1 if the device being asked to leave the network is also being asked to remove its child devices, if any. Otherwise, it has a value of 0.
Rejoin	Bit	0 or 1	This field has a value of 1 if the device being asked to leave from the current parent is requested to rejoin the network. Otherwise, it has a value of 0.

2.4.3.3.5.1 When Generated

The Mgmt_Leave_req is generated from a Local Device requesting that a Remote Device leave the network or to request that another device leave the network. The Mgmt_Leave_req is generated by a management application which directs the request to a Remote Device where the NLME-LEAVE.request is to be executed using the parameter supplied by Mgmt_Leave_req.

2.4.3.3.5.2 Effect on Receipt

Upon receipt, the remote device shall process the leave request by executing the procedure in section 3.6.1.10.3.1. If the leave request was validated and accepted, then the receiving device shall generate the NLME-LEAVE.request to disassociate from the currently associated network. The NLME-LEAVE.request shall have the DeviceAddress parameter set to the local device's *nwkIeeeAddress* from the NIB, the RemoveChildren shall be set to FALSE, and the Rejoin parameter shall be set to FALSE.

The results of the leave attempt shall be reported back to the local device via the Mgmt_Leave_rsp command.

Versions of this specification prior to revision 21 did not mandate the requirement to support this command. Therefore if the remote device did not support this optional management request, it would return a status of NOT_SUPPORTED. All devices certified against version 21 and later are now required to support this command.

If the leave attempt was executed successfully, the Mgmt_Leave_rsp command shall contain a status of SUCCESS. If the leave attempt was not executed successfully, the Mgmt_Leave_rsp command shall contain the error code reported in the NLME-LEAVE.confirm primitive.

2.4.3.3.6 Mgmt_Direct_Join_req

The Mgmt_Direct_Join_req command (ClusterID=0x0035) shall be formatted as illustrated in Figure 2.59.

Figure 2.59 Format of the Mgmt_Direct_Join _req Command Frame

Octets: 8	1
Device Address	Capability Information

Table 2.87 specifies the fields for the Mgmt_Direct_Join_req command frame.

Table 2.87 Fields of the Mgmt_Direct_Join_req Command

Name	Type	Valid Range	Description
DeviceAddress	Device Address	An extended 64-bit, IEEE address	See section 3.2.2.14 for details on the DeviceAddress parameter within NLME-DIRECT-JOIN.request.
CapabilityInformation	Bitmap	See Table 3-47	The operating capabilities of the device being directly joined.

2.4.3.3.6.1 When Generated

The Mgmt_Direct_Join_req is generated from a Local Device requesting that a Remote Device permit a device designated by DeviceAddress to join the network directly. The Mgmt_Direct_Join_req is generated by a management application which directs the request to a Remote Device where the NLME-DIRECT-JOIN.request is to be executed using the parameter supplied by Mgmt_Direct_Join_req.

2.4.3.3.6.2 Effect on Receipt

Upon receipt, the remote device shall issue the NLME-DIRECT-JOIN.request primitive using the DeviceAddress and CapabilityInformation parameters supplied with the Mgmt_Direct_Join_req command. The results of the direct join attempt shall be reported back to the local device via the Mgmt_Direct_Join_rsp command.

If the remote device does not support this optional management request, it shall return a status of NOT_SUPPORTED. If the direct join attempt was executed successfully, the Mgmt_Direct_Join_rsp command shall contain a status of SUCCESS. If the direct join attempt was not executed successfully, the Mgmt_Direct_Join_rsp command shall contain the error code reported in the NLME-DIRECT-JOIN.confirm primitive.

2.4.3.3.7 Mgmt_Permit_Joining_req

The Mgmt_Permit_Joining_req command (ClusterID=0x0036) shall be formatted as illustrated in Figure 2.60.

Figure 2.60 Format of the Mgmt_Permit_Joining_req Command Frame

Octets: 1	1
PermitDuration	TC_Significance

Table 2.88 specifies the fields of the Mgmt_Permit_Joining_req command frame.

Table 2.88 Fields of the Mgmt_Permit_Joining_req Command

Name	Type	Valid Range	Description
PermitDuration	Integer	0x00 - 0xfe	See section 3.2.2.5 for details on the PermitDuration parameter within NLME-PERMIT-JOINING.request.
TC_Significance	Boolean Integer	0x00 - 0x01	This field shall always have a value of 1, indicating a request to change the Trust Center policy. If a frame is received with a value of 0, it shall be treated as having a value of 1.

2.4.3.3.7.1 When Generated

The Mgmt_Permit_Joining_req is generated from a Local Device requesting that a remote device or devices allow or disallow association. The Mgmt_Permit_Joining_req is generated by a management application or commissioning tool which directs the request to a remote device(s) where the NLME-PERMIT-JOINING.request is executed using the PermitDuration parameter supplied by Mgmt_Permit_Joining_req. Additionally, if the remote device is the Trust Center and TC_Significance is set to 1, the Trust Center authentication policy will be affected. The addressing may be unicast or 'broadcast to all routers and coordinator.'

2.4.3.3.7.2 Effect on Receipt

Upon receipt, the remote device(s) shall issue the NLME-PERMIT-JOINING.request primitive using the PermitDuration parameter supplied with the Mgmt_Permit_Joining_req command. If the PermitDuration parameter is not equal to zero or 0xFF, the parameter is a number of seconds and joining is permitted until it counts down to zero, after which time, joining is not permitted. If the PermitDuration is set to zero, joining is not permitted.

Versions of this specification prior to revision 21 allowed a value of 0xFF to be interpreted as 'forever'. Version 21 and later do not allow this. All devices conforming to this specification shall interpret 0xFF as 0xFE. Devices that wish to extend the PermitDuration beyond 0xFE seconds shall periodically re-send the Mgmt_Permit_Joining_req.

If a second Mgmt_Permit_Joining_req is received while the previous one is still counting down, it will supersede the previous request.

A value of zero for the TC_Significance field has been deprecated. The field shall always be included in the message and all received frames shall be treated as though set to 1, regardless of the actual received value. In other words, all Mgmt_Permit_Joining_req shall be treated as a request to change the TC Policy.

If the remote device is the Trust Center the Trust Center authorization policy may be affected. Whether the Trust Center accepts a change in its authorization policy is dependent upon its Trust Center policies. A Trust Center device receiving a Mgmt_Permit_Joining_req shall execute the procedure in section 4.7.3.2 to determine if the request is permitted. If the operation was not permitted, the status code of INVALID_REQUEST shall be set. If the operation was allowed, the status code of SUCCESS shall be set.[1]

If the Mgmt_Permit_Joining_req primitive was received as a unicast, the results of the NLME-PERMIT-JOINING.request shall be reported back to the local device via the Mgmt_Permit_Joining_rsp command. If the command was received as a broadcast, no response shall be sent back.

[1] CCB 1550

2.4.3.3.8 Mgmt_Cache_req

The Mgmt_Cache_req command (ClusterID=0x0037) shall be formatted as illustrated in Figure 2.61.

Figure 2.61 Fields of the Mgmt_Cache_req Command Frame

Octets: 1
StartIndex

Table 2.89 specifies the fields of the Mgmt_Cache_req command frame.

Table 2.89 Fields of the Mgmt_Cache_req Command

Name	Type	Valid Range	Description
StartIndex	Integer	0x00 - 0xff	Starting Index for the requested elements of the discovery cache list.

2.4.3.3.8.1 When Generated

The Mgmt_Cache_req is provided to enable ZigBee devices on the network to retrieve a list of ZigBee End Devices registered with a Primary Discovery Cache device. The destination addressing on this primitive shall be unicast.

2.4.3.3.8.2 Effect on Receipt

Upon receipt, the Remote Device shall determine whether it is a Primary Discovery Cache or whether this optional request primitive is supported. If it is not a Primary Discovery Cache device or the Mgmt_Cache_req primitive is not supported, the Remote Device shall return a status of NOT_SUPPORTED. If the Remote Device is a Primary Discovery Cache and supports the Mgmt_Cache_req, the Remote Device shall return SUCCESS to the Local Device along with the discovery cache list which consists of the NWKAddr and IEEEaddr for each ZigBee End Device registered.

2.4.3.3.9 Mgmt_NWK_Update_req

The Mgmt_NWK_Update_req command (ClusterID=0x0038) shall be formatted as illustrated in Figure 2.62.

Figure 2.62 Fields of the Mgmt_NWK_Update_req Command Frame

Octets: 4	**1**	**0/1**	**0/1**	**0/2**
ScanChannels	ScanDuration	ScanCount	*nwkUpdateId*	*nwkManagerAddr*

Table 2.90 specifies the fields of the Mgmt_NWK_Update_req command frame.

Table 2.90 Fields of the Mgmt_NWK_Update_req Command

Name	Type	Valid Range	Description
ScanChannels	Bitmap	32-bit field	See section 3.2.2.1 for details on NLME-ED-SCAN.request ScanChannels parameter.
ScanDuration	Integer	0x00-0x05 or 0xfe or 0xff	A value used to calculate the length of time to spend scanning each channel. The time spent scanning each channel is (aBaseSuperframeDuration * (2^n + 1)) symbols, where n is the value of the ScanDuration parameter. For more information on MAC sub-layer scanning (see [B1]). If ScanDuration has a value of 0xfe this is a request for channel change. If ScanDuration has a value of 0xff this is a request to change the *apsChannelMask* and *nwkManagerAddr* attributes.
ScanCount	Integer	0x00 - 0x05	This field represents the number of energy scans to be conducted and reported. This field shall be present only if the ScanDuration is within the range of 0x00 to 0x05.
nwkUpdateId	Integer	0x00 - 0xFF	The value of the *nwkUpdateId* contained in this request. This value is set by the Network Channel Manager prior to sending the message. This field shall only be present of the ScanDuration is 0xfe or 0xff. If the ScanDuration is 0xff, then the value in the *nwkUpdateID* shall be ignored.
nwkManagerAddr	Device Address	16-bit NWK address	This field shall be present only if the ScanDuration is set to 0xff, and, where present, indicates the NWK address for the device with the Network Manager bit set in its Node Descriptor.

2.4.3.3.9.1 When Generated

This command is provided to allow updating of network configuration parameters or to request information from devices on network conditions in the local operating environment. The destination addressing on this primitive shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.

2.4.3.3.9.2 Effect on Receipt

Upon receipt, the Remote Device shall determine from the contents of the ScanDuration parameter whether this request is an update to the apsChannelMask and nwkManagerAddr attributes, a channel change command, or a request to scan channels and report the results.

If the ScanDuration parameter is equal to 0xfe, the message is a command to change channels. The receiver shall determine if the channel is one within the range of the current PHY, and if the command has a channel outside that range a response of INVALID_REQUEST shall be generated, and the original request shall be dropped and no further processing shall be done. This command provides a new active channel as a single channel in the ChannelMask in which case the APS IB is not updated. If the channel is valid for the current PHY then the procedure for channel change shall be initiated.

If the ScanDuration parameter is equal to 0xff, the command provides a set of new apsChannelMask along with a new nwkManagerAddr. The Remote Device shall store the apsChannelMask in the APS IB and the nwkManagerAddr in the NIB without invocation of an NLME-ED-SCAN.request.

If this command is unicast with ScanDuration set to 0xfe or 0xff, the Remote Device shall not respond. The network manager should request an APS acknowledgement in this case.

If the ScanDuration is equal to 0x00 to 0x05 and the destination addressing on this command was unicast then the command is interpreted as a request to scan the channels described in ChannelMask. If the channel mask contains a channel outside the range of the current PHY, then a response of INVALID_REQUEST shall be sent back. Otherwise the device shall use the parameter ScanDuration and ScanCount, via invocation of an NLME-ED-SCAN.request. If the Remote Device does not support fragmentation and the resulting response will exceed the APDU, the Remote Device shall perform the Energy Detect Scan on as many of the requested channels as will fit into a single APDU, highlighting the list of actual scanned channels in the response parameter. If multiple scans are requested in the ScanCount, each scan is reported as a separate result. The Remote Device will employ an Energy Detect Scan using the request parameters, modified by the limitation described for fragmentation, and supply the results to the requesting device with a Mgmt_NWK_Update_notify with a SUCCESS status.

Otherwise, if the ScanDuration is equal to 0x06 to 0xfd and the destination addressing on this command was unicast then the Remote Device shall return a status of INVALID_REQUEST.

If the destination addressing on this command was not unicast then the Remote Device shall not transmit a response.

2.4.4 Server Services

The Device Profile Server Services support the processing of device and service discovery requests, end device bind requests, bind requests, unbind requests, and network management requests. Additionally, Server Services support transmission of these responses back to the requesting device.

2.4.4.1 ZDO Response Requirements

A device shall be required to support generation of the correct, corresponding ZDO response to all ZDO requests including ZDO messages defined in a future version of this specification. Server Processing marked optional in Table 2.91, Table 2.114, and Table 2.126 allow for the server to use NOT_SUPPORTED as the status code in the response to indicate the lack of support. ZDO requests unknown to the device shall be treated as unsupported and also use a NOT_SUPPORTED status code to indicate the device's lack of support for that feature. See below for construction of ZDO responses to unsupported requests.For all broadcast addressed requests (of any broadcast address type) to the server, if the command is not supported, the server shall drop the packet. No error status shall be unicast back to the Local Device for any broadcast addressed client request including, but not limited to, requests which are not supported on the server.

For all unicast addressed requests to the server, if the command is not supported, the server shall formulate a response packet including the response Cluster ID and status fields only. The response Cluster ID shall be created by taking the request Cluster ID and setting the high order bit to create the response Cluster ID. The status field shall be set to NOT_SUPPORTED. The resulting response shall be unicast to the requesting client.

2.4.4.2 Device and Service Discovery Server

Table 2.91 lists the commands supported by the Device and Service Discovery Server Services device profile. Each of these commands will be discussed in the following sections. For receipt of the Device_annce command, the server shall check all internal references to the IEEE and 16-bit NWK addresses supplied in the request. For all references to the IEEE address in the Local Device, the corresponding NWK address supplied in the Device_annce shall be substituted. For any other references to the NWK address in the Local Device, the corresponding entry shall be marked as not having a known valid 16-bit NWK address, even if the IEEEAddr field in the message carries the value of 0xffffffffffffffff. The server shall not supply a response to the Device_annce.

Table 2.91 Device and Service Discovery Server Service Primitives

Device and Service Discovery Server Services	Server Processing	Server Generation
NWK_addr_rsp	M	M
IEEE_addr_rsp	M	M
Node_Desc_rsp	M	M
Power_Desc_rsp	M	M
Simple_Desc_rsp	M	M
Active_EP_rsp	M	M
Match_Desc_rsp	M	M
Complex_Desc_rsp	O	M
User_Desc_rsp	O	M
User_Desc_conf	O	M
Parent_annce_rsp	M	M
System_Server_Discovery_rsp	O	M
Discovery_store_rsp	O	M
Node_Desc_store_rsp	O	M
Power_Desc_store_rsp	O	M
Active_EP_store_rsp	O	M
Simple_Desc_store_rsp	O	M

Device and Service Discovery Server Services	**Server Processing**	**Server Generation**
Remove_node_cache_rsp	O	M
Find_node_cache_rsp	O	M
Extended_Simple_Desc_rsp	O	M
Extended_Active_EP_rsp	O	M

For Server Generation requirements see section 2.4.4.1.

2.4.4.2.1 NWK_addr_rsp

The NWK_addr_rsp command (ClusterID=0x8000) shall be formatted as illustrated in Figure 2.63.

Figure 2.63 Format of the NWK_addr_rsp Command Frame

Octets: 1	**8**	**2**	**0/1**	**0/1**	**Variable**
Status	IEEEAddr RemoteDev	NWKAddr RemoteDev	Num AssocDev	StartIndex	NWKAddr AssocDevList

Table 2.92 specifies the fields of the NWK_addr_rsp command frame.

Table 2.92 Fields of the NWK_addr_rsp Command

Name	**Type**	**Valid Range**	**Description**
Status	Integer	SUCCESS, INV_REQUESTTYPE, or DEVICE_NOT_FOUND	The status of the NWK_addr_req command.
IEEEAddrRemoteDev	Device Address	An extended 64-bit, IEEE address	64-bit address for the Remote Device.
NWKAddrRemoteDev	Device Address	A 16-bit, NWK address	16-bit address for the Remote Device.

Name	Type	Valid Range	Description
NumAssocDev	Integer	0x00-0xff	Count of the number of 16-bit short addresses to follow. If the RequestType in the request is Extended Response and there are no associated devices on the Remote Device, this field shall be set to 0. If an error occurs or the Request Type in the request is for a Single Device Response, this field shall not be included in the frame.
StartIndex	Integer	0x00-0xff	Starting index into the list of associated devices for this report. If the RequestType in the request is Extended Response and there are no associated devices on the Remote Device, this field shall not be included in the frame. If an error occurs or the Request Type in the request is for a Single Device Response, this field shall not be included in the frame.
NWKAddrAssocDevList	Device Address List	List of NumAssocDev 16-bit short addresses, each with range 0x0000 - 0xffff	A list of 16-bit addresses, one corresponding to each associated device to Remote Device; The number of 16-bit network addresses contained in this field is specified in the NumAssocDev field. If the RequestType in the request is Extended Response and there are no associated devices on the Remote Device, this field shall not be included in the frame. If an error occurs or the Request Type in the request is for a Single Device Response, this field shall not be included in the frame.

2.4.4.2.1.1 When Generated

The NWK_addr_rsp is generated by a Remote Device in response to a NWK_addr_req command inquiring as to the NWK address of the Remote Device or the NWK address of an address held in the neighbor table (see section 2.4.3.1.1.2 for a detailed description). The destination addressing on this command is unicast.

2.4.4.2.1.2 Effect on Receipt

On receipt of the NWK_addr_rsp command, the recipient is either notified of the status of its attempt to discover a NWK address from an IEEE address or notified of an error. If the NWK_addr_rsp command is received with a Status of SUCCESS, the remaining fields of the command contain the appropriate discovery information, according to the RequestType as specified in the original NWK_Addr_req command. Otherwise, the Status field indicates the error and the NumAssocDev, StartIndex, and NWKAddrAssocDevList fields shall not be included.

2.4.4.2.2 IEEE_addr_rsp

The IEEE_addr_rsp command (ClusterID=0x8001) shall be formatted as illustrated in Figure 2.64.

Figure 2.64 Format of the IEEE_addr_rs Command Frame

Octets: 1	8	2	0/1	0/1	Variable
Status	IEEEAddr RemoteDev	NWKAddr RemoteDev	NumAssocDev	StartIndex	NWKAddr AssocDevList

Table 2.93 specifies the fields of the IEEE_addr_rs command frame.

Table 2.93 IEEE_addr_rsp Parameters

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INV_REQUESTTYPE or DEVICE_NOT_FOUND	The status of the IEEE_addr_req command.
IEEEAddrRemoteDev	Device Address	An extended 64-bit, IEEE address	64-bit address for the Remote Device.
NWKAddrRemoteDev	Device Address	A 16-bit, NWK address	16-bit address for the Remote Device.
NumAssocDev	Integer	0x00-0xff	Count of the number of 16-bit short addresses to follow. If the RequestType in the request is Extended Response and there are no associated devices on the Remote Device, this field shall be set to 0. If an error occurs or the RequestType in the request is for a Single Device Response, this field shall not be included in the frame.

Name	Type	Valid Range	Description
StartIndex	Integer	0x00-0xff	Starting index into the list of associated devices for this report. If the RequestType in the request is Extended Response and there are no associated devices on the Remote Device, this field shall not be included in the frame. If an error occurs or the RequestType in the request is for a Single Device Response, this field shall not be included in the frame.
NWKAddrAssocDevList	Device Address List	List of NumAssocDev 16-bit short addresses, each with range 0x0000 - 0xffff	A list of 16-bit addresses, one corresponding to each associated device to Remote Device; The number of 16-bit network addresses contained in this field is specified in the NumAssocDev field. If the RequestType in the request is Extended Response and there are no associated devices on the Remote Device, this field shall not be included in the frame. If an error occurs or the RequestType in the request is for a Single Device Response, this field shall not be included in the frame

2.4.4.2.2.1 When Generated

The IEEE_addr_rsp is generated by a Remote Device in response to an IEEE_addr_req command inquiring as to the 64-bit IEEE address of the Remote Device or the 64-bit IEEE address of an address held in a local discovery cache (see section 2.4.3.1.2.2 for a detailed description). The destination addressing on this command shall be unicast.

2.4.4.2.2.2 Effect on Receipt

On receipt of the IEEE_addr_rsp command, the recipient is either notified of the status of its attempt to discover an IEEE address from an NWK address or notified of an error. If the IEEE_addr_rsp command is received with a Status of SUCCESS, the remaining fields of the command contain the appropriate discovery information, according to the RequestType as specified in the original IEEE_Addr_req command. Otherwise, the Status field indicates the error and the NumAssocDev, StartIndex, and NWKAddrAssocDevList fields shall not be included.

2.4.4.2.3 Node_Desc_rsp

The Node_Desc_rsp command (ClusterID=0x8002) shall be formatted as illustrated in Figure 2.65.

Figure 2.65 Format of the Node_Desc_rsp Command Frame

Octets: 1	2	See section 2.3.2.3
Status	NWKAddrOfInterest	Node Descriptor

Table 2.94 specifies the fields of the Node_Desc_rsp command frame.

Table 2.94 Fields of the Node_Desc_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, DEVICE_NOT_FOUND, INV_REQUESTTYPE, or NO_DESCRIPTOR	The status of the Node_Desc_req command.
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
NodeDescriptor	Node Descriptor		See the Node Descriptor format in section 2.3.2.3. This field shall only be included in the frame if the status field is equal to SUCCESS.

2.4.4.2.3.1 When Generated

The Node_Desc_rsp is generated by a remote device in response to a Node_Desc_req directed to the remote device. This command shall be unicast to the originator of the Node_Desc_req command.

The remote device shall generate the Node_Desc_rsp command using the format illustrated in Table 2.94 Fields of the Node_Desc_rsp Command. The NWKAddrOfInterest field shall match that specified in the original Node_Desc_req command. If the NWKAddrOfInterest field matches the network address of the remote device, it shall set the Status field to SUCCESS and include its node descriptor (see section 2.3.2.3) in the NodeDescriptor field.

If the NWKAddrOfInterest field does not match the network address of the remote device and it is an end device, it shall set the Status field to INV_REQUESTTYPE and not include the NodeDescriptor field. If the NWKAddrOfInterest field does not match the network address of the remote device and it is the coordinator or a router, it shall determine whether the NWKAddrOfInterest field matches the network address of one of its children. If the NWKAddrOfInterest field does not match the network address of one of the children of the remote device, it shall set the Status field to DEVICE_NOT_FOUND and not include the NodeDescriptor field. If the NWKAddrOfInterest matches the network address of one of the children of the remote device, it shall determine whether a node descriptor for that device is available. If a node descriptor is not available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to
NO_DESCRIPTOR and not include the NodeDescriptor field. If a node descriptor is available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to SUCCESS and include the node descriptor (see section 2.3.2.3) of the matching child device in the NodeDescriptor field.

2.4.4.2.3.2 Effect on Receipt

On receipt of the Node_Desc_rsp command, the recipient is either notified of the node descriptor of the remote device indicated in the original Node_Desc_req command or notified of an error. If the Node_Desc_rsp command is received with a Status of SUCCESS, the NodeDescriptor field shall contain the requested node descriptor. Otherwise, the Status field indicates the error and the NodeDescriptor field shall not be included.

2.4.4.2.4 Power_Desc_rsp

The Power_Desc_rsp command (ClusterID=0x8003) shall be formatted as illustrated in Figure 2.66.

Figure 2.66 Format of the Power_Desc_rsp Command Frame

Octet: 1	**2**	**Variable**
Status	NWKAddrOfInterest	Power Descriptor

Table 2.95 specifies the fields of the Power_Desc_rsp command frame.

Table 2.95 Fields of the Power_Desc_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the Power_Desc_req command.
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
PowerDescriptor	Power Descriptor		See the Node Power Descriptor format in section 2.3.2.4. This field shall only be included in the frame if the status field is equal to SUCCESS.

2.4.4.2.4.1 When Generated

The Power_Desc_rsp is generated by a remote device in response to a Power_Desc_req directed to the remote device. This command shall be unicast to the originator of the Power_Desc_req command.

The remote device shall generate the Power_Desc_rsp command using the format illustrated in Table 2.95. The NWKAddrOfInterest field shall match that specified in the original Power_Desc_req command. If the NWKAddrOfInterest field matches the network address of the remote device, it shall set the Status field to SUCCESS and include its power descriptor (see section 2.3.2.4) in the PowerDescriptor field.

If the NWKAddrOfInterest field does not match the network address of the remote device and it is an end device, it shall set the Status field to INV_REQUESTTYPE and not include the PowerDescriptor field. If the NWKAddrOfInterest field does not match the network address of the remote device and it is the coordinator or a router, it shall determine whether the NWKAddrOfInterest field matches the network address of one of its children. If the NWKAddrOfInterest field does not match the network address of one of the children of the remote device, it shall set the Status field to DEVICE_NOT_FOUND and not include the PowerDescriptor field. If the NWKAddrOfInterest matches the network address of one of the children of the remote device, it shall determine whether a power descriptor for that device is available. If a power descriptor is not available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to NO_DESCRIPTOR and not include the PowerDescriptor field. If a power descriptor is available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to SUCCESS and include the power descriptor (see section 2.3.2.4) of the matching child device in the PowerDescriptor field.

2.4.4.2.4.2 Effect on Receipt

On receipt of the Power_Desc_rsp command, the recipient is either notified of the power descriptor of the remote device indicated in the original Power_Desc_req command or notified of an error. If the Power_Desc_rsp command is received with a Status of SUCCESS, the PowerDescriptor field shall contain the requested power descriptor. Otherwise, the Status field indicates the error and the PowerDescriptor field shall not be included.

2.4.4.2.5 Simple_Desc_rsp

The Simple_Desc_rsp command (ClusterID=0x8004) shall be formatted as illustrated in Figure 2.67.

Figure 2.67 Format of the Simple_Desc_rsp Command Frame

Octet: 1	2	1	Variable
Status	NWKAddrOfInterest	Length	Simple Descriptor

Table 2.96 specifies the fields of the Simple_Desc_rsp command frame.

Table 2.96 Fields of the Simple_Desc_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INVALID_EP, NOT_ACTIVE, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the Simple_Desc_req command.
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
Length	Integer	0x00-0xff	Length in bytes of the Simple Descriptor to follow.
SimpleDescriptor	Simple Descriptor		See the Simple Descriptor format in section 2.3.2.5. This field shall only be included in the frame if the status field is equal to SUCCESS.

2.4.4.2.5.1 When Generated

The Simple_Desc_rsp is generated by a remote device in response to a Simple_Desc_req directed to the remote device. This command shall be unicast to the originator of the Simple_Desc_req command.

The remote device shall generate the Simple_Desc_rsp command using the format illustrated in Table 2.96. The NWKAddrOfInterest field shall match that specified in the original Simple_Desc_req command. If the endpoint field specified in the original Simple_Desc_req command does not fall within the correct range specified in Table 2.96 Fields of the Simple_Desc_req Command, the remote device shall set the Status field to INVALID_EP, set the Length field to 0 and not include the SimpleDescriptor field.

If the NWKAddrOfInterest field matches the network address of the remote device, it shall determine whether the endpoint field specifies the identifier of an active endpoint on the device. If the endpoint field corresponds to an active endpoint, the remote device shall set the Status field to SUCCESS, set the Length field to the length of the simple descriptor on that endpoint, and include the simple descriptor (see section 2.3.2.5) for that endpoint in the SimpleDescriptor field. If the endpoint field does not correspond to an active endpoint, the remote device shall set the Status field to NOT_ACTIVE, set the Length field to 0, and not include the SimpleDescriptor field.

If the NWKAddrOfInterest field does not match the network address of the remote device and it is an end device, it shall set the Status field to INV_REQUESTTYPE, set the Length field to 0, and not include the SimpleDescriptor field. If the NWKAddrOfInterest field does not match the network address of the remote device and it is the coordinator or a router, it shall determine whether the NWKAddrOfInterest field matches the network address of one of its children. If the NWKAddrOfInterest field does not match the network address of one of the children of the remote device, it shall set the Status field to DEVICE_NOT_FOUND, set the Length field to 0, and not include the SimpleDescriptor field.

If the NWKAddrOfInterest matches the network address of one of the children of the remote device, it shall determine whether a simple descriptor for that device and on the requested endpoint is available. If a simple descriptor is not available on the requested endpoint of the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to NO_DESCRIPTOR, set the Length field to 0, and not include the SimpleDescriptor field. If a simple descriptor is available on the requested endpoint of the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to SUCCESS, set the Length field to the length of the simple descriptor on that endpoint, and include the simple descriptor (see section 2.3.2.5) for that endpoint of the matching child device in the SimpleDescriptor field.

2.4.4.2.5.2 Effect on Receipt

On receipt of the Simple_Desc_rsp command, the recipient is either notified of the simple descriptor on the endpoint of the remote device indicated in the original Simple_Desc_req command or notified of an error. If the Simple_Desc_rsp command is received with a Status of SUCCESS, the SimpleDescriptor field shall contain the requested simple descriptor. Otherwise, the Status field indicates the error and the SimpleDescriptor field shall not be included.

2.4.4.2.6 Active_EP_rsp

The Active_EP_rsp command (ClusterID=0x8005) shall be formatted as illustrated in Figure 2.68.

Figure 2.68 Format of the Active_EP_rsp Command Frame

Octet: 1	**2**	**1**	**Variable**
Status	NWKAddrOfInterest	ActiveEPCount	ActiveEPList

Table 2.97 specifies the fields of the Active_EP_rsp command frame.

Table 2.97 Fields of the Active_EP_rsp Command

Name	**Type**	**Valid Range**	**Description**
Status	Integer	SUCCESS, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the Active_EP_req command.
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
ActiveEPCount	Integer	0x00-0xff	The count of active endpoints on the Remote Device.
ActiveEPList			List of bytes each of which represents an 8-bit endpoint.

2.4.4.2.6.1 When Generated

The Active_EP_rsp is generated by a remote device in response to an Active_EP_req directed to the remote device. This command shall be unicast to the originator of the Active_EP_req command.

The remote device shall generate the Active_EP_rsp command using the format illustrated in Table 2.97. The NWKAddrOfInterest field shall match that specified in the original Active_EP_req command. If the NWKAddrOfInterest field matches the network address of the remote device, it shall set the Status field to SUCCESS, set the ActiveEPCount field to the number of active endpoints on that device and include an ascending list of all the identifiers of the active endpoints on that device in the ActiveEPList field.

If the NWKAddrOfInterest field does not match the network address of the remote device and it is an end device, it shall set the Status field to INV_REQUESTTYPE, set the ActiveEPCount field to 0, and not include the ActiveEPList field. If the NWKAddrOfInterest field does not match the network address of the remote device and it is the coordinator or a router, it shall determine whether the NWKAddrOfInterest field matches the network address of a device it holds in a discovery cache. If the NWKAddrOfInterest field does not match the network address of a device it holds in a discovery cache, it shall set the Status field to DEVICE_NOT_FOUND, set the ActiveEPCount field to 0, and not include the ActiveEPList field. If the NWKAddrOfInterest matches the network address of a device held in a discovery cache on the remote device, it shall determine whether that device has any active endpoints. If the discovery information corresponding to the ActiveEP request has not yet been uploaded to the discovery cache, the remote device shall set the Status field to NO_DESCRIPTOR, set the ActiveEPCount field to 0 and not include the ActiveEPList field. If the cached device has no active endpoints, the remote device shall set the Status field to SUCCESS, set the ActiveEPCount field to 0, and not include the ActiveEPList field. If the cached device has active endpoints, the remote device shall set the Status field to SUCCESS, set the ActiveEPCount field to the number of active endpoints on that device, and include an ascending list of all the identifiers of the active endpoints on that device in the ActiveEPList field.

2.4.4.2.6.2 Effect on Receipt

On receipt of the Active_EP_rsp command, the recipient is either notified of the active endpoints of the remote device indicated in the original Active_EP_req command or notified of an error. If the Active_EP_rsp command is received with a Status of SUCCESS, the ActiveEPCount field indicates the number of entries in the ActiveEPList field. Otherwise, the Status field indicates the error and the ActiveEPList field shall not be included.

2.4.4.2.7 Match_Desc_rsp

The Match_Desc_rsp command (ClusterID=0x8006) shall be formatted as illustrated in Figure 2.69.

Figure 2.69 Format of the Match_Desc_rsp Command Frame

Octet: 1	**2**	**1**	**Variable**
Status	NWKAddrOfInterest	Match Length	Match List

Table 2.98 specifies the fields of the Match_Desc_rsp command frame.

Table 2.98 Fields of the Match_Desc_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the Match_Desc_req command.
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
MatchLength	Integer	0x00-0xff	The count of endpoints on the Remote Device that match the request criteria.
MatchList			List of bytes each of which represents an 8-bit endpoint.

2.4.4.2.7.1 When Generated

The Match_Desc_rsp is generated by a remote device in response to a Match_Desc_req either broadcast or unicast to the remote device. This command shall be unicast to the originator of the Match_Desc_req command.

The following describes the procedure for processing the Match_Desc_req and generation of Match_Desc_rsp.

1. Set MatchLength to 0 and create an empty list MatchList.
2. If the receiving device is an End Device and the NWKAddrOfInterest within the Match_Desc_req message does not match the nwkNetworkAddress of the NIB and is not a broadcast address, the following shall be performed. Otherwise it shall proceed to step 3.
    a. If the NWK destination of the message is a broadcast address, no further processing shall be done.
    b. If the NWK destination is a unicast address, the following shall be performed.
        i. Set the Status value to INV_REQUESTTYPE.
        ii. Set the MatchLength to 0.
        iii. Construct a Match_Desc_Rsp with only Status and MatchLength fields.
        iv. Send the message as a unicast to the source of the Match_Desc_req.
        v. No further processing shall be done.
3. If the NWKAddrOfInterest is equal to the nwkNetworkAddress of the NIB, or is a broadcast address, perform the following procedure. Otherwise proceed to step 4.
    a. Apply the match criteria in section 2.4.4.2.7.1.1 for all local Simple Descriptors.
    b. For each Simple Descriptor that matches with at least one cluster, add the endpoint once to MatchList and increment MatchLength.

4. If the NWKAddrOfInterest is not a broadcast address, the NWKAddressOfInterest is not equal to the nwkNetworkAddress of the local NIB, and the device is a coordinator or router, then the following shall be performed. Otherwise proceed to step 5.
    a. Examine each entry in the nwkNeighborTable and perform the following procedure.
        i. If the Network Address of the entry does not match the NWKAddrOfInterest or the Device Type is not equal to 0x02 (ZigBee End Device), do not process this entry. Continue to the next entry in the nwkNeighborTable.
        ii. If no cached Simple Descriptors for the device are available, skip this device and proceed to the next entry in the nwkNeighborTable.
        iii. Apply the match criteria in section 2.4.4.2.7.1.1 for each cached Simple Descriptor.
        iv. For each endpoint that matches with at least once cluster, add that endpoint once to the MatchList and increment MatchLength.
        v. Proceed to step 7.
    b. If the NWKAddrOfInterest does not match any entry in the nwkNeighborTable, perform the following:
        i. Set the Status to DEVICE_NOT_FOUND.
        ii. Construct a Match_Desc_Rsp with Status and MatchLength fields only.
        iii. Unicast the message to the source of the Match_Desc_req.
        iv. No further processing shall take place.
5. If the MatchLength is 0 and the NWK destination of the Match_Desc_Req was a broadcast address, no further processing shall be done. Otherwise proceed to step 6.
6. If the MatchLength is 0 and the NWKAddrOfInterest matched an entry in the nwkNeighborTable, the following shall be performed. Otherwise proceed to step 7.
    a. Set the Status to NO_DESCRIPTOR
    b. Construct a Match_Desc_Rsp with Status and MatchLength only.
    c. Unicast the Match_Desc_Rsp to the source of the Match_Desc_Req.
    d. No further processing shall be done.
7. The following shall be performed. This is the case for both MatchLength > 0 and MatchLength == 0.
    a. Set the Status to SUCCESS.
    b. Construct a Match_Desc_Rsp with Status, NWKAddrOfInterest, MatchLength, and MatchList.
    c. Unicast the response to the NWK source of the Match_Desc_Req.

2.4.4.2.7.1.1 Simple Descriptor Matching Rules

These rules will examine a ProfileID, InputClusterList, OutputClusterList, and a SimpleDescriptor. The following shall be performed:

1. The device shall first check if the ProfileID field matches using the Profile ID of the SimpleDescriptor and the Endpoint Matching Rules (see section 2.3.3.2). If the profile identifiers do not match, the device shall note the match as unsuccessful and perform no further processing.

2. Examine the InputClusterList and compare each item to the Application Input Cluster List of the SimpleDescriptor.
    a. If a cluster ID matches exactly, then the device shall note the match as successful and perform no further matching. Processing is complete.
3. Examine the OutputClusterList and compare each item to the Application Output Cluster List of the SimpleDescriptor.
    a. If a cluster ID matches exactly, then the device shall note the match as successful and perform no further matching. Processing is complete.
4. The device shall note the match as unsuccessful. Processing is complete.

2.4.4.2.7.2 Effect on Receipt

On receipt of the Match_Desc_rsp command, the recipient is either notified of the results of its match criterion query indicated in the original Match_Desc_req command or notified of an error. If the Match_Desc_rsp command is received with a Status of SUCCESS, the MatchList field shall contain the list of endpoints containing simple descriptors that matched the criterion. Otherwise, the Status field indicates the error and the MatchList field shall not be included.

2.4.4.2.8 Complex_Desc_rsp

The Complex_Desc_rsp command (ClusterID=0x8010) shall be formatted as illustrated in Figure 2.70.

Figure 2.70 Format of the Complex_Desc_rsp Command Frame

Octet: 1	**2**	**1**	**Variable**
Status	NWKAddrOfInterest	Length	Complex Descriptor

Table 2.99 specifies the fields of the Complex_Desc_rsp command frame.

Table 2.99 Fields of the Complex_Desc_rsp Command

Name	**Type**	**Valid Range**	**Description**
Status	Integer	SUCCESS, NOT_SUPPORTED, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the Complex_Desc_req command.
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
Length	Integer	0x00-0xff	Length in bytes of the ComplexDescriptor field.

Name	Type	Valid Range	Description
ComplexDescriptor	Complex Descriptor		See the Complex Descriptor format in section 2.3.2.6. This field shall only be included in the frame if the status field is equal to SUCCESS.

2.4.4.2.8.1 When Generated

The Complex_Desc_rsp is generated by a remote device in response to a Complex_Desc_req directed to the remote device. This command shall be unicast to the originator of the Complex_Desc_req command.

The remote device shall generate the Complex_Desc_rsp command using the format illustrated in Table 2.99. The NWKAddrOfInterest field shall match that specified in the original Complex_Desc_req command. If the NWKAddrOfInterest field matches the network address of the remote device but a complex descriptor does not exist, it shall set the Status field to NOT_SUPPORTED, set the Length field to 0, and not include the ComplexDescriptor field. If the NWKAddrOfInterest field matches the network address of the remote device and a complex descriptor exists, it shall set the Status field to SUCCESS, set the Length field to the length of the complex descriptor, and include its complex descriptor (see section 2.3.2.6Complex Descriptor) in the ComplexDescriptor field.

If the NWKAddrOfInterest field does not match the network address of the remote device and it is an end device, it shall set the Status field to INV_REQUESTTYPE, set the Length field to 0, and not include the ComplexDescriptor field. If the NWKAddrOfInterest field does not match the network address of the remote device and it is the coordinator or a router, it shall determine whether the NWKAddrOfInterest field matches the network address of one of its children. If the NWKAddrOfInterest field does not match the network address of one of the children of the remote device, it shall set the Status field to DEVICE_NOT_FOUND, set the Length field to 0, and not include the ComplexDescriptor field. If the NWKAddrOfInterest matches the network address of one of the children of the remote device, it shall determine whether a complex descriptor for that device is available. If a complex descriptor is not available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to NO_DESCRIPTOR, set the Length field to 0, and not include the ComplexDescriptor field. If a complex descriptor is available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to SUCCESS, set the Length field to the length of the complex descriptor for that device, and include the complex descriptor (see section 2.3.2.6) of the matching child device in the ComplexDescriptor field.

2.4.4.2.8.2 Effect on Receipt

On receipt of the Complex_Desc_rsp command, the recipient is either notified of the complex descriptor of the remote device indicated in the original Complex_Desc_req command or notified of an error. If the Complex_Desc_rsp command is received with a Status of SUCCESS, the ComplexDescriptor field shall contain the requested complex descriptor. Otherwise, the Status field indicates the error and the ComplexDescriptor field shall not be included.

2.4.4.2.9 User_Desc_rsp

The User_Desc_rsp command (ClusterID=0x8011) shall be formatted as illustrated in Figure 2.71.

Figure 2.71 Format of the User_Desc_rsp Command Frame

Octet: 1	2	1	Variable
Status	NWKAddrOfInterest	Length	User Descriptor

Table 2.100 specifies the fields of the User_Desc_rsp command frame.

Table 2.100 Fields of the User_Desc_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the User_Desc_req command.
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
Length	Integer	0x00-0x10	Length in bytes of the UserDescriptor field.
UserDescriptor	User Descriptor		See the User Descriptor format in section 2.3.2.7. This field shall only be included in the frame if the status field is equal to SUCCESS.

2.4.4.2.9.1 When Generated

The User_Desc_rsp is generated by a remote device in response to a User_Desc_req directed to the remote device. This command shall be unicast to the originator of the User_Desc_req command.

The remote device shall generate the User_Desc_rsp command using the format illustrated in Table 2.100. The NWKAddrOfInterest field shall match that specified in the original User_Desc_req command. If the NWKAddrOfInterest field matches the network address of the remote device but a user descriptor does not exist, it shall set the Status field to NO_DESCRIPTOR, set the Length field to 0, and not include the UserDescriptor field. If the NWKAddrOfInterest field matches the network address of the remote device and a user descriptor exists, it shall set the Status field to SUCCESS, set the Length field to the length of the user descriptor, and include its user descriptor (see section 2.3.2.7) in the UserDescriptor field.

If the NWKAddrOfInterest field does not match the network address of the remote device and it is an end device, it shall set the Status field to INV_REQUESTTYPE, set the Length field to 0, and not include the UserDescriptor field. If the NWKAddrOfInterest field does not match the network address of the remote device and it is the coordinator or a router, it shall determine whether the NWKAddrOfInterest field matches the network address of one of its children. If the NWKAddrOfInterest field does not match the network address of one of the children of the remote device, it shall set the Status field to DEVICE_NOT_FOUND, set the Length field to 0, and not include the UserDescriptor field. If the NWKAddrOfInterest matches the network address of one of the children of the remote device, it shall determine whether a user descriptor for that device is available. If a user descriptor is not available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to NO_DESCRIPTOR, set the Length field to 0, and not include the UserDescriptor field. If a user descriptor is available for the child indicated by the
NWKAddrOfInterest field, the remote device shall set the Status field to SUCCESS, set the Length field to the length of the user descriptor for that device, and include the user descriptor (see section 2.3.2.7) of the matching child device in the UserDescriptor field.

2.4.4.2.9.2 Effect on Receipt

On receipt of the User_Desc_rsp command, the recipient is either notified of the user descriptor of the remote device indicated in the original User_Desc_req command or notified of an error. If the User_Desc_rsp command is received with a Status of SUCCESS, the UserDescriptor field shall contain the requested user descriptor. Otherwise, the Status field indicates the error and the UserDescriptor field shall not be included.

2.4.4.2.10 System_Server_Discovery_rsp

The System_Server_Discovery_rsp command (ClusterID=0x8015) shall be formatted as illustrated in Figure 2.72.

Figure 2.72 System_Server_Discovery_rsp Command Frame

Octet: 1	**2**
Status	ServerMask

Table 2.101 specifies the fields of the System_Server_Discovery_rsp command frame.

Table 2.101 Fields of the System_Server_Discovery_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS	The status of the System_Server_Discovery_rsp command.
ServerMask	Integer	Bitmap	See Table 2.32 for bit assignments.

2.4.4.2.10.1 When Generated

The System_Server_Discovery_rsp is generated from Remote Devices on receipt of a System_Server_Discovery_req primitive if the parameter matches the Server Mask field in its node descriptor. If there is no match, the System_Server_Discovery_req shall be ignored and no response given. Matching is performed by masking the ServerMask parameter of the System_Server_Discovery_req with the Server Mask field in the node descriptor. This command shall be unicast to the device which sent System_Server_Discovery_req with Acknowledge request set in TxOptions. The parameter ServerMask contains the bits in the parameter of the request which match the server mask in the node descriptor.

2.4.4.2.10.2 Effect on Receipt

The requesting device is notified that this device has some of the system server functionality that the requesting device is seeking.

If the Network Manager bit was set in the System_Server_Discovery_rsp, then the Remote Device's NWK address shall be set into the *nwkManagerAddr* of the NIB.

2.4.4.2.11 User_Desc_conf

The User_Desc_conf command (ClusterID=0x8014) shall be formatted as illustrated in Figure 2.73.

Figure 2.73 Format of the User_Desc_conf Command Frame

Octets: 1	2
Status	NWKAddrOfInterest

Table 2.102 specifies the fields of the User_Desc_conf command frame.

Table 2.102 Fields of the User_Desc_conf Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the User_Desc_set command.
NWKAddrOfInterest	Device Address	Any 16-bit NWK address	The network address of the device on which the user descriptor set attempt was made.

2.4.4.2.11.1 When Generated

The User_Desc_conf is generated by a remote device in response to a User_Desc_set directed to the remote device. This command shall be unicast to the originator of the User_Desc_set command.

The remote device shall generate the User_Desc_conf command using the format illustrated in Table 2.102. The NWKAddrOfInterest field shall match that specified in the original User_Desc_set command. If the NWKAddrOfInterest field matches the network address of the remote device but a user descriptor does not exist, it shall set the Status field to NOT_SUPPORTED. If the NWKAddrOfInterest field matches the network address of the remote device and a user descriptor exists, it shall set the Status field to SUCCESS and configure the user descriptor with the ASCII character string specified in the original User_Desc_set command.

If the NWKAddrOfInterest field does not match the network address of the remote device and it is an end device, it shall set the Status field to INV_REQUESTTYPE. If the NWKAddrOfInterest field does not match the network address of the remote device and it is the coordinator or a router, it shall determine whether the NWKAddrOfInterest field matches the network address of one of its children. If the NWKAddrOfInterest field does not match the network address of one of the children of the remote device, it shall set the Status field to DEVICE_NOT_FOUND. If the NWKAddrOfInterest matches the network address of one of the children of the remote device, it shall determine whether a user descriptor for that device is available. If a user descriptor is not available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to NO_DESCRIPTOR. If a user descriptor is available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to SUCCESS and configure the user descriptor with the ASCII character string specified in the original User_Desc_set command.

2.4.4.2.11.2 Effect on Receipt

The local device is notified of the results of its attempt to configure the user descriptor on a remote device.

2.4.4.2.12 Discovery_Cache_rsp

The Discovery_Cache_rsp command (ClusterID=0x8012) shall be formatted as illustrated in Figure 2.74.

Figure 2.74 Format of the Discovery_Cache_rsp Command Frame

Octets: 1
Status

Table 2.103 specifies the fields of the Discovery_Cache_rsp Command Frame.

Table 2.103 Fields of the Discovery_Cache_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS	The status of the Discovery_Cache_req command.

2.4.4.2.12.1 When Generated

The Discovery_Cache_rsp is generated by Primary Discovery Cache devices receiving the Discovery_Cache_req. Remote Devices which are not Primary Discovery Cache devices (as designated in its Node Descriptor) should not respond to the Discovery_Cache_req command.

2.4.4.2.12.2 Effect on Receipt

Upon receipt of the Discovery_Cache_rsp, the Local Device determines if a SUCCESS status was returned. If no Discovery_Cache_rsp messages were returned from the original Discovery_Cache_req command, then the Local Device should increase the radius for the request to locate Primary Discovery Cache devices beyond the radius supplied in the previous request. If a SUCCESS status is returned, the Local Device should use the Discovery_Store_req, targeted to the Remote Device supplying the response, to determine whether sufficient discovery cache storage is available.

2.4.4.2.13 Discovery_store_rsp

The Discovery_store_rsp command (ClusterID=0x8016) shall be formatted as illustrated in Figure 2.75.

Figure 2.75 Format of the Discovery_store_rsp Command Frame

Octets: 1
Status

Table 2.104 specifies the fields of the Discovery_store_rsp command frame.

Table 2.104 Fields of the Discovery_store_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INSUFFICIENT_SPACE or NOT_SUPPORTED	The status of the Discovery_store_req command.

2.4.4.2.13.1 When Generated

The Discovery_store_rsp is provided to notify a Local Device of the request status from a Primary Discovery Cache device. Included in the response is a status code to notify the Local Device whether the request is successful (the Primary Cache Device has space to store the discovery cache data for the Local Device), whether the request is unsupported (meaning the Remote Device is not a Primary Discovery Cache device), or insufficient space exists.

2.4.4.2.13.2 Effect on Receipt

Upon receipt, the Local Device shall determine whether the response status indicates that the Remote Device is not a Primary Cache Device as indicated by a NOT_SUPPORTED status. If a NOT_SUPPORTED status is returned, the Local Device should process any other Discovery_store_rsp devices from other Remote Devices or re-perform the Discovery_Cache_req to determine the address of another Primary Discovery Cache device (eliminating the address of the Remote Device that responded with NOT_SUPPORTED if it responds again to the Discovery_Cache_req). If an INSUFFICIENT_SPACE status is returned, the Local Device should also process any other Discovery_store_rsp and re-perform the Discovery_Cache_req if none of the responses indicate SUCCESS (with the radius field increased to include more Remote Devices). If a
SUCCESS status is returned, the Local Device shall upload its discovery cache information to the Remote Device via the Node_Desc_store_req, Power_Desc_store_req, Active_EP_store_req, and Simple_Desc_store_req.

2.4.4.2.14 Node_Desc_store_rsp

The Node_Desc_store_rsp command (ClusterID=0x8017) shall be formatted as illustrated in Figure 2.76.

Figure 2.76 Format of the Node_Desc_store_rsp Command Frame

Octets: 1
Status

Table 2.105 specifies the fields of the Node_Desc_store_rsp command frame.

Table 2.105 Fields of the Node_Desc_store_rsp Command

Name	**Type**	**Valid Range**	**Description**
Status	Integer	SUCCESS, INSUFFICIENT_SPACE, NOT_PERMITTED or NOT_SUPPORTED	The status of the Node_store_rsp command.

2.4.4.2.14.1 When Generated

The Node_store_rsp is provided to notify a Local Device of the request status from a Primary Discovery Cache device. Included in the response is a status code to notify the Local Device whether the request is successful (the Primary Cache Device has space to store the discovery cache data for the Local Device), whether the request is not supported (meaning the Remote Device is not a Primary Discovery Cache device), or insufficient space exists.

2.4.4.2.14.2 Effect on Receipt

Upon receipt, the Local Device shall determine whether the response status indicates that the Remote Device is not a Primary Cache Device as indicated by a NOT_SUPPORTED status. If a NOT_SUPPORTED status is returned, the Local Device should re-perform discovery of the Primary Discovery Cache device. If a NOT_PERMITTED status is returned, the local device must first issue a Discovery_store_req with a returned SUCCESS status. If an INSUFFICIENT_SPACE status is returned, the Local Device shall also send the Remote Device a Remove_node_cache_req. If a SUCCESS status is returned, the Local Device should continue to upload its remaining discovery cache information to the Remote Device via the Power_Desc_store_req, Active_EP_store_req, and Simple_Desc_store_req.

2.4.4.2.15 Power_Desc_store_rsp

The Power_Desc_store_rsp command (ClusterID=0x8018) shall be formatted as illustrated in Figure 2.77.

Figure 2.77 Format of the Power_Desc_store_rsp Command Frame

Octets: 1	8	Variable
Status	IEEEAddr	PowerDescriptor

Table 2.106 specifies the fields of the Power_Desc_store_rsp command frame.

Table 2.106 Fields of the Power_Desc_store_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS INSUFFICIENT_SPACE, NOT_PERMITTED or NOT_SUPPORTED	The status of the Power_store_rsp command.

2.4.4.2.15.1 When Generated

The Power_Desc_store_rsp is provided to notify a Local Device of the request status from a Primary Discovery Cache device. Included in the response is a status code to notify the Local Device whether the request is successful (the Primary Cache Device has space to store the discovery cache data for the Local Device), whether the request is not supported (meaning the Remote Device is not a Primary Discovery Cache device), or insufficient space exists.

2.4.4.2.15.2 Effect on Receipt

Upon receipt, the Local Device shall determine whether the response status indicates that the Remote Device is not a Primary Cache Device as indicated by a NOT_SUPPORTED status. If a NOT_SUPPORTED status is returned, the Local Device should re-perform discovery on the Primary Discovery Cache. If a NOT_PERMITTED status is returned, the local device must first issue a Discovery_store_req with a returned SUCCESS status. If an INSUFFICIENT_SPACE status is returned, the Local Device shall discontinue upload of discovery information, issue a Remove_node_cache_req (citing the Local Device), and cease attempts to upload discovery information to the Remote Device.

If a SUCCESS status is returned, the Local Device should continue to upload its remaining discovery cache information to the Remote Device via the Active_EP_store_req and Simple_Desc_store_req.

2.4.4.2.16 Active_EP_store_rsp

The Active_EP_store_rsp command (ClusterID=0x8019) shall be formatted as illustrated in Figure 2.78.

Figure 2.78 Format of the Active_EP_store_rsp Command Frame

Octets: 1
Status

Table 2.107 specifies the fields of the Active_EP_store_rsp command frame.

Table 2.107 Fields of the Active_EP_store_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INSUFFICIENT_SPACE, NOT_PERMITTED or NOT_SUPPORTED	The status of the Active_EP_store_rsp command.

2.4.4.2.16.1 When Generated

The Active_EP_store_rsp is provided to notify a Local Device of the request status from a Primary Discovery Cache device. Included in the response is a status code to notify the Local Device whether the request is successful (the Primary Cache Device has space to store the discovery cache data for the Local Device), the request is not supported (meaning the Remote Device is not a Primary Discovery Cache device), or insufficient space exists.

2.4.4.2.16.2 Effect on Receipt

Upon receipt, the Local Device shall determine whether the response status indicates that the Remote Device is not a Primary Cache Device as indicated by a NOT_SUPPORTED status. If a NOT_SUPPORTED status is returned, the Local Device should re-perform discovery on the Primary Discovery Cache. If a NOT_PERMITTED status is returned, the local device must first issue a Discovery_store_req with a returned SUCCESS status. If an INSUFFICIENT_SPACE status is returned, the Local Device shall discontinue upload of discovery information, issue a Remove_node_cache_req (citing the Local Device), and cease attempts to upload discovery information to the Remote Device. If a SUCCESS status is returned, the Local Device should continue to upload its remaining discovery cache information to the Remote Device via the Simple_Desc_store_req.

2.4.4.2.17 Simple_Desc_store_rsp

The Simple_Desc_store_rsp command (ClusterID=0x801a) shall be formatted as illustrated in Figure 2.79.

Figure 2.79 Format of the Simple_Desc_store_rsp Command Frame

Octets: 1
Status

Table 2.108 specifies the fields of the Simple_Desc_store_rsp command frame.

Table 2.108 Fields of the Simple_Desc_store_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INSUFFICIENT_SPACE, NOT_PERMITTED or NOT_SUPPORTED	The status of the Simple_desc_store_rsp command.

2.4.4.2.17.1 When Generated

The Simple_Desc_store_rsp is provided to notify a Local Device of the request status from a Primary Discovery Cache device. Included in the response is a status code to notify the Local Device whether the request is successful (the Primary Cache Device has space to store the discovery cache data for the Local Device), the request is not supported (meaning the Remote Device is not a Primary Discovery Cache device), or insufficient space exists.

2.4.4.2.17.2 Effect on Receipt

Upon receipt, the Local Device shall determine whether the response status indicates that the Remote Device is not a Primary Cache Device as indicated by a NOT_SUPPORTED status. If a NOT_SUPPORTED status is returned, the Local Device should re-perform discovery on the Primary Discovery Cache. If a NOT_PERMITTED status is returned, the local device must first issue a Discovery_store_req with a returned SUCCESS status. If an INSUFFICIENT_SPACE status is returned, the Local Device shall discontinue upload of discovery information, issue a Remove_node_cache_req (citing the Local Device), and cease attempts to upload discovery information to the Remote Device. If a SUCCESS status is returned, the Local Device should continue to upload its remaining discovery cache information to the Remote Device via the Simple_Desc_store_req for other endpoints on the Local Device.

2.4.4.2.18 Remove_node_cache_rsp

The Remove_node_cache_rsp command (ClusterID=0x801b) shall be formatted as illustrated in Figure 2.80.

Figure 2.80 Format of the Remove_node_cache_rsp Command Frame

Octets: 1
Status

Table 2.109 specifies the fields of the Remove_node_cache_rsp command frame.

Table 2.109 Fields of the Remove_node_cache_rsp Command

Name	**Type**	**Valid Range**	**Description**
Status	Integer	SUCCESS, DEVICE_NOT_FOUND or NOT_SUPPORTED	The status of the Remove_node_cache_rsp command

2.4.4.2.18.1 When Generated

The Remove_node_cache_rsp is provided to notify a Local Device of the request status from a Primary Discovery Cache device. Included in the response is a status code to notify the Local Device whether the request is successful (the Primary Cache Device has removed the discovery cache data for the indicated device of interest), or the request is not supported (meaning the Remote Device is not a Primary Discovery Cache device).

2.4.4.2.18.2 Effect on Receipt

Upon receipt, the Local Device shall determine whether the response status indicates that the Remote Device is not a Primary Cache Device as indicated by a NOT_SUPPORTED status. If a NOT_SUPPORTED status is returned, the Local Device should re-perform Find_node_cache_req to locate the Primary Discovery Cache device holding the discovery cache information for the indicated device of interest. When the Primary Discovery Cache device holding the discovery information for the device of interest is located, the Local Device should repeat the Remove_node_cache_req to successfully remove the discovery information. If a status of DEVICE_NOT_FOUND is received, this indicates that the Remote Device is the Primary Discovery Cache but does not hold the discovery information for the NWKAddr and the IEEEAddr presented in the request. The Local Device should employ the device discovery commands NWK_Addr_req and IEEE_Addr_req to determine the correct values for NWKAddr and IEEEAddr. If a SUCCESS status is returned, the Local Device has successfully removed the discovery cache information for the indicated device of interest within the request.

2.4.4.2.19 Find_node_cache_rsp

The Find_node_cache_rsp command (ClusterID=0x801c) shall be formatted as illustrated in Figure 2.81.

Figure 2.81 Format of the Find_node_cache_rsp Command Frame

Octets: 2	**2**	**8**
CacheNWKAddress	NWKAddr	IEEEAddr

Table 2.110 specifies the fields of the Find_node_cache_rsp command frame.

Table 2.110 Fields of the Find_node_cache_rsp Command

Name	**Type**	**Valid Range**	**Description**
CacheNWKAddr	Device Address	16-bit NWK Address	NWK Address for the Primary Discovery Cache device holding the discovery information (or the device of interest if it responded to the request directly).
NWKAddr	Device Address	16-bit NWK Address	NWK Address for the device of interest.
IEEEAddr	Device Address	64-bit IEEE Address	IEEE address for the device of interest.

2.4.4.2.19.1 When Generated

The Find_node_cache_rsp is provided to notify a Local Device of the successful discovery of the Primary Discovery Cache device for the given NWKAddr and IEEEAddr fields supplied in the request, or to signify that the device of interest is capable of responding to discovery requests. The Find_node_cache_rsp shall be generated only by Primary Discovery Cache devices holding discovery information for the NWKAddr and IEEEAddr in the request or the device of interest itself and all other Remote Devices shall not supply a response.

2.4.4.2.19.2 Effect on Receipt

Upon receipt, the Local Device shall utilize the CacheNWKAddr as the Remote Device address for subsequent discovery requests relative to the NWKAddr and IEEEAddr in the response.

2.4.4.2.20 Extended_Simple_Desc_rsp

The Extended_Simple_Desc_rsp command (ClusterID=0x801d) shall be formatted as illustrated in Figure 2.82.

Figure 2.82 Format of the Extended_Simple_Desc_rsp Command Frame

Octet:1	2	1	1	1	1	Variable
Status	NWKAddr OfInterest	Endpoint	AppInput ClusterCount	AppOutput ClusterCount	StartIndex	AppClusterList

Table 2.111 specifies the fields of the Extended_Simple_Desc_rsp command frame.

Table 2.111 Fields of the Extended_Simple_Desc_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INVALID_EP, NOT_ACTIVE, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the Extended_Simple_Desc_req command.
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
Endpoint	8 bits	1-254	The endpoint on the destination.
AppInputClusterCount	8 bits	0x00-0xff	The total count of application input clusters in the Simple Descriptor for this endpoint.
AppOutputClusterCount	8 bits	0x00-0xff	The total count of application output clusters in the Simple Descriptor for this endpoint.
StartIndex	8 bits	0x00-0xff	Starting index within the AppClusterList of the response represented by an ordered list of the Application Input Cluster List and Application Output Cluster List from the Simple Descriptor for this endpoint.

Name	Type	Valid Range	Description
AppClusterList			A concatenated, ordered list of the AppInputClusterList and AppOutputClusterList, beginning with StartIndex, from the Simple Descriptor. This field shall only be included in the frame if the status field is equal to SUCCESS.

2.4.4.2.20.1 When Generated

The Extended_Simple_Desc_rsp is generated by a remote device in response to an Extended_Simple_Desc_req directed to the remote device. This command shall be unicast to the originator of the Extended_Simple_Desc_req command.

The remote device shall generate the Extended_Simple_Desc_rsp command using the format illustrated in Table 2.111. The NWKAddrOfInterest field shall match that specified in the original Extended_Simple_Desc_req command. If the endpoint field specified in the original Extended_Simple_Desc_req command does not fall within the correct range specified in Table 2.50, the remote device shall set the Status field to INVALID_EP, set the Endpoint and StartIndex fields to their respective values supplied in the request, and not include the AppClusterList field.

If the NWKAddrOfInterest field matches the network address of the remote device, it shall determine whether the endpoint field specifies the identifier of an active endpoint on the device. If the endpoint field corresponds to an active endpoint, the remote device shall set the Status field to SUCCESS, set the AppClusterList field to the sequence of octets from the concatenated AppInput ClusterList and AppOutputClusterList from the Simple Descriptor (see clause 2.3.2.3), and supply that field as AppClusterList in the response. Note that dependent on the value of StartIndex in the request, the results in AppClusterList may be empty (for example, the StartIndex begins after the sequence of octets given by the concatenation of AppInputClusterList and
AppOutputClusterList). If the endpoint field does not correspond to an active endpoint, the remote device shall set the Status field to NOT_ACTIVE, set the StartIndex field to the value supplied in the request, and not include the AppClusterList field.

2.4.4.2.20.2 Effect on Receipt

On receipt of the Extended_Simple_Desc_rsp command, the recipient is either notified of the requested AppClusterList on the endpoint of the remote device indicated in the original Extended_Simple_Desc_req command or notified of an error. If the Extended_Simple_Desc_rsp command is received with a Status of SUCCESS, the AppClusterList field shall contain the requested portion of the application input cluster list and application output cluster list, starting with the StartIndex. Otherwise, the Status field indicates the error and the AppClusterList field shall not be included.

2.4.4.2.21 Extended_Active_EP_rsp

The Extended_Active_EP_rsp command (ClusterID=0x801e) shall be formatted as illustrated in Figure 2.83.

Figure 2.83 Format of the Extended_Active_EP_rsp Command Frame

Octet: 1	2	1	1	Variable
Status	NWKAddrOfInterest	ActiveEPCount	StartIndex	ActiveEPList

Table 2.112 specifies the fields of the Extended_Active_EP_rsp command frame.

Table 2.112 Fields of the Extended_Active_EP_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, DEVICE_NOT_FOUND, INV_REQUESTTYPE or NO_DESCRIPTOR	The status of the Extended_Active_EP_req command.
NWKAddrOfInterest	Device Address	16-bit NWK address	NWK address for the request.
ActiveEPCount	Integer	0x00-0xff	The count of active endpoints on the Remote Device.
StartIndex	Integer	0x00-0xff	Starting index for the list of active end-points for this report.
ActiveEPList			List of bytes each of which represents an 8-bit endpoint. The list begins with the entry starting with StartIndex and continues until the remaining active endpoints are listed or the ASDU size is exhausted with whole endpoint entries.

2.4.4.2.21.1 When Generated

The Extended_Active_EP_rsp is generated by a remote device in response to an Extended_Active_EP_req directed to the remote device. This command shall be unicast to the originator of the Extended_Active_EP_req command.

The remote device shall generate the Extended_Active_EP_rsp command using the format illustrated in Table 2.111. The NWKAddrOfInterest field shall match that specified in the original Extended_Active_EP_req command. If the NWKAddrOfInterest field matches the network address of the remote device, it shall set the Status field to SUCCESS, set the ActiveEPCount field to the number of active endpoints on that device, and include an ascending list of all the identifiers of the active endpoints, beginning with StartIndex, on that device in the ActiveEPList field and continuing until the remaining list of active endpoints from StartIndex forward is listed or until the ASDU size is exhausted with whole endpoint entries.

If the NWKAddrOfInterest field does not match the network address of the remote device and it is an end device, it shall set the Status field to INV_REQUESTTYPE, set the ActiveEPCount field to 0, and not include the ActiveEPList field. If the NWKAddrOfInterest field does not match the network address of the remote device and it is the coordinator or a router, it shall determine whether the NWKAddrOfInterest field matches the network address of a device it holds in a discovery cache. If the NWKAddrOfInterest field does not match the network address of a device it holds in a discovery cache, it shall set the Status field to DEVICE_NOT_FOUND, set the ActiveEPCount field to 0, and not include the ActiveEPList field. If the NWKAddrOfInterest matches the network address of a device held in a discovery cache on the remote device, it shall determine whether that device has any active endpoints. If the discovery information corresponding to the ActiveEP request has not yet been uploaded to the discovery cache, the remote device shall set the Status field to NO_DESCRIPTOR, set the ActiveEPCount field to 0, and not include the ActiveEPList field. If the cached device has no active endpoints, the remote device shall set the Status field to SUCCESS, set the ActiveEPCount field to 0, and not include the ActiveEPList field. If the cached device has active endpoints, the remote device shall set the Status field to SUCCESS, set the ActiveEPCount field to the number of active endpoints on that device and include an ascending list of all the identifiers of the active endpoints, beginning with StartIndex, on that device in the ActiveEPList field.

2.4.4.2.21.2 Effect on Receipt

On receipt of the Extended_Active_EP_rsp command, the recipient is either notified of the active endpoints of the remote device indicated in the original Extended_Active_EP_req command or notified of an error. If the Extended_Active_EP_rsp command is received with a Status of SUCCESS, the ActiveEPCount field indicates the number of entries in the ActiveEPList field. Otherwise, the Status field indicates the error and the ActiveEPList field shall not be included. The requesting device may need to employ Extended_Active_EP_req multiple times, with different StartIndex values, to receive the full ActiveEPList from the remote device.

2.4.4.2.22 Parent_annce_rsp

The Parent_annce_rsp command (ClusterID = 0x801f) shall be formatted as illustrated in Figure 2.84, and is generated in response to a Parent_annce.

Figure 2.84 Format of the Parent_annce_rsp Command Frame

Octets: 1	**1**	**Variable**	**…**	**Variable**
Status	NumberOfChildren	ChildInfo[0]	…	ChildInfo[n]

Table 2.113 specifies the fields of the Parent_annce_rsp.

Table 2.113 Fields of the Parent_annce_rsp

Name	**Type**	**Valid Range**	**Description**
Status	Integer	SUCCESS, NOT_SUPPORTED, NO_MATCH	The status of the Parent_annce command.
NumberOfChildren	Integer	0-255	The number of ChildInfo structures contained in the message.
ChildInfo	ChildInfo	Variable	The child information. See Table 2.57.

Table 2.57 specifies the contents of the ChildInfo structure. This is the same format as the Parent_annce.

2.4.4.2.22.1 When Generated

Upon receipt of a Parent_annce message, a router shall construct but not yet send a Parent_annce_rsp message with the NumberOfChildren field set to 0. It shall then examine each Extended Address present in the Parent_annce message and search its Neighbor Table for an entry that matches. If a device is found and the Device Type is ZigBee end device (0x02), the router shall do the following.

1. If the Keepalive Received value is TRUE, it shall keep the parent/child relationship in the neighbor table unmodified. It shall then do the following:
    a. Append the ChildInfo structure to the Parent_annce_rsp.
    b. Increment NumberOfChildren by 1.
2. If the Keepalive Received value is FALSE, it shall remove the entry.

If the NumberOfChildren field value is 0, the local device shall discard the previously constructed Parent_Annce_rsp. No response message shall be sent.

If the NumberOfChildren field in the Parent_Annce_rsp is greater than 0, it shall unicast the message to the sender of the Parent_Annce message.

If the device has more ChildInfo entries than fit in a single message, it shall send additional messages. These messages do not have to be jittered or delayed since they are unicast to a single device. Each Parent_annce_rsp shall set the NumberOfChildren field to the number of entries contained within the message.

2.4.4.2.22.2 Effect on Receipt

On receipt of a Parent_annce_rsp, the device shall examine its Neighbor Table for each extended address in the ChildInfo entry and do the following.

i) If the entry matches and the Device Type is Zigbee End Device (0x02), it shall do the following:
    (1) Delete the entry from the Neigbor table.
ii) If the entry does not match, no more processing is performed on this ChildInfo entry.

There is no message generated in response to a Parent_annce_rsp.

2.4.4.3 End Device Bind, Bind, Unbind Bind Management Server Services

Table 2.114 lists the commands supported by Device Profile: End Device Bind, Bind and Unbind Server Services. Each of these primitives will be discussed in the following sections.

Table 2.114 End Device Bind, Unbind and Bind Management Server Services Primitives

End Device Bind, Bind and Unbind Server Service Commands	Server Processing	Server Generation
End_Device_Bind_rsp	O	M
Bind_rsp	O	M

End Device Bind, Bind and Unbind Server Service Commands	Server Processing	Server Generation
Unbind_rsp	O	M
Bind_Register_rsp	O	M
Replace_Device_rsp	O	M
Store_Bkup_Bind_Entry_rsp	O	M
Remove_Bkup_Bind_Entry_rsp	O	M
Backup_Bind_Table_rsp	O	M
Recover_Bind_Table_rsp	O	M
Backup_Source_Bind_rsp	O	M
Recover_Source_Bind_rsp	O	M

For Server Generation requirements see section 2.4.4.1.

2.4.4.3.1 End_Device_Bind_rsp

The End_Device_Bind_rsp command (ClusterID=0x8020) shall be formatted as illustrated in Figure 2.85.

Figure 2.85 Format of the End_Device_Bind_rsp Command Frame

Octets: 1
Status

Table 2.115 specifies the fields of the End_Device_Bind_rsp command frame.

Table 2.115 Fields of the End_Device_Bind_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, INVALID_EP, TIMEOUT, NO_MATCH, or DEVICE_BINDING_TABLE_FULL	The status of the End_Device_Bind_req command

2.4.4.3.1.1 When Generated

The End_Device_Bind_rsp is generated by the ZigBee Coordinator in response to an End_Device_Bind_req and contains the status of the request. This command shall be unicast to each device involved in the bind attempt, using the acknowledged data service.

A Status of NOT_SUPPORTED indicates that the request was directed to a device which was not the ZigBee Coordinator or that the ZigBee Coordinator does not support End Device Binding. Otherwise, End_Device_Bind_req processing is performed as described below, including transmission of the End_Device_Bind_rsp.

2.4.4.3.1.2 Effect on Receipt

When an End_Device_Bind_req is received, determination is made if a Status of NOT_SUPPORTED is warranted as indicated in the previous section. Assuming this device is the ZigBee Coordinator, the supplied endpoint shall be checked to determine whether it falls within the specified range. If it does not, a Status of INVALID_EP shall be returned. If the supplied endpoint falls within the specified range and if this is the first End_Device_Bind_req submitted for evaluation, it shall be stored and a timer started which expires at a pre-configured timeout value. This timeout value shall be a configurable item on the ZigBee Coordinator. If the timer expires before a second End_Device_Bind_req is received, a Status of TIMEOUT is returned. Otherwise, if a second End_Device_Bind_req is received within the timeout window, the two End_Device_Bind_req's are compared for a match. A Status of NO_MATCH indicates that two End_Device_Bind_req were evaluated for a match, but either the ProfileID parameters did not match (see section 2.3.3.2.2) or the ProfileID parameter matched but there was no match of any element of the InClusterList or OutClusterList. A Status of SUCCESS means that a match was detected and a resulting Bind_req will subsequently be directed to the device indicated by the BindingTarget field of the End_Device_Bind_req with matched elements of the OutClusterList.

2.4.4.3.2 Bind_rsp

The Bind_rsp command (ClusterID=0x8021) shall be formatted as illustrated in Figure 2.86.

Figure 2.86 Format of the Bind_rsp Command Frame

Octets: 1
Status

Table 2.116 specifies the fields of the Bind_rsp command frame.

Table 2.116 Fields of the Bind_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, INVALID_EP, TABLE_FULL or NOT_AUTHORIZED	The status of the Bind_req command.

2.4.4.3.2.1 When Generated

The Bind_rsp is generated in response to a Bind_req. If the Bind_req is processed and the Binding Table entry committed on the Remote Device, a Status of SUCCESS is returned. If the Remote Device is not a Primary binding table cache or the SrcAddress, a Status of NOT_SUPPORTED is returned. The Simple Descriptor in the receiving device correlating to the endpoint in the Bind_req shall be looked up. If the Simple Descriptor cannot be found then INVALID_EP shall be returned. If the Simple Descriptor is found, it shall be examined to see if the value of the ClusterID field in the Bind_Req message can be found within the Application output cluster list of the Simple Descriptor. If it cannot be found, then INVALID_EP shall be returned. . If the Remote Device is the Primary binding table cache or SrcAddress but does not have Binding Table resources for the request, a Status of TABLE_FULL is returned.

2.4.4.3.2.2 Effect on Receipt

Upon receipt, error checking is performed on the request as described in the previous section. Assuming the Status is SUCCESS, the parameters from the Bind_req are entered into the Binding Table at the Remote Device via the APSME-BIND.request primitive.

2.4.4.3.3 Unbind_rsp

The Unbind_rsp command (ClusterID=0x8022) shall be formatted as illustrated in Figure 2.87.

Figure 2.87 Format of the Unbind_rsp Command Frame

Octets: 1
Status

Table 2.117 specifies the fields of the Unbind_rsp command frame.

Table 2.117 Fields of the Unbind_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, INVALID_EP, NO_ENTRY or NOT_AUTHORIZED	The status of the Unbind_req command.

2.4.4.3.3.1 When Generated

The Unbind_rsp is generated in response to an Unbind_req. If the Unbind_req is processed and the corresponding Binding Table entry is removed from the Remote Device, a Status of SUCCESS is returned. If the Remote Device is not the ZigBee Coordinator or the SrcAddress, a Status of NOT_SUPPORTED is returned. The supplied endpoint shall be checked to determine whether it falls within the specified range. If it does not, a Status of INVALID_EP shall be returned. If the Remote Device is the ZigBee Coordinator or SrcAddress but does not have a Binding Table entry corresponding to the parameters received in the request, a Status of NO_ENTRY is returned.

2.4.4.3.3.2 Effect on Receipt

Upon receipt, error checking is performed on the response. If the status is SUCCESS, the device has successfully removed the binding entry for the parameters specified in the Unbind_req.

2.4.4.3.4 Bind_Register_rsp

The Bind_Register_rsp command (ClusterID=0x8023) shall be formatted as illustrated in Figure 2.88.

Figure 2.88 Format of the Bind_Register_rsp Command Frame

Octets: 1	**2**	**2**	**Variable**
Status	BindingTableEntries	BindingTableListCount	BindingTableList

Table 2.118 specifies the fields of the Bind_Register_rsp command frame.

Table 2.118 Fields of the Bind_Register_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, TABLE_FULL	The status of the Bind_Register_reg command.
BindingTableEntries	Integer	0x0000 - 0ffff	Number of binding table entries for the requesting device held by the primary binding table cache.
BindingTableListCount	Integer	0x0000 - 0xffff	Number of source binding table entries contained in this response.
BindingTableList	List of source binding descriptors	This list shall contain the number of elements given by the BindingTableListCount	A list of source binding.

2.4.4.3.4.1 When Generated

The Bind_Register_rsp is generated from a primary binding table cache device in response to a Bind_Register_req and contains the status of the request. This command shall be unicast to the requesting device.

If the device receiving the Bind_Register_req is not a primary binding table cache a Status of NOT_SUPPORTED is returned. If its list of devices which choose to store their own binding table entries is full, a status of TABLE_FULL is returned. In these error cases, BindingTableEntries and BindingTableListCount shall be zero and BindingTableList shall be empty. A Status of SUCCESS indicates that the requesting device has been successfully registered.

In the successful case, the primary binding table cache device shall search its cache for existing entries whose source address is the same as the parameter supplied in the Bind_Register_req command. The number of such entries is given in the response as BindingTableEntries. The entries are used to generate BindingTableList up to the maximum that can be contained in the response. The actual number of entries is given in the response as BindingTableListCount and may be less than BindingTableEntries if this is too large. In this case (which is expected to be rare) the primary binding table cache device shall use Bind_req commands to send the rest of the entries to the requesting device.

2.4.4.3.4.2 Effect on Receipt

The requesting device is notified of the results of its attempt to register. If successful, it shall store the source binding table entries from the response into its source binding table.

2.4.4.3.5 Replace_Device_rsp

The Replace_Device_rsp command (ClusterID=0x8024) shall be formatted as illustrated in Figure 2.89.

Figure 2.89 Format of the Replace_Device_rsp Command Frame

Octets: 1
Status

Table 2.119 specifies the fields of the Replace_Device_rsp command frame.

Table 2.119 Fields of the Replace_Device_rsp Command

Name	Type	Valid Range	Description
Status	Integer	NOT_SUPPORTED, INV_REQUESTTYPE	The status of the Replace_Device_req command.

2.4.4.3.5.1 When Generated

The Replace_Device_rsp is generated from a primary binding table cache device in response to a Replace_Device_req and contains the status of the request. This command shall be unicast to the requesting device. If the device receiving the Replace_Device_req is not a primary binding table cache, a Status of NOT_SUPPORTED is returned. The primary binding table cache shall search its binding table for entries whose source address and source endpoint, or whose destination address and destination endpoint match OldAddress and OldEndpoint, as described in the text for Replace_Device_req. It shall change these entries to have NewAddress and possibly NewEndpoint. It shall then return a response of SUCCESS.

2.4.4.3.5.2 Effect on Receipt

The requesting device is notified of the status of its Replace_Device_req command.

2.4.4.3.6 Store_Bkup_Bind_Entry_rsp

The Store_Bkup_Bind_Entry_rsp command (ClusterID=0x8025) shall be formatted as illustrated in Figure 2.90.

Figure 2.90 Format of the Store_Bkup_Bind_Entry_rsp Command Frame

Octets: 1
Status

Table 2.120 specifies the fields of the Store_Bkup_Bind_Entry_rsp command frame.

Table 2.120 Fields of the Store_Bkup_Bind_Entry_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, INV_REQUESTTYPE. TABLE_FULL	The status of the Store_Bkup_Bind_Entry_rsp command.

2.4.4.3.6.1 When Generated

The Store_Bkup_Bind_Entry_rsp is generated from a backup binding table cache device in response to a Store_Bkup_Bind_Entry_req from a primary binding table cache, and contains the status of the request. This command shall be unicast to the requesting device. If the remote device is not a backup binding table cache, it shall return a status of NOT_SUPPORTED. If the originator of the request is not recognized as a primary binding table cache, it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall add the binding entry to its binding table and return a status of SUCCESS. If there is no room, it shall return a status of TABLE_FULL.

2.4.4.3.6.2 Effect on Receipt

The requesting device is notified of the status of its attempt to store a bind entry.

2.4.4.3.7 Remove_Bkup_Bind_Entry_rsp

The Remove_Bkup_Bind_Entry_rsp command (ClusterID=0x8026) shall be formatted as illustrated in Figure 2.91.

Figure 2.91 Format of the Remove_Bkup_Bind_Entry_rsp Command Frame

Octets: 1
Status

Table 2.121 specifies the fields of the Remove_Bkup_Bind_Entry_rsp command frame.

Table 2.121 Fields of the Remove_Bkup_Bind_Entry_rsp Command

Name	**Type**	**Valid Range**	**Description**
Status	Integer	SUCCESS, NOT_SUPPORTED, INV_REQUESTTYPE, NO_ENTRY	The status of the Remove_Bkup_Bind_Entry_rsp command.

2.4.4.3.7.1 When Generated

The Remove_Bkup_Bind_Entry_rsp is generated from a backup binding table cache device in response to a Remove_Bkup_Bind_Entry_req from the primary binding table cache and contains the status of the request. This command shall be unicast to the requesting device. If the remote device is not a backup binding table cache, it shall return a status of NOT_SUPPORTED. If the originator of the request is not recognized as a primary binding table cache, it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall delete the binding entry from its binding table and return a status of SUCCESS. If the entry is not found, it shall return a status of NO_ENTRY.

2.4.4.3.7.2 Effect on Receipt

The requesting device is notified of the status of its attempt to remove a bind entry from the backup cache.

2.4.4.3.8 Backup_Bind_Table_rsp

The Backup_Bind_Table_rsp command (ClusterID=0x8027) shall be formatted as illustrated in Figure 2.92.

Figure 2.92 Format of the Backup_Bind_Table_rsp Command Frame

Octets: 1	2
Status	EntryCount

Table 2.122 specifies the fields of the Backup_Bind_Table_rsp command frame.

Table 2.122 Fields of the Backup_Bind_Table_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, TABLE_FULL, INV_REQUESTTYPE	The status of the Backup_Bind_Table_rsp command.
EntryCount	Integer	0x0000 - 0xFFFF	The number of entries in the backup binding table.

2.4.4.3.8.1 When Generated

The Backup_Bind_Table_rsp is generated from a backup binding table cache device in response to a Backup_Bind_Table_req from a primary binding table cache and contains the status of the request. This command shall be unicast to the requesting device. If the remote device is not a backup binding table cache, it shall return a status of NOT_SUPPORTED. If the originator of the request is not recognized as a primary binding table cache, it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall overwrite the binding entries in its binding table starting with StartIndex and continuing for BindingTableListCount entries. If this exceeds its table size, it shall fill in as many entries as possible and return a status of TABLE_FULL and the EntryCount parameter will be the number of entries in the table. Otherwise, it shall return a status of SUCCESS and EntryCount will be equal to StartIndex + BindingTableListCount from Backup_Bind_Table_req.

2.4.4.3.8.2 Effect on Receipt

The requesting device is notified of the status of its attempt to store a binding table.

2.4.4.3.9 Recover_Bind_Table_rsp

The Backup_Bind_Table_rsp command (ClusterID=0x8028) shall be formatted as illustrated in Figure 2.93.

Figure 2.93 Format of the Backup_Bind_Table_rsp Command Frame

Octets: 1	2	2	2	Variable
Status	BindingTableEntries	StartIndex	BindingTableListCount	BindingTableList

Table 2.123 specifies the fields of the Recover_Bind_Table_rsp command frame.

Table 2.123 Fields of the Recover_Bind_Table_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, INV_REQUESTTYPE, NO_ENTRY	The status of the Recover_Bind_Table_rsp command.
BindingTableEntries	Integer	0x0000 - 0xffff	Total number of binding table entries in the backup binding cache.
StartIndex	Integer	0x0000 - 0xffff	Starting index within the binding table to begin reporting for the binding table list.
BindingTableListCount	Integer	0x0000 - 0xffff	Number of binding entries included within BindingTableList.
BindingTableList	Integer	The list shall contain the number of elements given by BindingTableListCount	A list of descriptors, beginning with the StartIndex element and continuing for BindingTableListCount of elements in the backup binding table cache.

2.4.4.3.9.1 When Generated

The Recover_Bind_Table_rsp is generated from a backup binding table cache device in response to a Recover_Bind_Table_req from a primary binding table cache and contains the status of the request. This command shall be unicast to the requesting device. If the responding device is not a backup binding table cache, it shall return a status of NOT_SUPPORTED. If the originator of the request is not recognized as a primary binding table cache it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall prepare a list of binding table entries from its backup beginning with StartIndex. It will fit in as many entries as possible into a Recover_Bind_Table_rsp command and return a status of SUCCESS. If StartIndex is more than the number of entries in the Binding table, a status of NO_ENTRY is returned. For a successful response, BindingTableEntries is the total number of entries in the backup binding table, and BindingTableListCount is the number of entries which is being returned in the response.

2.4.4.3.9.2 Effect on Receipt

The requesting device is notified of the status of its attempt to restore a binding table.

2.4.4.3.10 Backup_Source_Bind_rsp

The Backup_Source_Bind_rsp command (ClusterID=0x8029) shall be formatted as illustrated in Figure 2.94.

Figure 2.94 Format of the Backup_Source_Bind_rsp Command Frame

Octets: 1
Status

Table 2.124 specifies the fields of the Backup_Source_Bind_rsp command frame.

Table 2.124 Fields of the Backup_Source_Bind_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, TABLE_FULL, INV_REQUESTTYPE	The status of the Backup_Source_Bind_rsp command.

2.4.4.3.10.1 When Generated

The Backup_Source_Bind_rsp is generated from a backup binding table cache device in response to a Backup_Source_Bind_req from a primary binding table cache and contains the status of the request. This command shall be unicast to the requesting device. If the remote device is not a backup binding table cache, it shall return a status of NOT_SUPPORTED. If the originator of the request is not recognized as a primary binding table cache, it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall overwrite its backup source binding table starting with StartIndex and continuing for BindingTableListCount entries. If this exceeds its table size, it shall return a status of TABLE_FULL. Otherwise it shall return a status of SUCCESS.

2.4.4.3.10.2 Effect on Receipt

The requesting device is notified of the status of its attempt to backup the source binding table.

2.4.4.3.11 Recover_Source_Bind_rsp

The Recover_Source_Bind_rsp command (ClusterID=0x802a) shall be formatted as illustrated in Figure 2.95.

Figure 2.95 Format of the Recover_Source_Bind_rsp Command Frame

Octets: 1	2	2	2	Variable
Status	SourceTableEntries	StartIndex	SourceTableListCount	SourceTableList

Table 2.125 specifies the fields of the Recover_Source_Bind_rsp command frame.

Table 2.125 Fields of the Recover_Source_Bind_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, TABLE_FULL, INV_REQUESTTYPE	The status of the Recover_Source_Bind_rsp command.
SourceTableEntries	Integer	0x0000 - 0xffff	Total number of source table entries in the backup binding cache.
StartIndex	Integer	0x0000 - 0xffff	Starting index within the source table to begin reporting for the source table list.

Name	Type	Valid Range	Description
SourceTableListCount	Integer	0x0000 - 0xffff	Number of source table entries included within SourceTableList.
SourceTableList	List of source descriptors	The list shall contain the number of elements given by SourceTableListCount	A list of descriptors, beginning with the StartIndex element and continuing for SourceTableListCount of elements in the backup source table cache (consisting of IEEE addresses).

2.4.4.3.11.1 When Generated

The Recover_Source_Bind_rsp is generated from a backup binding table cache device in response to a Recover_Source_Bind_req from a primary binding table cache and contains the status of the request. This command shall be unicast to the requesting device. If the responding device is not a backup binding table cache, it shall return a status of NOT_SUPPORTED. If the originator of the request is not recognized as a primary binding table cache, it shall return a status of INV_REQUESTTYPE. Otherwise, the backup binding table cache shall prepare a list of binding table entries from its backup beginning with StartIndex. It will fit in as many entries as possible into a Recover_Source_Bind_rsp command and return a status of SUCCESS. If StartIndex is more than the number of entries in the Source table, a status of NO_ENTRY is returned. For a successful response, SourceTableEntries is the total number of entries in the backup source table, and SourceTableListCount is the number of entries which is being returned in the response.

2.4.4.3.11.2 Effect on Receipt

The requesting device is notified of the status of its attempt to restore a source binding table.

2.4.4.4 Network Management Server Services

Table 2.126 lists the commands supported by Device Profile: Network Management Server Services. Each of these commands will be discussed in the following sections.

Table 2.126 Network Management Server Service Commands

Network Management Server Service Command	Server Processing	Server Generation
Mgmt_NWK_Disc_rsp	O	M
Mgmt_Lqi_rsp	M[2]	M
Mgmt_Rtg_rsp	O	M
Mgmt_Bind_rsp	O	M
Mgmt_Leave_rsp	O	M
Mgmt_Direct_Join_rsp	O	M

[2] CCB 1604

Network Management Server Service Command	Server Processing	Server Generation
Mgmt_Permit_Joining_rsp	M	M
Mgmt_Cache_rsp	O	M
Mgmt_NWK_Update_notify	O	M

For Server Generation requirements see section 2.4.4.1.

2.4.4.4.1 Mgmt_NWK_Disc_rsp

The Mgmt_NWK_Disc_rsp command (ClusterID=0x8030) shall be formatted as illustrated in Figure 2.96.

Figure 2.96 Format of the Mgmt_NWK_Disc_rsp Command Frame

Octets: 1	1	1	1	Variable
Status	NetworkCount	StartIndex	NetworkListCount	NetworkList

Table 2.127 specifies the fields of the Mgmt_NWK_Disc_rsp command frame.

Table 2.127 Fields of the Mgmt_NWK_Disc_rsp Command

Name	Type	Valid Range	Description
Status	Integer	NOT_SUPPORTED or any status code returned from the NLME-NETWORK-DISCOVERY.request primitive.	The status of the Mgmt_NWK_Disc_req command.
NetworkCount	Integer	0x00-0xff	The total number of networks reported by the NLME-NETWORK-DISCOVERY.confirm.
StartIndex	Integer	0x00-0xff	The starting point in the NetworkList from the NLME-NETWORK-DISCOVERY.confirm where reporting begins for this response.
NetworkList-Count	Integer	0x00-0xff	The number of network list descriptors reported within this response.

Name	Type	Valid Range	Description
NetworkList	List of Network Descriptors	The list shall contain the number of elements given by the NetworkListCount parameter.	A list of descriptors, one for each of the networks discovered, beginning with the StartIndex element and continuing for NetworkListCount, of the elements returned by the NLME-NETWORK-DISCOVERY.confirm primitive. Each entry shall be formatted as illustrated in Table 2.128.

Table 2.128 NetworkList Record Format

Name	Size (Bits)	Valid Range	Description
ExtendedPanID	64	A 64-bit PAN identifier	The 64-bit extended PAN identifier of the discovered network.
LogicalChannel	8	Selected from the available logical channels supported by the PHY (see [B1])	The current logical channel occupied by the network.
StackProfile	4	0x0-0xf	A ZigBee stack profile identifier indicating the stack profile in use in the discovered network.
ZigBeeVersion	4	0x0-0xf	The version of the ZigBee protocol in use in the discovered network.
BeaconOrder	4	0x0-0xf	This specifies how often the MAC sub-layer beacon is to be transmitted by a given device on the network. For a discussion of MAC sub-layer beacon order see [B1].
SuperframeOrder	4	0x0-0xf	For beacon-oriented networks, *i.e.*, beacon order < 15, this specifies the length of the active period of the superframe. For a discussion of MAC sub-layer superframe order see [B1].
PermitJoining	1	TRUE or FALSE	A value of TRUE indicates that at least one ZigBee router on the network currently permits joining, *i.e.*, its NWK has been issued an NLME-PERMIT-JOINING primitive and the time limit, if given, has not yet expired.
Reserved	7		Each of these bits shall be set to 0.

2.4.4.4.1.1 When Generated

The Mgmt_NWK_Disc_rsp is generated in response to an Mgmt_NWK_Disc_req. If this management command is not supported, a status of NOT_SUPPORTED shall be returned and all parameter fields after the Status field shall be omitted. Otherwise, the Remote Device shall implement the following process.

Upon receipt of and after support for the Mgmt_NWK_Disc_req has been verified, the Remote Device shall issue an NLME-NETWORK-DISCOVERY.request primitive using the ScanChannels and ScanDuration parameters, supplied in the Mgmt_NWK_Disc_req command. Upon receipt of the NLME-NETWORK-DISCOVERY.confirm primitive, the Remote Device shall report the results, starting with the StartIndex element, via the Mgmt_NWK_Disc_rsp command. The NetworkList field shall contain whole NetworkList records, formatted as specified in Table 2.128, until the limit on MSDU size, i.e., *aMaxMACFrameSize* (see [B1]), is reached. The number of results reported shall be set in the NetworkListCount.

2.4.4.4.1.2 Effect on Receipt

The local device is notified of the results of its attempt to perform a remote network discovery.

2.4.4.4.2 Mgmt_Lqi_rsp

The Mgmt_Lqi_rsp command (ClusterID=0x8031) shall be formatted as illustrated in Figure 2.97.

Figure 2.97 Format of the Mgmt_Lqi_rsp Command Frame

Octets: 1	**1**	**1**	**1**	**Variable**
Status	NeighborTable Entries	Start Index	NeighborTable ListCount	NeighborTable List

Table 2.129 specifies the fields of the Mgmt_Lqi_rsp command frame.

Table 2.129 Fields of the Mgmt_Lqi_rsp Command

Name	**Type**	**Valid Range**	**Description**
Status	Integer	NOT_SUPPORTED or any status code returned from the NLME-GET.confirm primitive	The status of the Mgmt_Lqi_req command.
NeighborTableEntries	Integer	0x00-0xff	Total number of Neighbor Table entries within the Remote Device.
StartIndex	Integer	0x00-0xff	Starting index within the Neighbor Table to begin reporting for the NeighborTableList.
NeighborTableListCount	Integer	0x00-0x02	Number of Neighbor Table entries included within NeighborTableList.

NeighborTableList	List of Neighbor Descriptors	The list shall contain the number elements given by the NeighborTableListCount	A list of descriptors, beginning with the StartIndex element and continuing for NeighborTable-ListCount, of the elements in the Remote Device's Neighbor Table including the device address and associated LQI (see Table 2.130 for details).

Table 2.130 NeighborTableList Record Format

Name	Size (Bits)	Valid Range	Description
Extended PAN Id	64	A 64-bit PAN identifier	The 64-bit extended PAN identifier of the neighboring device.
Extended address	64	An extended 64-bit, IEEE address	64-bit IEEE address that is unique to every device. If this value is unknown at the time of the request, this field shall be set to 0xffffffffffffffff.
Network address	16	Network address	The 16-bit network address of the neighboring device.
Device type	2	0x00 - 0x03	The type of the neighbor device: 0x00 = ZigBee coordinator 0x01 = ZigBee router 0x02 = ZigBee end device 0x03 = Unknown
RxOnWhenIdle	2	0x00 - 0x02	Indicates if neighbor's receiver is enabled during idle portions of the CAP: 0x00 = Receiver is off 0x01 = Receiver is on 0x02 = unknown
Relationship	3	0x00 - 0x04	The relationship between the neighbor and the current device: 0x00 = neighbor is the parent 0x01 = neighbor is a child 0x02 = neighbor is a sibling 0x03 = None of the above 0x04 = previous child

Name	Size (Bits)	Valid Range	Description
Reserved	1		This reserved bit shall be set to 0.
Permit joining	2	0x00 - 0x02	An indication of whether the neighbor device is accepting join requests: 0x00 = neighbor is not accepting join requests 0x01 = neighbor is accepting join requests 0x02 = unknown
Reserved	6		Each of these reserved bits shall be set to 0.
Depth	8	0x00 - nwkcMaxDepth	The tree depth of the neighbor device. A value of 0x00 indicates that the device is the ZigBee coordinator for the network.
LQI	8	0x00 - 0xff	The estimated link quality for RF transmissions from this device. See [B1] for discussion of how this is calculated.

2.4.4.4.2.1 When Generated

The Mgmt_Lqi_rsp is generated in response to an Mgmt_Lqi_req. If this management command is not supported, a status of NOT_SUPPORTED shall be returned and all parameter fields after the Status field shall be omitted. Otherwise, the Remote Device shall implement the following processing.

Upon receipt of and after support for the Mgmt_Lqi_req has been verified, the Remote Device shall perform an NLME-GET.request (for the *nwkNeighborTable* attribute) and process the resulting neighbor table (obtained via the NLME-GET.confirm primitive) to create the Mgmt_Lqi_rsp command. If *nwkNeighborTable* was successfully obtained but one or more of the fields required in the NeighborTableList record (see Table 2.130) are not supported (as they are optional), the Mgmt_Lqi_rsp shall return a status of NOT_SUPPORTED and all parameter fields after the Status field shall be omitted. Otherwise, the Mgmt_Lqi_rsp command shall contain the same status that was contained in the NLME-GET.confirm primitive and if this was not SUCCESS, all parameter fields after the status field shall be omitted.

From the *nwkNeighborTable* attribute, the neighbor table shall be accessed, starting with the index specified by StartIndex, and shall be moved to the NeighborTableList field of the Mgmt_Lqi_rsp command. The entries reported from the neighbor table shall be those, starting with StartIndex and including whole NeighborTableList records (see Table 2.130) until the limit on MSDU size, i.e., *aMaxMACFrameSize* (see [B1]), is reached. Within the Mgmt_Lqi_Rsp command, the NeighborTableEntries field shall represent the total number of Neighbor Table entries in the Remote Device. The parameter NeighborTableListCount shall be the number of entries reported in the NeighborTableList field of the Mgmt_Lqi_rsp command.

The extended address, device type, RxOnWhenIdle, and permit joining fields have “unknown” values which shall be returned where the values are not available.

2.4.4.4.2.2 Effect on Receipt

The local device is notified of the results of its attempt to obtain the neighbor table.

2.4.4.4.3 Mgmt_Rtg_rsp

The Mgmt_Rtg_rsp command (ClusterID=0x8032) shall be formatted as illustrated in Figure 2.98.

Figure 2.98 Format of the Mgmt_Rtg_rsp Command Frame

Octets: 1	1	1	1	Variable
Status	RoutingTable Entries	Start Index	RoutingTable ListCount	RoutingTable List

Table 2.131 specifies the fields of the Mgmt_Rtg_rsp command frame.

Table 2.131 Fields of the Mgmt_Rtg_rsp Command

Name	Type	Valid Range	Description
Status	Integer	NOT_SUPPORTED or any status code returned from the NLME-GET.confirm primitive	The status of the Mgmt_Rtg_req command.
RoutingTableEntries	Integer	0x00-0xff	Total number of Routing Table entries within the Remote Device.
StartIndex	Integer	0x00-0xff	Starting index within the Routing Table to begin reporting for the RoutingTableList.
RoutingTableListCount	Integer	0x00-0xff	Number of Routing Table entries included within RoutingTableList.
RoutingTableList	List of Routing Descriptors	The list shall contain the number elements given by the RoutingTableListCount	A list of descriptors, beginning with the StartIndex element and continuing for RoutingTableListCount, of the elements in the Remote Device's Routing Table (see Table 2.132 for details).

Table 2.132 RoutingTableList Record Format

Name	Size (Bits)	Valid Range	Description
Destination address	16	The 16-bit network address of this route.	Destination address.
Status	3	The status of the route.	0x0=ACTIVE. 0x1=DISCOVERY_UNDERWAY. 0x2=DISCOVERY_FAILED. 0x3=INACTIVE. 0x4=VALIDATION_UNDERWAY 0x5-0x7=RESERVED.
Memory Constrained	1		A flag indicating whether the device is a memory constrained concentrator.
Many-to-one	1		A flag indicating that the destination is a concentrator that issued a many-to-one request.
Route record required	1		A flag indicating that a route record command frame should be sent to the destination prior to the next data packet.
Reserved	2		
Next-hop address	16	The 16-bit network address of the next hop on the way to the destination.	Next-hop address.

2.4.4.4.3.1 When Generated

The Mgmt_Rtg_rsp is generated in response to an Mgmt_Rtg_req. If this management command is not supported, a status of NOT_SUPPORTED shall be returned and all parameter fields after the Status field shall be omitted. Otherwise, the Remote Device shall implement the following processing.

Upon receipt of and after support for the Mgmt_Rtg_req has been verified, the Remote Device shall perform an NLME-GET.request (for the *nwkRouteTable* attribute) and process the resulting NLME-GET.confirm (containing the *nwkRouteTable* attribute) to create the Mgmt_Rtg_rsp command. The Mgmt_Rtg_rsp command shall contain the same status that was contained in the NLME-GET.confirm primitive and if this was not SUCCESS, all parameter fields after the status field shall be omitted.

From the *nwkRouteTable* attribute, the routing table shall be accessed, starting with the index specified by StartIndex, and moved to the RoutingTableList field of the Mgmt_Rtg_rsp command. The entries reported from the routing table shall be those, starting with StartIndex and including whole RoutingTableList records (see Table 2.132) until MSDU size limit, i.e., *aMaxMACFrameSize* (see [B1]), is reached. Within the Mgmt_Rtg_Rsp command, the RoutingTableEntries field shall represent the total number of Routing Table entries in the Remote Device. The RoutingTableListCount field shall be the number of entries reported in the RoutingTableList field of the Mgmt_Rtg_req command.

2.4.4.4.3.2 Effect on Receipt

The local device is notified of the results of its attempt to obtain the routing table.

2.4.4.4.4 Mgmt_Bind_rsp

The Mgmt_Bind_rsp command (ClusterID=0x8033) shall be formatted as illustrated in Figure 2.99.

Figure 2.99 Format of the Mgmt_Bind_rsp Command Frame

Octets: 1	1	1	1	Variable
Status	BindingTable Entries	Start Index	BindingTable ListCount	BindingTable List

Table 2.133 specifies the fields of the Mgmt_Bind_rsp command frame.

Table 2.133 Fields of the Mgmt_Bind_rsp Command

Name	Type	Valid Range	Description
Status	Integer	NOT_SUPPORTED or any status code returned from the APSME-GET.confirm primitive	The status of the Mgmt_Bind_req command.
BindingTableEntries	Integer	0x00-0xff	Total number of Binding Table entries within the Remote Device.
StartIndex	Integer	0x00-0xff	Starting index within the Binding Table to begin reporting for the BindingTableList.
BindingTableListCount	Integer	0x00-0xff	Number of Binding Table entries included within BindingTableList.
BindingTableList	List of Binding Descriptors	The list shall contain the number elements given by the BindingTableListCount	A list of descriptors, beginning with the StartIndex element and continuing for BindingTableListCount, of the elements in the Remote Device's Binding Table (see Table 2.134 for details).

Table 2.134 BindingTableList Record Format

Name	Size (Bits)	Valid Range	Description
SrcAddr	64	A valid 64-bit IEEE address	The source IEEE address for the binding entry.
SrcEndpoint	8	0x01 - 0xfe	The source endpoint for the binding entry.
ClusterId	16	0x0000 - 0xffff	The identifier of the cluster on the source device that is bound to the destination device.
DstAddrMode	8	0x00 - 0xff	The addressing mode for the destination address. This field can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddr and DstEndpoint not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddr and DstEndp present 0x04 – 0xff = reserved
DstAddr	16/64	As specified by the DstAddr-Mode field	The destination address for the binding entry.
DstEndpoint	0/8	0x01 - 0xff	This field shall be present only if the DstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.

2.4.4.4.4.1 When Generated

The Mgmt_Bind_rsp is generated in response to a Mgmt_Bind_req. If this management command is not supported, a status of NOT_SUPPORTED shall be returned and all parameter fields after the Status field shall be omitted. Otherwise, the Remote Device shall implement the following processing.

Upon receipt of and after support for the Mgmt_Bind_req has been verified, the Remote Device shall perform an APSME-GET.request (for the *apsBindingTable* attribute) and process the resulting APSME-GET.confirm (containing the *apsBindingTable* attribute) to create the Mgmt_Bind_rsp command. The Mgmt_Bind_rsp command shall contain the same status that was contained in the APSME-GET.confirm primitive and if this was not SUCCESS, all parameter fields after the status field shall be omitted.

From the *apsBindingTable* attribute, the binding table shall be accessed, starting with the index specified by StartIndex, and moved to the BindingTableList field of the Mgmt_Bind_rsp command. The entries reported from the binding table shall be those, starting with StartIndex and including whole BindingTableList records (see Table 2.134) until the MSDU size limit, i.e., *aMaxMACFrameSize* (see [B1]), is reached. Within the Mgmt_Bind_Rsp command, the BindingTableEntries field shall represent the total number of Binding Table entries in the Remote Device. The BindingTableListCount field shall be the number of entries reported in the BindingTableList field of the Mgmt_Bind_req command.

2.4.4.4.4.2 Effect on Receipt

The local device is notified of the results of its attempt to obtain the binding table.

2.4.4.4.5 Mgmt_Leave_rsp

The Mgmt_Leave_rsp command (ClusterID=0x8034) shall be formatted as illustrated in Figure 2.100.

Figure 2.100 Format of the Mgmt_Leave_rsp Command Frame

Octets: 1
Status

Table 2.135 specifies the fields of the Mgmt_Leave_rsp command frame.

Table 2.135 Fields of the Mgmt_Leave_rsp Command

Name	Type	Valid Range	Description
Status	Integer	NOT_SUPPORTED, NOT_AUTHORIZED or any status code returned from the NLME-LEAVE.confirm primitive	The status of the Mgmt_Leave_req command.

2.4.4.4.5.1 When Generated

The Mgmt_Leave_rsp is generated in response to a Mgmt_Leave_req. Stacks certified prior to revision 21 may or may not support this command. If this management command is not supported, a status of NOT_SUPPORTED shall be returned. All stacks certified to revision 21 and later must support this command.

2.4.4.4.5.2 Effect on Receipt

Upon receipt of the Mgmt_leave_rsp the device may parse the Status field to determine whether or not the remote device accepted the leave request.

2.4.4.4.6 Mgmt_Direct_Join_rsp

The Mgmt_Direct_Join_rsp (ClusterID=0x8035) shall be formatted as illustrated in Figure 2.101.

Figure 2.101 Format of the Mgmt_Direct_Join_rsp Command Frame

Octets: 1
Status

Table 2.136 specifies the fields of the Mgmt_Direct_Join_rsp command frame.

Table 2.136 Fields of the Mgmt_Direct_Join_rsp Command

Name	Type	Valid Range	Description
Status	Integer	NOT_SUPPORTED, NOT_AUTHORIZED or any status code returned from the NLME-DIRECT-JOIN.confirm primitive	The status of the Mgmt_Direct_Join_req command.

2.4.4.4.6.1 When Generated

The Mgmt_Direct_Join_rsp is generated in response to a Mgmt_Direct_Join_req. If this management command is not supported, a status of NOT_SUPPORTED shall be returned. Otherwise, the Remote Device shall implement the following processing.

Upon receipt and after support for the Mgmt_Direct_Join_req has been verified, the Remote Device shall execute the NLME-DIRECT-JOIN.request to directly associate the DeviceAddress contained in the Mgmt_Direct_Join_req to the network. The Mgmt_Direct_Join_rsp shall contain the same status that was contained in the NLME-DIRECT-JOIN.confirm primitive.

2.4.4.4.6.2 Effect on Receipt

Upon receipt and after support for the Mgmt_Direct_Join_req has been verified, the Remote Device shall execute the NLME-DIRECT-JOIN.request to directly associate the DeviceAddress contained in the Mgmt_Direct_Join_req to the network.

2.4.4.4.7 Mgmt_Permit_Joining_rsp

The Mgmt_Permit_Joining_rsp command (ClusterID=0x8036) shall be formatted as illustrated in Figure 2.102.

Figure 2.102 Format of the Mgmt_Permit_Joining_rsp Command Frame

Octets: 1
Status

Table 2.137 specifies the fields of the Mgmt_Permit_Joining_rsp command frame.

Table 2.137 Fields of the Mgmt_Permit_Joining_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INVALID_REQUEST, NOT_AUTHORIZED or any status code returned from the NLME-PERMIT-JOINING.confirm primitive	The status of the Mgmt_Permit_Joining_rsp command.

2.4.4.4.7.1 When Generated

The Mgmt_Permit_Joining_rsp is generated in response to a unicast Mgmt_Permit_Joining_req. In the description which follows, note that no response shall be sent if the Mgmt_Permit_Joining_req was received as a broadcast to all routers. If this management command is not permitted by the requesting device, a status of INVALID_REQUEST shall be returned. Upon receipt and after support for Mgmt_Permit_Joining_req has been verified, the Remote Device shall execute the NLME-PERMIT-JOINING.request. The Mgmt_Permit-Joining_rsp shall contain the same status that was contained in the NLME-PERMIT-JOINING.confirm primitive.

2.4.4.4.7.2 Effect on Receipt

The status of the Mgmt_Permit_Joining_req command is notified to the requestor.

2.4.4.4.8 Mgmt_Cache_rsp

The Mgmt_Cache_rsp command (ClusterID=0x8037) shall be formatted as illustrated in Figure 2.103.

Figure 2.103 Format of the Mgmt_Cache_rsp Command Frame

Octets: 1	1	1	1	Variable
Status	DiscoveryCache Entries	StartIndex	DiscoveryCacheListCount	DiscoveryCacheList

Table 2.138 specifies the fields of the Mgmt_Cache_rsp command frame.

Table 2.138 Fields of the Mgmt_Cache_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS or NOT_SUPPORTED	The status of the Mgmt_Cache_rsp command.
DiscoveryCacheEntries	Integer	0x00 - 0xff	DiscoveryCacheEntries.
StartIndex	Integer	0x00 - 0xff	StartIndex.

Name	Type	Valid Range	Description
DiscoveryCacheListCount	Integer	0x00 - 0xff	The list shall contain the number of elements given by the DiscoveryCacheListCount parameter.
DiscoveryCacheList	Integer	List of DiscoveryCache descriptors	A list of descriptors, one for each of the Discovery cache devices registered, beginning with the StartIndex element and continuing for DiscoveryCacheListCount, of the registered devices in the Primary Discovery Cache. Each entry shall be formatted as illustrated in Table 2.139.

Table 2.139 DiscoveryCacheList Record Format

Name	Size (Bits)	Valid Range	Description
Extended Address	64	An extended 64-bit IEEE Address	64-bit IEEE Address of the cached device.
Network Address	16	Network address	The 16-bit network address of the cached device.

2.4.4.4.8.1 When Generated

The Mgmt_Cache_rsp is generated in response to an Mgmt_Cache_req. If this management command is not supported, or the Remote Device is not a Primary Cache Device, a status of NOT_SUPPORTED shall be returned and all parameter fields after the Status field shall be omitted. Otherwise, the Remote Device shall implement the following processing. Upon receipt of the Mgmt_Cache_req and after support for the Mgmt_Cache_req has been verified, the Remote Device shall access an internally maintained list of registered ZigBee End Devices utilizing the discovery cache on this Primary Discovery Cache device. The entries reported shall be those, starting with StartIndex and including whole DiscoveryCacheList records (see Table 2.142) until the limit on MSDU size, i.e., *aMaxMACFrameSize* (see [B1]), is reached. Within the Mgmt_Cache_rsp command, the DiscoveryCacheListEntries field shall represent the total number of registered entries in the Remote Device. The parameter DiscoveryCacheListCount shall be the number of entries reported in the DiscoveryCacheList field of the Mgmt_Cache_rsp command.

2.4.4.4.8.2 Effect on Receipt

The local device is notified of the results of its attempt to obtain the discovery cache list.

2.4.4.4.9 Mgmt_NWK_Update_notify

The Mgmt_NWK_Update_notify command (ClusterID=0x8038) shall be formatted as illustrated in Figure 2.104.

Figure 2.104 Format of the Mgmt_NWK_Update_notify Command Frame

Octets: 1	4	2	2	1	Variable
Status	ScannedChannels	TotalTransmissions	TransmissionFailures	ScannedChannelsListCount	EnergyValues

Table 2.140 specifies the fields of the Mgmt_NWK_Update_notify command frame.

Table 2.140 Fields of the Mgmt_NWK_Update_notify Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INVALID_REQUEST, NOT_SUPPORTED or any status values returned from the PLME-SET,confirm primitive	The status of the Mgmt_NWK_Update_notify command.
ScannedChannels	Bitmap	0x00000000 - 0xffffffff.	List of channels scanned by the request.
TotalTransmissions	Integer	0x0000 -0xffff	Count of the total transmissions reported by the device.
TransmissionFailures	Integer	x0000 -0xffff	Sum of the total transmission failures reported by the device.
ScannedChannelsListCount	Integer	0x00 - 0xff	The list shall contain the number of records contained in the EnergyValues parameter.
EnergyValues	Integer	List of ED values each of which can be in the range of 0x00 - 0xff	The result of an energy measurement made on this channel in accordance with [B1].

2.4.4.4.9.1 When Generated

The Mgmt_NWK_Update_notify is provided to enable ZigBee devices to report the condition on local channels to a network manager. The scanned channel list is the report of channels scanned and it is followed by a list of records, one for each channel scanned, each record including one byte of the energy level measured during the scan, or 0xff if there is too much interference on this channel.

When sent in response to a Mgmt_NWK_Update_req command the status field shall represent the status of the request. When sent unsolicited the status field shall be set to SUCCESS.

2.4.4.4.9.2 Effect on Receipt

The local device is notified of the local channel conditions at the transmitting device, or of its attempt to update network configuration parameters.

2.4.5 ZDP Enumeration Description

This section explains the meaning of the enumerations used in the ZDP. Table 2.141 shows a description of the ZDP enumeration values.

Table 2.141 ZDP Enumerations Description

Enumeration	Value	Description
SUCCESS	0x00	The requested operation or transmission was completed successfully.
-	0x01-0x7f	Reserved.
INV_REQUESTTYPE	0x80	The supplied request type was invalid.
DEVICE_NOT_FOUND	0x81	The requested device did not exist on a device following a child descriptor request to a parent.
INVALID_EP	0x82	The supplied endpoint was equal to 0x00 or 0xff.
NOT_ACTIVE	0x83	The requested endpoint is not described by a simple descriptor.
NOT_SUPPORTED	0x84	The requested optional feature is not supported on the target device.
TIMEOUT	0x85	A timeout has occurred with the requested operation.
NO_MATCH	0x86	The end device bind request was unsuccessful due to a failure to match any suitable clusters.
-	0x87	Reserved.
NO_ENTRY	0x88	The unbind request was unsuccessful due to the coordinator or source device not having an entry in its binding table to unbind.
NO_DESCRIPTOR	0x89	A child descriptor was not available following a discovery request to a parent.
INSUFFICIENT_SPACE	0x8a	The device does not have storage space to support the requested operation.

Enumeration	Value	Description
NOT_PERMITTED	0x8b	The device is not in the proper state to support the requested operation.
TABLE_FULL	0x8c	The device does not have table space to support the operation.
NOT_AUTHORIZED	0x8d	The device has rejected the command due to security restrictions.
DEVICE_BINDING_TABLE_FULL	0x8e	The device does not have binding table space to support the operation.
-	0x8f-0xff	Reserved.

2.4.6 Conformance

When conformance to this Profile is claimed, all capabilities indicated mandatory for this Profile shall be supported in the specified manner (process mandatory). This also applies to optional and conditional capabilities, for which support is indicated, and is subject to verification as part of the ZigBee certification program.

2.5 The ZigBee Device Objects (ZDO)

2.5.1 Scope

This section describes the concepts, structures, and primitives needed to implement a ZigBee Device Objects application on top of a ZigBee Application Support Sub-layer (section 2.2) and ZigBee Network Layer (Chapter 3).

ZigBee Device Objects are applications which employ network and application support layer primitives to implement ZigBee End Devices, ZigBee Routers, and ZigBee Coordinators.

The ZigBee Device Object Profile employs Clusters to describe its primitives. The ZigBee Device Profile Clusters do not employ attributes and are analogous to messages in a message transfer protocol. Cluster identifiers are employed within the ZigBee Device Profile to enumerate the messages employed within ZigBee Device Objects.

ZigBee Device Objects also employ configuration attributes. The configuration attributes within ZigBee Device Objects are attributes set by the application or stack profile. The configuration attributes are also not related to the ZigBee Device Profile, though both the configuration attributes and the ZigBee Device Profile are employed with ZigBee Device Objects.

2.5.2 Device Object Descriptions

The ZigBee Device Objects are an application solution residing within the Application Layer (APL) and above the Application Support Sub-layer (APS) in the ZigBee stack architecture as illustrated in Figure 1.1.

The ZigBee Device Objects are responsible for the following functions:

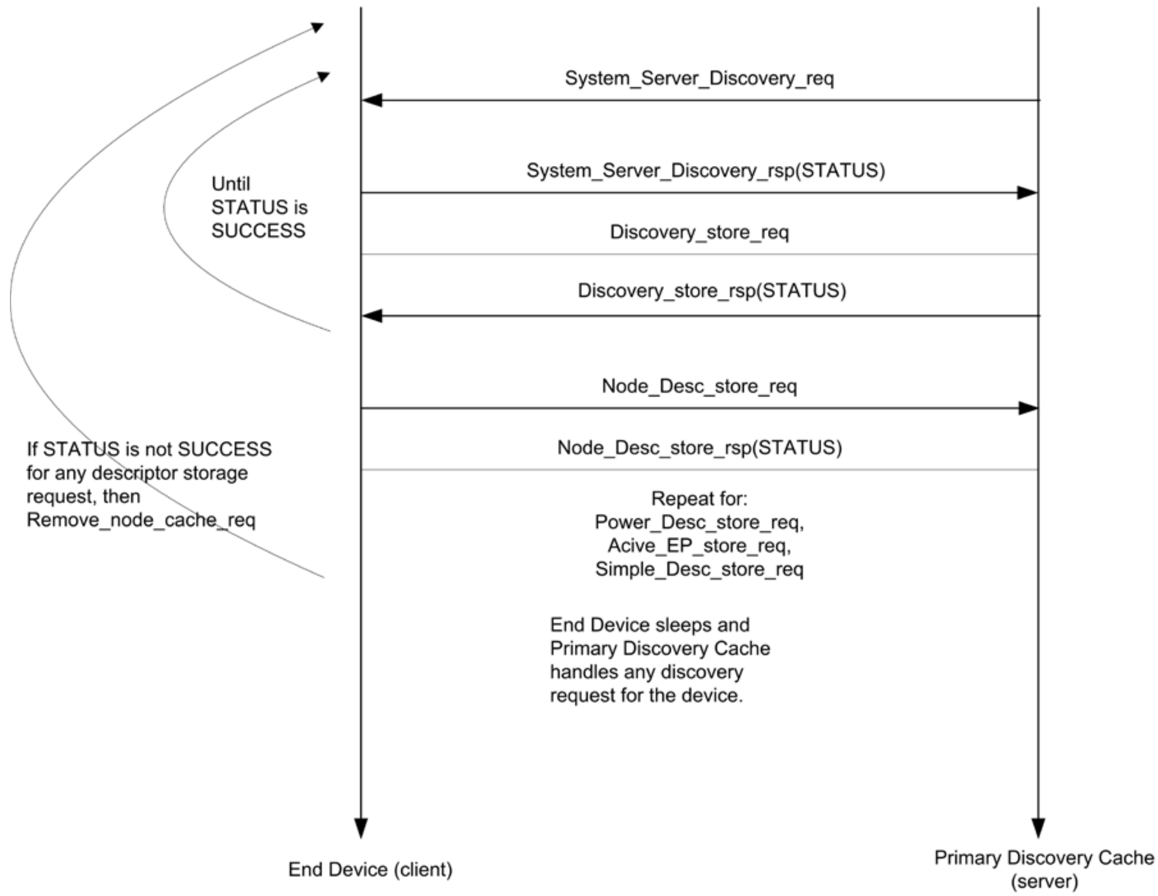
- Initializing the Application Support Sublayer (APS), Network Layer (NWK), Security Service Provider (SSP) and any other ZigBee device layer other than the end applications residing over Endpoints 1-254.
- Assembling configuration information from the end applications to determine and implement the functions described in the following sections.

2.5.2.1 Primary Discovery Cache Device Operation

The Primary Discovery Cache device is designated through configuration of the device and advertisement in the Node Descriptor. The Primary Discovery Cache device operates as a state machine with respect to clients wishing to utilize the services of the Primary Discovery Cache. The following states and operations, as described in Figure 2.105, shall be supported by the Primary Discovery Cache device:

- Undiscovered:
    - The client employs the Find Node Cache request, broadcast to all devices for which macRxOnWhenIdle=TRUE to determine if there is an existing discovery cache entry for the Local Device. If a discovery cache device responds to the request, the Local Device may update the discovery information and shall transition to the Registered state.
    - The client employs the radius limited message System Server Discovery request, broadcast to all devices for which macRxOnWhenIdle = TRUE, to locate a Primary Discovery Cache device within the radius supplied by the request.
- Discovered:
    - The client employs the unicast Discovery store request directed to the Discovery Cache device containing the sizes of the discovery cache information it wishes to store. The Discovery Cache Device will respond with a SUCCESS, INSUFFICIENT_SPACE or NOT_SUPPORTED.
- Registered:
    - This state is reached when a SUCCESS status was received by the client from the Discovery Cache device from a previous Discovery cache request or the Find Node Cache request found a pre-existing discovery cache entry. The client must now upload its discovery information using the Node Descriptor store request, Power Descriptor store request, Active Endpoint store request, and Simple Descriptor store requests to enable the Primary Discovery Cache device to fully respond on its behalf.
- Unregistered:
    - The client (or any other device) may request to be unregistered. The Remove Node Cache request removes the device from the Primary Discovery Cache device. The Primary Cache Device responds to device and service discovery requests for all registered clients it supports. The Find Node Cache request is employed by clients wanting to locate the device and service discovery location for a given device of interest. Note that if the discovery information is held by the device itself, that device must also respond to identify itself as the repository of discovery information. See Figure 2.105 for details on state machine processing for the Primary Discovery Cache device.

Figure 2.105 Primary Discovery Cache State Machine



2.5.2.2 Device and Service Discovery

This function shall support device and service discovery within a single PAN. Additionally, for all ZigBee device types, this function shall perform the following:

- Within each network employing sleeping ZigBee End Devices, some ZigBee Routers (or the ZigBee Coordinator) may be designated as Primary Discovery Cache Devices as described by their Node Descriptor. These Primary Cache Devices are themselves discoverable and provide server services to upload and store discovery information on behalf of sleeping ZigBee End Devices. Additionally, the Primary Cache Devices respond to discovery requests on behalf of the sleeping ZigBee End Devices. Each Primary Discovery Cache Device shall be either a ZigBee Router or the ZigBee Coordinator.
- For ZigBee End Devices which intend to sleep as indicated by:Config_Node_Power, Device and Service Discovery may manage upload and storage of the NWK Address, IEEE Address, Active Endpoints, Simple Descriptors, Node Descriptor, and Power Descriptor onto a Primary Discovery Cache device selected by the ZigBee End Device to permit device and service discovery operations on these sleeping devices.
- For the ZigBee Coordinator and ZigBee Routers designated as Primary Discovery Cache Devices, this function shall respond to discovery requests on behalf of sleeping ZigBee End Devices who have registered and uploaded their discovery information.
- For all ZigBee devices, Device and Service Discovery shall support device and service discovery requests from other devices and permit generation of requests from their local Application Objects. Note that Device and Service Discovery services may be provided by the Primary Discovery Cache devices on behalf of other ZigBee End Devices. In cases where the Primary Discovery Cache Device is the target of the request, the NWKAddrOfInterest or Device of Interest fields shall be filled in the request

and/or response to differentiate the target of the request from the device that is the target of discovery. The following discovery features shall be supported:

- o Device Discovery:
  - — Based on a unicast inquiry of a ZigBee Coordinator or ZigBee Router's IEEE address, the IEEE Address of the requested device plus, optionally, the NWK Addresses of all associated devices shall be returned.
  - — Based on a unicast inquiry of a ZigBee End Device's IEEE address, the IEEE Address of the requested device shall be returned.
  - — Based on a broadcast inquiry (of any broadcast address type) of a ZigBee Coordinator or ZigBee Router's NWK Address with a supplied IEEE Address, the NWK Address of the requested device plus, optionally, the NWK Addresses of all associated devices shall be returned.
  - — Based on a broadcast inquiry (of any broadcast address type) of a ZigBee End Device's NWK Address with a supplied IEEE Address, the NWK Address of the requested device shall be returned. The responding device shall employ APS acknowledged service for the unicast response to the broadcast inquiry.
- o Service Discovery: Based on the following inputs, the corresponding responses shall be supplied:
  - — NWK address plus Active Endpoint query type – Specified device shall return the endpoint number of all applications residing in that device. Should the list of active endpoints exceed the ASDU size and where fragmentation is not supported on the server device, an extended version of the query type is also provided to return the full list through multiple requests.
  - — NWK address or broadcast address (of any broadcast address type) plus Service Match including Profile ID and, optionally, Input and Output Clusters – Specified device matches Profile ID with all active endpoints to determine a match. If no input or output clusters are specified, the endpoints that match the request are returned. If input and/or output clusters are provided in the request, those are matched as well, and any matches are provided in the response with the list of endpoints on the device providing the match. The responding device shall employ APS acknowledged service for the unicast response to the broadcast inquiry. By convention, in cases where the application profile enumerates input clusters and their response output clusters with the same cluster identifier, the application profile shall list only the input cluster within the Simple Descriptor for the purposes of Service Discovery.
  - — NWK address plus Node Descriptor or Power Descriptor query type – Specified device shall return the Node or Power Descriptor for the device.
  - — NWK address, Endpoint Number plus Simple Descriptor query type – Specified address shall return the Simple Descriptor associated with that Endpoint for the device. Should the list of input and/or output clusters exceed the ASDU size capacity to return the Simple Descriptor in a single packet an extended version of the query type is also provided to return the full list through multiple requests.
  - — Optionally, NWK address plus Complex or User Descriptor query type
- ▪ If supported, specified address shall return the Complex or User Descriptor for the device

2.5.2.3 Security Manager

This function determines whether security is enabled or disabled and, if enabled, shall perform the following:

- Transport Key
- Request Key
- Update Device
- Remove Device
- Switch Key

The Security Manager function addresses the Security Services Specification (Chapter 4). The Security Management entity, implemented by APSME primitive calls by ZDO, performs the following:

- Transports the NWK Key from the Trust Center using secured communication with the Trust Center. This step employs the APSME-TRANSPORT-KEY primitive.
- Establishes or transports Link Keys, as required, with specific devices in the network. These steps employ the APSME-TRANSPORT-KEY and/or APSME-REQUEST-KEY primitives.
- Informs the Trust Center of any devices that join the network using the APSME-UPDATE-DEVICE primitives. This function is only performed if the device is a ZigBee router.
- Permits devices to obtain keys from the Trust Center using the APSME-REQUEST-KEY primitives.
- Permits the Trust Center to remove devices from the network using the APSME-REMOVE-DEVICE primitives.
- Permits the Trust Center to switch the active network key using the APSME-SWITCH-KEY primitives.

2.5.2.4 Network Manager

This function shall implement the ZigBee Coordinator, ZigBee Router, or ZigBee End Device logical device types according to configuration settings established either via a programmed application or during installation. If the device type is a ZigBee Router or ZigBee End Device, this function shall provide the ability to select an existing PAN to join and implement procedures which permit the device to rejoin if network communication is lost. If the device type is a ZigBee Coordinator or ZigBee Router, this function shall provide the ability to select an unused channel for creation of a new PAN. Note that it is possible to deploy a network without a device pre-designated as ZigBee Coordinator where the first Full Function Device (FFD) activated assumes the role of ZigBee Coordinator. The following description covers processing addressed by Network Management:

- Permits specification of a channel list for network scan procedures. Default is to specify use of all channels in the selected band of operation.
- Manages network scan procedures to determine neighboring networks and the identity of their ZigBee coordinators and routers.
- Permits selection of a channel to start a PAN (ZigBee Coordinator) or selection of an existing PAN to join (ZigBee Router or ZigBee End Device).
- Supports orphaning and extended procedures to rejoin the network, including support for intra_PAN portability.
- May support direct join. For ZigBee Coordinators and ZigBee Routers, a local version of direct join may be supported to enable the device to join via the orphaning or rejoin procedures.
- May support Management Entities that permit external network management.
- Detects and reports interference to support changing network channels.
- Manages network interference reporting and selection of a new channel for network operation if interference exists on the initial channel if the particular node is identified as the network manager for the overall PAN.

2.5.2.5 Binding Manager

The Binding Manager performs the following:

- Establishes resource size for the Binding Table. The size of this resource is determined via a programmed application or via a configuration attribute defined during installation.
- Processes bind requests for adding or deleting entries from the APS binding table.

- Supports Bind and Unbind commands from external applications such as those that may be hosted on a commissioning or network management tool to support assisted binding. Bind and Unbind commands shall be supported via the ZigBee Device Profile (see clause 2.4).
- For the ZigBee Coordinator, supports the End Device Bind that permits binding on the basis of button presses or other manual means.
- Permits source devices to register with a primary binding table cache their ability to hold their own binding table.
- Permits configuration tools to exchange one device for another in all the binding table entries which refer to it.
- Permits the primary binding table cache to backup and recover individual bind entries or the entire binding table or the table of source devices holding their own binding tables.

2.5.2.6 Node Manager

For ZigBee Coordinators and ZigBee Routers, the Node Management function performs the following:

- Permits remote management commands to perform network discovery.
- Provides remote management commands to retrieve the routing table.
- Provides remote management commands to retrieve the binding table.
- Provides a remote management command to have the device leave the network or to direct that another device leave the network.
- Provides a remote management command to retrieve the LQI for neighbors of the remote device.
- Provides a remote management command to Permit or disallow joining on particular routers or to generally allow or disallow joining via the Trust Center.

2.5.2.7 Group Manager

The Group Manager performs the following:

- Provides for inclusion of application objects within the local device into groups under application control.
- Provides for removal of application objects within the local device from group membership under application control.

2.5.3 Layer Interface Description

Unlike other device descriptors for applications residing above Endpoints 1-254, the ZigBee Device Objects (ZDO) interface to the APS via the APSME-SAP in addition to the APSDE-SAP. ZDO communicates over Endpoint 0 using the APSDE-SAP via Profiles like all other applications. The Profile used by ZDO is the ZigBee Device Profile (see clause 2.4). ZDO frames shall not be fragmented.

ZigBee Device Objects shall employ Endpoint 0 as the source and destination endpoint in any transmitted ZigBee Device Profile request frames, and shall expect Endpoint 0 as the source and destination endpoint in any received response frames.

2.5.4 System Usage

2.5.4.1 Object Overview

ZigBee Device Objects contain six Objects:

- Device and Service Discovery
- Network Manager
- Binding Manager
- Security Manager
- Node Manager
- Group Manager

Table 2.142 describes these ZigBee Device Objects.

Table 2.142 ZigBee Device Objects

Object		Description
Name	Status	
:Device_and_Service_Discovery	M	Handles device and service discovery.
:Network_Manager	M	Handles network activities such as network discovery, leaving/joining a network, resetting a network connection and creating a network.
:Binding_Manager	O	Handles end device binding, binding and unbinding activities.
:Security_Manager	O	Handles security services such as key loading, key establishment, key transport and authentication.
:Node_Manager	O	Handles management functions.
:Group Manager	O	Handles management of groups

2.5.4.2 Optional and Mandatory Objects and Attributes

Objects listed as Mandatory shall be present on all ZigBee devices. However, for certain ZigBee logical types, Objects listed as Optional for all ZigBee devices may be Mandatory in specific logical device types. For example, the NLME-NETWORK-FORMATION.request within the Network_Manager object is in a Mandatory object and is an Optional attribute, though the attribute is required for ZigBee Coordinator logical device types. The introduction section of each Device Object section will detail the support requirements for Objects and Attributes by logical device type.

2.5.4.3 Security Key Usage

ZigBee Device Objects may employ security for packets created by ZigBee Device Profile primitives. These application packets using APSDE on Endpoint 0 shall utilize the APSDE Security Service Provider interface like all other Application Objects.

2.5.4.4 Public and Private Methods

Methods that are accessible to any endpoint application on the device are called public methods. Private methods are only accessible to the Device Application on endpoint 0 and not to the end applications (which run on endpoints 1 through 254).

2.5.4.5 State Machine Functional Descriptions

2.5.4.5.1 ZigBee Coordinator

2.5.4.5.1.1 Initialization

The implementation shall set the startup-related IB attributes shown in Table 2.143 to values that reflect the desired startup behavior for the device. In particular, the *apsDesignatedCordinator* attribute of the IB shall be set to TRUE. If the device implements more than one option for ZigBee protocol version or stack profile, it shall choose a single value for each and set *nwkcProtocolVersion* and *nwkStackProfile* accordingly. Additionally, provision shall be made to provide configuration elements to describe the Node Descriptor, Power Descriptor, Simple Descriptor for each active endpoint and application plus the list of active endpoints. These configurations shall be embodied in :Config_Node_Descriptor, :Config_Power_Descriptor, and
:Config_Simple_Descriptors. If the :Config_Node_Descriptor configuration object indicates that this device is a Primary Discovery Cache device, the device shall be configured to process server commands for the ZigBee Device Profile associated with requests to the Primary Discovery Cache and shall operate according to the state machine description provided in section 2.5.2.1.

If supported, provision shall be made to supply configuration elements for the Complex Descriptor, User Descriptor, and the maximum number of bind entries. These elements shall be embodied in :Config_Complex_Descriptor, :Config_User_Descriptor, and :Config_Max_Bind.

To start as a ZigBee coordinator, the device application shall execute the startup procedure described in section 2.5.4.5.6.2 with startup attributes set as described above. This should have the effect of executing the procedure for network formation described in section 3.6.1.1. The device application shall set the *nwkSecurityLevel* and *nwkAllFresh* NIB attributes according to the values established by convention within the Stack Profile employed by the device. The device application shall check the return status via the NLME-NETWORK-FORMATION.confirm to verify successful creation of the PAN. The
:Config_Permit_Join_Duration shall be set according to the default attribute value supplied using the NLME-PERMIT-JOINING.request. Additionally, the nwkNetworkBroadcastDeliveryTime and nwkTransactionPersistenceTime Network Information Block attributes (see section 3.6.2) shall be set with :
Config_NWK_BroadcastDeliveryTime and :Config_NWK_TransactionPersistenceTime respectively (see section 2.5.5).

Provision shall be made to ensure APS primitive calls from the end applications over EP 1 through EP 254 return appropriate error status values prior to completion of the Initialization state by ZigBee Device Objects and transition to the normal operating state.

2.5.4.5.1.2 Normal Operating State

In this state, the ZigBee Coordinator shall process the list of direct joined addresses in
:Config_NWK_Join_Direct_Addrs by issuing an NLME-DIRECT-JOIN.request for each included address in the list. Processing of the direct joined addresses shall employ the :Config_Max_Assoc attribute in evaluating whether to successfully process a direct joined address within :Config_NWK_Join_Direct_Addrs.

The ZigBee coordinator shall allow other devices to join the network based on the configuration items :Config_Permit_Join_Duration and :Config_Max_Assoc. When a new device joins the network, the device application shall be informed via the NLME-JOIN.indication. Should the device be admitted to the PAN, the ZigBee coordinator shall indicate this via the NLME-JOIN.confirm with SUCCESS status.

The ZigBee coordinator shall respond to any device discovery or service discovery operations requested of its own device, and if it is designated as a Primary Discovery Cache device, shall also respond on behalf of registered devices that have stored discovery information. The device application shall also ensure that the number of binding entries does not exceed the :Config_Max_Bind attribute.

The ZigBee coordinator shall support the NLME-PERMIT-JOINING.request and NLME-PERMIT-JOINING.confirm to permit application control of network join processing.

The ZigBee coordinator shall support the NLME-LEAVE.request and NLME-LEAVE.indication employing the :Config_NWK_Leave_removeChildren attribute where appropriate to permit removal of associated devices under application control. Conditions that lead to removal of associated devices may include lack of security credentials, removal of the device via a privileged application or detection of exception.

The ZigBee coordinator shall maintain a list of currently associated devices and facilitate support of orphan scan and rejoin processing to enable previously associated devices to rejoin the network. The ZigBee coordinator may support the ability for devices to be directly included in the network via the NLME-DIRECT-JOIN.request and NLME-DIRECT-JOIN.confirm. This feature shall permit lists of ZigBee IEEE addresses to be provided to the ZigBee coordinator and for those addresses to be included as previously associated devices. It shall be possible for ZigBee devices with those addresses to directly join the network via orphaning or rejoin procedures rather than associating directly.

The ZigBee coordinator shall support the NLME-NWK-STATUS.indication and process those notifications per clause 3.2.2.30.

The ZigBee coordinator shall process End_Device_Bind_req from ZigBee Routers and ZigBee End Devices. Upon receipt of an End_Device_Bind_req, the ZigBee Coordinator shall use the :Config_EndDev_Bind_Timeout value in the attribute and await a second End_Device_Bind_req. Should the second indication arrive within the timeout period, the ZigBee coordinator shall match the Profile ID in the two indications (see section 2.3.3.2). If the Profile IDs in the two indications do not match, an appropriate error status is returned to each device via End_Device_Bind_rsp. Should the Profile IDs match, the ZigBee Coordinator shall match the AppInClusterLists and AppOutClusterLists in the two indications. Cluster IDs in the AppInClusterList of the first indication which match Cluster IDs in the AppOutClusterList of the second indication shall be saved in a list for inclusion in the resulting Bind_req notifying the devices of the match.

The ZigBee coordinator shall process Device_annce messages from other ZigBee devices. Upon receipt of a Device_annce where *nwkUseTreeRouting* is TRUE, the ZigBee coordinator shall check all internal tables holding 64-bit IEEE addresses for devices within the PAN for a match with the address supplied in the Device_annce message. If a match is detected, the ZigBee coordinator shall update its *nwkAddressMap* attribute of the NIB corresponding to the matched 64- bit IEEE address to reflect the updated 16-bit NWK address contained in the Device_annce. Upon receipt of a Device_annce where *nwkUseTreeRouting* is FALSE, the ZigBee Coordinator shall employ the address conflict resolution procedure detailed in section 3.6.9.

The ZigBee coordinator may generate APSME-AUTHENTICATE.requests under application control from other application objects, and may process and respond to APSME-AUTHENTICATE.indications from other devices. The ZigBee coordinator shall supply APSME-AUTHENTICATE.confirms to application objects whose requests have been processed.

2.5.4.5.1.3 Trust Center Operation

The network device pointed to by the address in *apsTrustCenterAddress* shall function as the Trust Center when security is enabled on the network.

The Trust Center operation is defined within section 4.6.2.

2.5.4.5.2 ZigBee Router

2.5.4.5.2.1 Initialization

The implementation shall set the startup-related IB attributes shown in Table 2.143 to values that reflect the desired startup behavior for the device. In particular, the *apsDesignatedCordinator* attribute of the IB shall be set to FALSE. If the :Config_Node_Descriptor configuration object indicates that this device is a Primary Discovery Cache device, the device shall be configured to process server commands for the ZigBee Device Profile associated with requests to the Primary Discovery Cache and shall operate according to the state machine description provided in section 2.5.2.1.

If supported, provision shall be made to supply configuration elements for the Complex Descriptor, User Descriptor, and the maximum number of bind entries.. These elements shall be embodied in :Config_Complex_Descriptor, :Config_User_Descriptor, and :Config_Max_Bind.

To start as a ZigBee router, the device application shall execute the startup procedure described in section 2.5.4.5.6.2 with startup attributes set as described above. This should have the effect of executing either the procedure for network rejoin described in section 3.6.1.4.2 or else the full procedure for network join through MAC association described in section 3.6.1.4.1. The NLME-NETWORK-DISCOVERY.request procedure shall be implemented :Config_NWK_Scan_Attempts, each separated in time by :Config_NWK_Time_btwn_Scans. The purpose of repeating the NLME-NETWORK-DISCOVERY.request is to provide a more accurate neighbor list and associated link quality indications to the NWK layer. Specification of the algorithm for selection of the PAN shall be left to the profile description and may include use of the Extended PAN ID, operational mode of the network, identity of the ZigBee Router or Coordinator identified on the PAN, depth of the ZigBee Router on the PAN from the ZigBee Coordinator for the PAN, capacity of the ZigBee Router or Coordinator, the routing cost, or the Protocol Version Number (these parameters are supplied by the NLME-NETWORK-DISCOVERY.confirm and the beacon payload).

The ZigBee router may join networks employing the current protocol version number or may join networks employing a previous protocol version number, under application control, if backward compatibility is supported in the device. A single ZigBee PAN shall consist of devices employing only a single protocol version number (networks with devices employing different protocol version numbers and frame formats within the same PAN are not permitted). An optional configuration attribute, :Config_NWK_alt_protocol_version, provides the protocol version numbers which the device may choose to employ other than the current protocol version number. Once the ZigBee router chooses a PAN and a specific protocol version number, it shall employ that protocol version number as its *nwkcProtocolVersion*. Additionally, the ZigBee router shall then adhere to all frame formats and processing rules supplied by the version of the ZigBee Specification employing that protocol version number.

The :Config_Permit_Join_Duration shall be set according to the default parameter value supplied using NLME-PERMIT-JOINING.request. The router shall support the NLME-START-ROUTER.request and NLME-START-ROUTER.confirm to begin operations as a router within the PAN it has joined. Additionally, the *nwkNetworkBroadcastDeliveryTime* and *nwkTransactionPersistenceTime* Network Information Block attributes (see section 3.6.2) shall be set with :Config_NWK_BroadcastDeliveryTime and :Config_NWK_TransactionPersistenceTime respectively (see section 2.5.5).

Provision shall be made to ensure APS primitive calls from the end applications over EP 1 through EP 254 return appropriate error status values prior to completion of the Initialization state by ZigBee Device Objects and transition to the normal operating state.

If the network has security enabled, the device shall wait for successful acquisition of the NWK key to start functioning as a router in the network. See section 4.6.2 for details on Trust Center operations.

The device application shall set the *nwkSecurityLevel* NIB attribute to the values used in the network and begin functioning as a router using NLME-START-ROUTER.req.

2.5.4.5.2.2 Normal Operating State

In this state, the ZigBee router shall allow other devices to join the network based on the configuration items :Config_Permit_Join_Duration and :Config_Max_Assoc. When a new device joins the network, the device application shall be informed via the NLME-JOIN.indication attribute. Should the device be admitted to the PAN, the ZigBee router shall indicate this via the NLME-JOIN.confirm with SUCCESS status. If security is enabled on the network, the device application shall inform the Trust Center via the APSME-UPDATE-DEVICE. request.

Orphan indications for which this device is not the parent are notified to the ZDO from the NWK layer by receipt of an NLME-JOIN.indication primitive with parameter IsParent set to value FALSE. The mechanism by which this is handled is described in section 2.5.4.5.4.

The ZigBee router shall respond to any device discovery or service discovery operations requested of its own device, and if it is designated as a Primary Discovery Cache device, shall also respond on behalf of registered devices that have stored discovery information. The device application shall also ensure that the number of binding entries does not exceed the :Config_Max_Bind attribute.

ZigBee router shall request the Trust Center to update its NWK key via the APSME-REQUEST-KEY.request. The ZigBee router shall support APSME-TRANSPORT-KEY.indication to receive keys from the Trust Center.

The ZigBee router shall support the NLME-PERMIT-JOINING.request and NLME-PERMIT-JOINING.confirm to permit application control of network join processing.

The ZigBee router shall support the NLME-NWK-STATUS.indication and process those notifications per section 3.2.2.30.

The ZigBee router shall support the NLME-LEAVE.request and NLME-LEAVE.confirm employing the :Config_NWK_Leave_removeChildren attribute where appropriate to permit removal of associated devices under application control. Conditions that lead to removal of associated devices may include lack of security credentials, removal of the device via a privileged application or detection of exception.

The ZigBee router shall process Device_annce messages from other ZigBee devices. Upon receipt of a Device_annce where *nwkUseTreeRouting* is TRUE, the ZigBee router shall check all internal tables holding 64-bit IEEE addresses for devices within the PAN for a match with the address supplied in the Device_annce message. If a match is detected, the ZigBee router shall update its *nwkAddressMap* of the NIB corresponding to the matched 64-bit IEEE address to reflect the updated 16-bit NWK address contained in the Device_annce. Upon receipt of a Device_annce where *nwkUseTreeRouting* is FALSE, the ZigBee Router shall employ the address conflict resolution procedure detailed in section 3.6.9.

The ZigBee router shall maintain a list of currently associated end devices and facilitate support of orphan scan and rejoin processing to enable previously associated end devices to rejoin the network.

The ZigBee router may decide it has lost contact with the network it was joined to. In this situation, the router should conduct an active scan to find the network. If the network is found more than once the router should attempt to rejoin where there is a more recent value of *nwkUpdateId* in the beacon payload.

2.5.4.5.3 Binding Table Cache Operation

Any router (including the coordinator) may be designated as either a primary binding table cache or a backup binding table cache.

It shall respond to the System_Server_Discovery_req primitive to enable other devices to discover it and use its facilities.

A primary binding table cache shall maintain a binding table and a table of devices registered to cache their binding tables.

A primary binding table cache shall respond to the Bind_Register_req and Replace_Device_req primitives described in clause 2.4.3.2.

If a backup binding table cache is available, a primary binding table cache shall use the additional bind management primitives to backup and restore its binding table and its table of source binding devices.

A backup binding table cache shall maintain a backup of the binding table and table of registered binding devices for one or more primary binding table caches. It shall support the bind management primitives for backup and restore of these tables.

2.5.4.5.4 Operations to Support Intra-PAN Portability

2.5.4.5.4.1 Overview

The operations described in this section are carried out by ZigBee Coordinator and ZigBee Router Devices for support of intra-PAN portability.

The main steps are summarized as follows:

- Detect the problem - The ZDO of the moved device is notified of acknowledgement failures via the NLME-NWK-STATUS.indication primitive, and identifies a problem.
- Carry out the NWK layer rejoin procedure - The ZDO of a moved ZED initiates this process using the NLME-JOIN.request primitive, either through a secured or un-secured rejoining procedure. The NWK rejoin procedures closely mirror the MAC association procedure. Note that ZigBee Routers shall also carry out this procedure periodically if they find that they are no longer in contact with the Trust Center.
- Security verification - Secured and unsecured protocol steps are described to ensure that the orphaned device should really be accepted.
- Inform the rest of the network - when a device changes parents the steps to complete address conflict detection in section 3.6.1.9 must be completed. These actions also serve to notify the old parent that the End Device has changed parents.
- Provide a means for parents that were temporarily unavailable and caused the end-device to rejoin are able to update their child tables once they are back online.

These steps are described in detail in the subsections below. The mechanism is illustrated for secured rejoin of a ZED in Figure 2.106, trust center rejoin of a ZED in Figure 2.107, and trust center rejoin of a ZR in **Error! Reference source not found.** respectively. Note that the NWK and SEC sections on secured and trust center rejoin (sections 3.2.2.11, 3.2.2.12, 3.2.2.13, 3.6.1.4 and 4.6.3) shall be the authoritative text for these procedures. The diagrams in this section are provided for illustrative purposes only.

Figure 2.106 Portability Message Sequence Chart: ZED Secured Rejoin

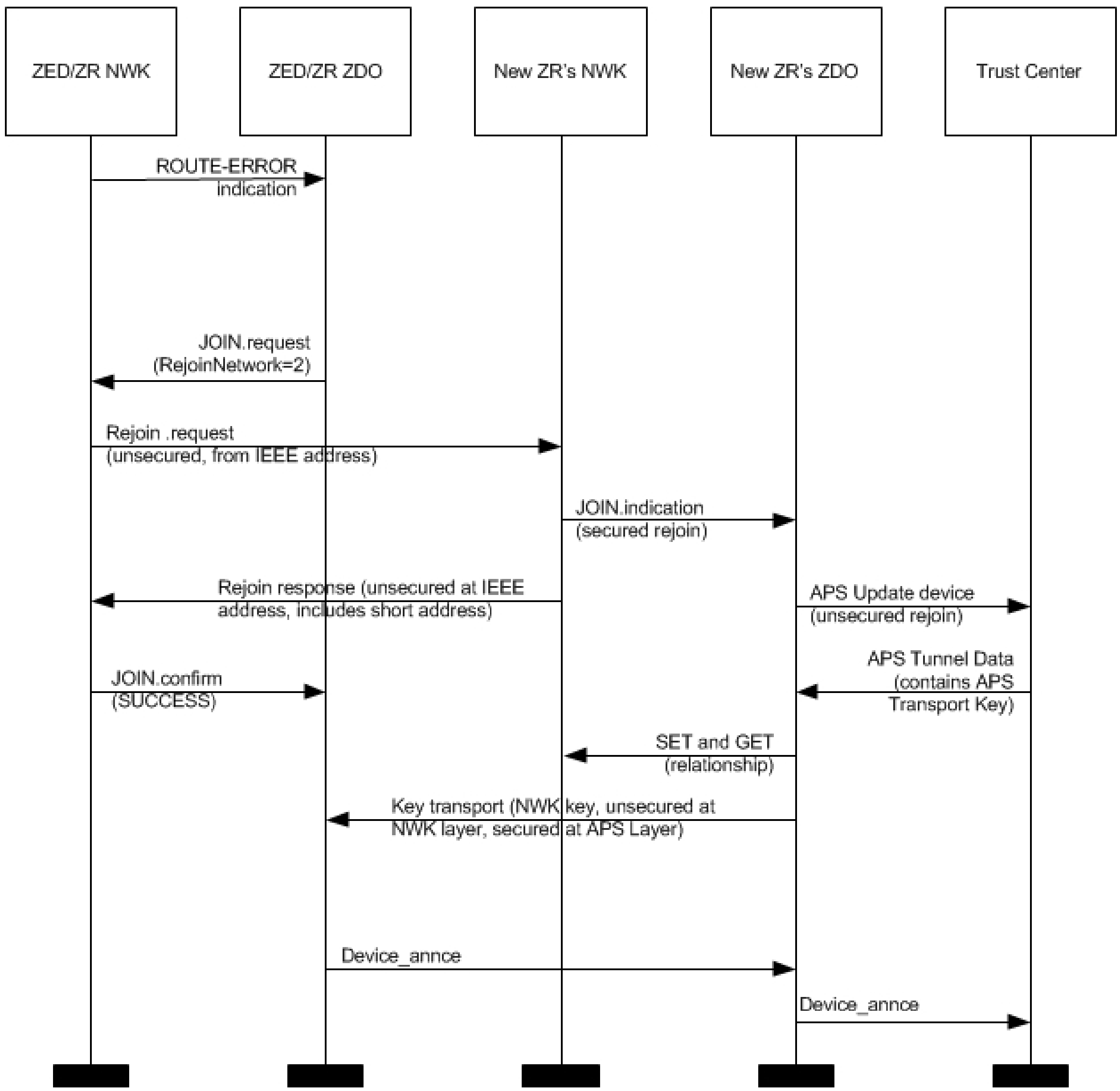
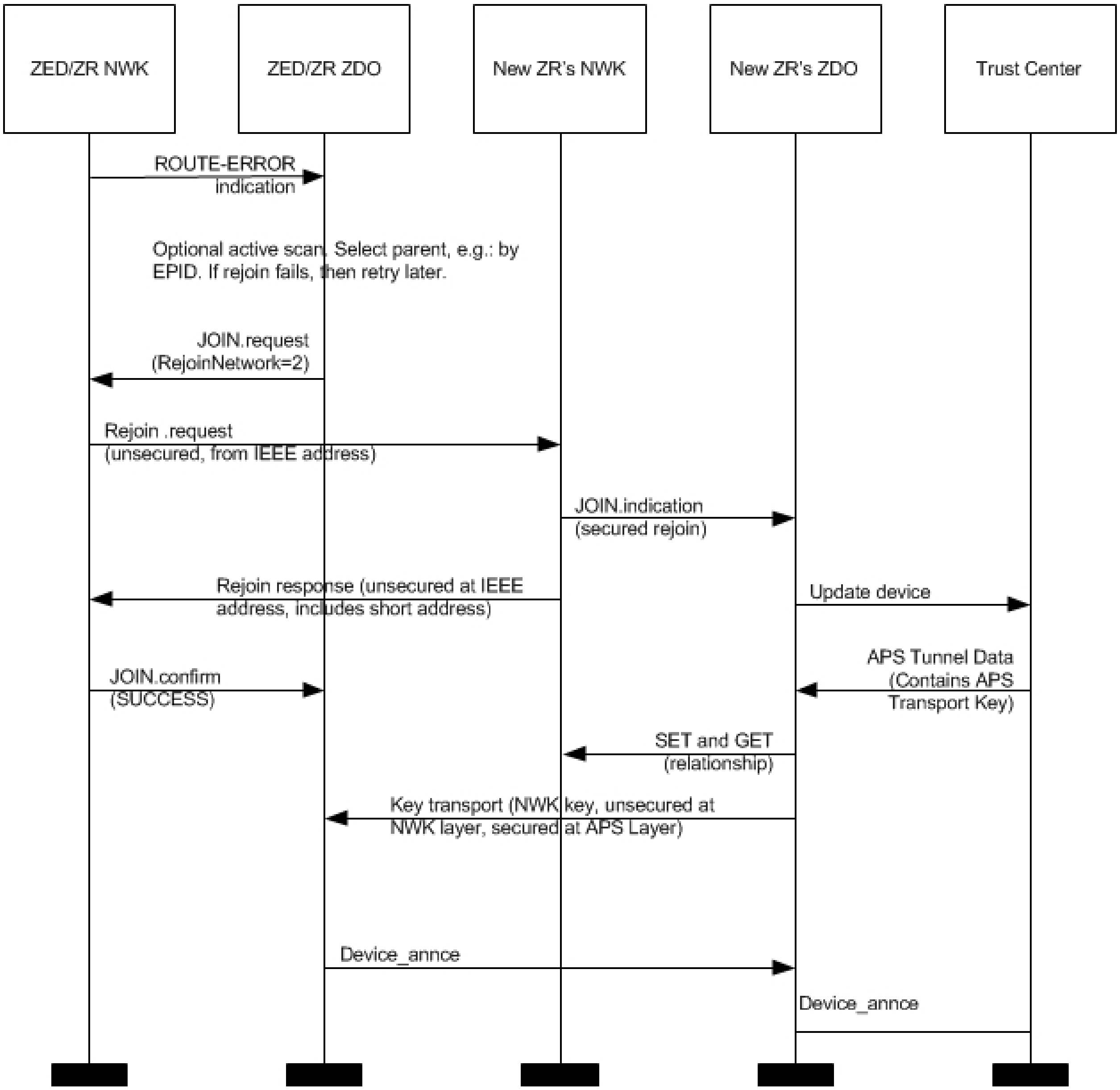


Figure 2.107 Portability Message Sequence Chart: ZR/ZED Trust Center Rejoin



2.5.4.5.4.2 Description of Operations for Security Verification

As for MAC association, a ZigBee Coordinator or ZigBee Router device is informed of a rejoined device when the NLME issues an NLME-JOIN.indication primitive. This shall be handed in the same way as for an association indication, except that for a secured rejoin the update device and key transport step.

Full network operation shall not be permitted until the verification steps described below have been carried out.

Measures shall be taken by a newly (re-)joined node and by its new parent to verify that it is really allowed to be on this network. Two cases are envisioned:

One or the other is not implemented according to this specification, and should not have joined. The measures described here allow both sides to revoke the join in this case.

One or the other device is a compromised/hacked device. In the case that security is enabled, the measures in section 4.6.3.6 are additionally applied so that an unauthorized join is revoked.

This verification is carried out using existing commands. Section 2.5.4.5.4.3 below describes the transmission of a Device_annce command to the new parent. The new parent shall check that this or some other message is correctly formed and contains the addressing fields corresponding to the orphaned device. If security is enabled, then this command shall be secured with the network key, and the new parent shall verify that all security processing is carried out correctly. If all these checks succeed then the orphaned device shall become joined to the network. Otherwise, it shall not become joined to the network at this time. As normal, messages sent from a device not joined to the network shall not be forwarded across the network, and commands shall not be carried out. Accordingly, the orphaned device shall only become joined to the network once it receives at least one correctly formed ZigBee message from the new parent. If security is enabled, this message must be secured with the network key and all security processing must be carried out correctly. If messages cannot be exchanged in protocol, then the orphaned device shall not become joined to the network at this time.

2.5.4.5.4.3 Description of Operations for Informing the Rest of the Network

If the ZigBee End Device rejoins a new parent using the orphaning of rejoin process it shall complete the address conflict process in section 3.6.1.9. Upon receiving the Device_annce, all devices shall check their internal tables holding 64-bit IEEE addresses for devices within the PAN for a match with the address supplied in the Device_annce message. If a match is detected, the device shall update the *nwkAddressMap* attribute of the NIB corresponding to the matched 64-bit IEEE address to reflect the updated 16-bit NWK address contained in the Device_annce. All devices shall use the NLME-SET and NLME-GET primitives to update the *nwkNeighborTable* in the NWK NIB. The previous parent of this ZED shall remove the ZED as one of its children by changing the Relationship field of the *nwkNeighborTable* to 0x04, “previous child.” Note that any unicast message sent to an address with this status shall result in an NLME-NWK-STATUS.indication primitive with status code of “Target Device Unavailable”, (see section 3.2.2.30). If
nwkUseTreeRouting is TRUE, address conflict detection is not provided and parent devices are not permitted, following intra-PAN portability, to remove devices or any other operation that reissue a short address for use by a child with a different IEEE address. Alternatively, if *nwkUseTreeRouting* is FALSE, address conflict detection is provided, however, devices will generally keep their existing NWK addresses during the intra-PAN portability procedure. Also, if the NWK address has changed during the intra-PAN portability procedure, the ZDO shall arrange that any IEEE address to short address mappings which have become known to applications running on this device be updated. This behavior is mandatory, but the mechanism by which it is achieved is outside the scope of this specification.

2.5.4.5.5 ZigBee End Device

2.5.4.5.5.1 Initialization

The implementation shall set the startup-related IB attributes shown in Table 2.143 to values that reflect the desired startup behavior for the device. In particular, the *apsDesignatedCordinator* attribute of the IB shall be set to FALSE.

If supported, provision shall be made to supply configuration elements for the Complex Descriptor, User Descriptor, and the maximum number of bind entries,. These elements shall be embodied in :Config_Complex_Descriptor, :Config_User_Descriptor, and :Config_Max_Bind. If the device application set the NLME-JOIN RxOnWhenIdle parameter to FALSE, the end device shall utilize the procedure described in section 2.5.2.1 to discover a Primary Discovery Cache device, register with it, and to successfully upload its device and service discovery information. To facilitate the process of uploading discovery information to the Primary Discovery Cache device, the local device may temporarily increase its polling rate with its parent. Prior to registering with any Primary Discovery Cache device, the end device shall utilize the Find Node Cache request to ensure it has not previously registered with any other Primary Discovery Cache device. If a server response indicates the end device has a previous registration, the end device shall update its discovery cache information on that Primary Discovery Cache device or shall remove its discovery cache information from that previous registration and create a new registration.

To start as a ZigBee end device, the device application shall execute the startup procedure described insection 2.5.4.5.6.2 with startup parameters set as described above. This should have the effect of executing either the procedure for network rejoin described in section 3.6.1.4.2 or else the full procedure for network join through MAC association described in section 3.6.1.4.1. The NLME-NETWORK-DISCOVERY.request procedure shall be implemented :Config_NWK_Scan_Attempts, each separated in time by :Config_NWK_Time_btwn_Scans. The purpose of repeating the NLME-NETWORK-DISCOVERY.request is to provide a more accurate neighbor list and associated link quality indications to the NWK layer. Specification of the algorithm for selection of the PAN shall be left to the profile description and may include use of the Extended PAN ID, operational mode of the network, identity of the ZigBee Router or Coordinator identified on the PAN, depth of the ZigBee Router on the PAN from the ZigBee Coordinator for the PAN, capacity of the ZigBee Router or Coordinator, the routing cost, or the Protocol Version Number (these parameters are supplied by the NLME-NETWORK-DISCOVERY.confirm and the beacon payload).

The ZigBee end device may join networks employing the current protocol version number or may join networks employing a previous protocol version number, under application control, if backward compatibility is supported in the device. A single ZigBee PAN shall consist of devices employing only a single protocol version number (networks with devices employing different protocol version numbers and frame formats within the same PAN are not permitted). An optional configuration attribute, :Config_NWK_alt_protocol_version, provides the protocol version numbers which the device may choose to employ other than the current protocol version number. Once the ZigBee end device chooses a PAN and a specific protocol version number, it shall employ that protocol version number as its *nwkcProtocolVersion*. Additionally, the ZigBee end device shall then adhere to all frame formats and processing rules supplied by the version of the ZigBee Specification employing that protocol version number.

If the device application sets the NLME-JOIN RxOnWhenIdle parameter to FALSE, the :Config_NWK_indirectPollRate shall be used to determine the polling rate for indirect message requests. The :Config_NWK_indirectPollRate shall be set according to the value established by the application profile(s) supported on the device. Once polling for indirect message requests is initiated, if communications failure with the parent is detected determined by failure of indirect message requests :Config_Parent_Link_Threshold_Retry consecutive attempts, the device application shall employ the network rejoin procedure.

Once the End Device has successfully joined a network, the device shall issue a Device_annce providing its 64-bit IEEE address and 16-bit NWK address.

Provision shall be made to ensure APS primitive calls from the end applications over EP 1 through EP 254 return appropriate error status values prior to completion of the Initialization state by ZigBee Device Objects and transition to the normal operating state.

If network has security enabled, the device shall wait successful acquisition of the NWK key to start functioning as an end device in the network. See section 4.6.2 for details on Trust Center operations.

2.5.4.5.5.2 Normal Operating State

If the device application set the NLME-JOIN RxOnWhenIdle parameter to FALSE, the :Config_NWK_ indirectPollRate shall be used to poll the parent for indirect transmissions while in the normal operating state. While a fragmented message is being received, the device may temporarily increase its polling rate, and shall ensure that it polls its parent at least once every macTransactionPersistenceTime seconds.

The ZigBee end device shall respond to any device discovery or service discovery operations requested of its own device using the attributes described in section 2.5.4.

ZigBee end device shall request the Trust Center to update its NWK key via the APSME-REQUEST-KEY.request. The ZigBee end device shall support APSME-TRANSPORT-KEY.indication to receive keys from the Trust Center.

The ZigBee End Device shall process Device_annce messages from other ZigBee devices. Upon receipt of a Device_annce where *nwkUseTreeRouting* is TRUE, the ZigBee End Device shall check all internal tables holding 64-bit IEEE addresses for devices within the PAN for a match with the address supplied in the Device_annce message. If a match is detected, the ZigBee End Device shall update the *nwkAddressMap* of the NIB corresponding to the matched 64-bit IEEE address to reflect the updated 16-bit NWK address contained in the Device_annce.

The ZigBee End Device shall process the NLME-NWK-STATUS.indication sent from the NWK layer. If the error code equals to 0x09 (Parent Link Failure), the ZED will update its failure counter maintained in ZDO. If the value of the failure counter is smaller than the :Config_Parent_Link_Retry_Threshold attribute, the ZED may decide to issue further commands to attempt to communicate with the parent node, depending on the application of the ZED. If the value of the failure counter exceeds the :Config_Parent_Link_ Retry_Threshold attribute, the ZED shall then prepare to start the rejoin process. Note that implementers may optionally use a more accurate time-windowed scheme to identify a link failure.

The rejoin process mirrors the MAC association process very closely, however, a device is permitted to rejoin a parent that is not accepting new associations. The ZDO may use the NLME-NETWORK-DISCOVERY.
request primitive to detect potential alternative parents, and in order to optimize recovery latency and reliability, shall select an appropriate new parent based on the following information from that device's beacon:

- PAN ID
- EPID (Extended PAN ID)
- Channel
- Signal strength
- Whether the potential parent indicates that it is currently able to communicate with its Trust Center
- Whether this device has recently failed to join this parent, or this network

Once a potential parent has been selected, the ZDO shall issue an NLME-JOIN.request primitive with RejoinNetwork set to 0x02.

The start time of the rejoin process is determined by the time the last NLME-JOIN.request primitive was sent and by the attribute :Config_Rejoin_Interval. Only if the interval between the current and the previous NLME-JOIN.request sent time is longer than the :Config_Rejoin_Interval shall a new NLME-JOIN.request primitive be sent. The application may want to gradually increase the :Config_Rejoin_Interval if a certain number of retries have been done (or a certain period of time has passed) but none of them were successful. The :Config_Rejoin_Interval should not exceed the :Config_Max_Rejoin_Interval. Every time an NLME-JOIN.confirm has been successfully received, the ZDO shall reset its failure counter to zero and the :Config_Rejoin_Interval attribute to its initial value. The choice of the default initial value and the algorithm of increasing the rejoin interval shall be determined by the application, and is out of the scope of this document.

If the ZigBee End Device rejoins a new parent using the rejoin process, it shall complete the address conflict process in section 3.6.1.9.

2.5.4.5.6 Support for Commissioning Applications

ZigBee devices in the field will need commissioning, and it will be up to developers to provide applications that perform such commissioning. There is a risk that applications from different vendors will work differently, thereby diminishing the ability of ZigBee devices from different vendors to operate seamlessly on the same network. As a partial solution to this problem, this section lists a common set of configuration attributes for ZigBee devices and outlines a common procedure for devices to use at start-up time. The other critical component of the solution is a common set of commissioning protocols and procedures, which are outside the scope of this document.

2.5.4.5.6.1 Configuration Attributes

The startup procedure outlined in section 2.5.4.5.6.2 is designed in such a way that, by using it consistently, devices can go through all the stages of commissioning up to being joined to the proper ZigBee network and able to send and receive application data traffic. Later-stage commissioning, including the commissioning of bindings and group membership is discussed briefly in section 2.5.4.5.6.3. The procedure makes use of the system attributes listed in Table 2.143.

Table 2.143 Startup Attributes

Name	Reference	Comment
nwkExtendedPANID	Table 3.43	This is the extended PANID of the network to which the device is joined. If it has a value of 0x0000000000000000, then the device is not connected to a network.
apsDesignatedCoordinator	Table 2.24	This boolean flag indicates whether the device should assume on startup that it must become a ZigBee coordinator.
apsChannelMask	Table 2.24	This is the mask containing allowable channels on which the device may attempt to form or join a network at startup time.
apsUseExtendedPANID	Table 2.24	The 64-bit identifier of the network to join or form.
apsUseInsecureJoin	Table 2.24	A Boolean flag that indicates if it is OK to use insecure join on startup.

2.5.4.5.6.2 Startup Procedure

The startup procedure uses the attributes listed in section 2.5.4.5.6.1 to perform a controlled startup of the ZigBee networking facilities of a device. The procedure should be run whenever the device restarts, but may also be run under application control at the discretion of the developer.

When a device starts up, it should check the value of *nwkExtendedPANID*. If *nwkExtendedPANID* has a non-zero value, then the device should assume it has all the network parameters required to operate on a network. Note that the device should assume the channel identifier present in its current network parameters but may need to scan over the ChannelMask if the *nwkExtendedPANID* is not found. In order for this to work effectively across power failures and processor resets, *nwkExtendedPANID* must be placed in non-volatile storage.

If the device finds it is not connected to a network, then it should check the value of *apsDesignatedCoordinator*. If this attribute has a value of TRUE, then the device should follow the procedures for starting a network outlined in section 3.6.1.4.1 and should use the value of *apsChannelMask* for the ScanChannels parameter of the NLME-NETWORK-FORMATION.request primitive, and set nwkExtendedPANID to the value given in *apsUseExtendedPANID* if *apsUseExtendedPANID* has a non-zero value.

If the device is not the designated coordinator and *apsUseExtendedPANID* has a non-zero value, the device should attempt to rejoin the network specified in *apsUseExtendedPANID*. To do this, it should use NLME-JOIN.request with the ExtendedPANID parameter equal to the value of *apsUseExtendedPANID*, the ScanChannels parameter of the primitive equal to the value of the *apsChannelMask* configuration attribute. The RejoinNetwork parameter of the NLME-JOIN.request primitive should have a value of 0x02 indicating rejoin.

If the network rejoin attempt fails, and the value of the *apsUseInsecureJoin* attribute of the AIB has a value of TRUE, then the device should follow the procedure outlined in section 3.6.1.4.1 for joining a network, using *apsChannelMask* any place that a ScanChannels mask is called for. If *apsUseExtendedPANID* has a non-zero value, then the device should join only the specified network and the procedure should fail if that network is found to be inaccessible. If *apsUseExtendedPANID* is equal to 0x0000000000000000, then the device should join the best available network.

2.5.4.5.6.3 Further Commissioning

Once a device is on a network and capable of communicating with other devices on the network in a secure manner, other commissioning becomes possible. Other items that should be subject to commissioning are shown in Table 2.144.

Table 2.144 Additional Commissioning Attributes

Name	Reference	Comment
apsBindingTable	Table 2.24	The binding table for this device. Binding provides a separation of concerns in the sense that applications may operate without having to manage recipient address information for the frames they emit. This information can be input at commissioning time without the main application on the device even being aware of it.
nwkGroupIDTable	Table 3.43	Commissioning applications should be able to manage group membership of a device and its endpoints by accessing this table.
nwkSecurityMaterialSet	Table 4.2	This set contains the network keying material, which should be accessible to commissioning applications.
apsDeviceKeyPairSet	Table 4.38	This is the set of link key pairs for devices that it wants to communicate using application layer encryption.
apsTrustCenterAddress	Table 4.38	The IEEE address of the Trust Center.

Name	Reference	Comment
nwkNetworkAddress	Table 3.44	Commissioning applications may set the network short address of devices as long as address conflicts that may arise as a result are subject to address conflict resolution as described in section 3.6.1.9.

2.5.4.6 Device and Service Discovery

The Device and Service Discovery function supports:

- Device Discovery
- Service Discovery

Device Management performs the above functions with the ZigBee Device Profile (see clause 2.4).

2.5.4.6.1 Optional and Mandatory Attributes Within Device and Service Discovery

All of the request attributes within the Device and Service Discovery Object are optional for all ZigBee logical device types. The responses listed in Table 2.145 as mandatory are mandatory for all ZigBee logical device types, and the responses listed as optional are optional for all ZigBee logical device types. See section The ZigBee Device Profile2.4 for a description of any of these attributes.

Table 2.145 Device and Service Discovery Attributes

Attribute	M/O	Type
NWK_addr_req	O	Public
NWK_addr_rsp	M	Public
IEEE_addr_req	O	Public
IEEE_addr_rsp	M	Public
Node_Desc_req	O	Public
Node_Desc_rsp	M	Public
Power_Desc_req	O	Public
Power_Desc_rsp	M	Public
Simple_Desc_req	O	Public
Simple_Desc_rsp	M	Public
Active_EP_req	O	Public

Attribute	**M/O**	**Type**
Active_EP_rsp	M	Public
Match_Desc_req	O	Public
Match_Desc_rsp	M	Public
Complex_Desc_req	O	Public
Complex_Desc_rsp	O	Public
User_Desc_req	O	Public
User_Desc_rsp	O	Public
Device_annce	M	Public
Parent_annce	M	Public
Parent_annce_rsp	M	Public
User_Desc_set	O	Public
User_Desc_conf	O	Public
Sys- tem_Server_Discovery_req	O	Public
Sys- tem_Server_Discovery_rsp	O	Public
Discovery_Cache_req	O	Public
Discovery_Cache_rsp	O	Public
Discovery_store_req	O	Public
Discovery_store_rsp	O	Public
Node_Desc_store_req	O	Public
Node_Desc_store_rsp	O	Public
Power_Desc_store_req	O	Public
Power_Desc_store_rsp	O	Public

Attribute	M/O	Type
Active_EP_store_req	O	Public
Active_EP_store_rsp	O	Public
Simple_Desc_store_req	O	Public
Simple_Desc_store_rsp	O	Public
Remove_node_cache_req	O	Public
Remove_node_cache_rsp	O	Public
Find_node_cache_req	O	Public
Find_node_cache_rsp	O	Public

2.5.4.7 Security Manager

The security manager determines whether security is enabled or disabled and, if enabled, shall perform the following:

- Establish Key
- Transport Key
- Authentication

2.5.4.7.1 Optional and Mandatory Attributes Within Security Manager

The Security Manager itself is an optional object for all ZigBee Device Types. If the Security Manager is present, all requests and responses are mandatory for all ZigBee device types. If the Security Manager is not present, none of the attributes in the Security Manager are present for any ZigBee logical device type. See section 2.4 for a description of any of the primitives listed in Table 2.146.

Table 2.146 Security Manager Attributes

Attribute	M/O	Type
APSME-TRANSPORT-KEY.request	O	Public
APSME-TRANSPORT-KEY.indication	O	Public
APSME-UPDATE-DEVICE.request	O	Public
APSME-UPDATE-DEVICE.indication	O	Public
APSME-REMOVE-DEVICE.request	O	Public
APSME-REMOVE-DEVICE.indication	O	Public

Attribute	M/O	Type
APSME-REQUEST-KEY.request	O	Public
APSME-REQUEST-KEY.indication	O	Public
APSME-SWITCH-KEY.request	O	Public
APSME-SWITCH-KEY.indication	O	Public

2.5.4.8 Binding Manager

The Binding Management function supports:

- End Device Binding
- Bind and Unbind

Binding Management performs the above functions with ZigBee Device Profile commands plus APSME-SAP primitives to commit/remove binding table entries once the indication arrives on the ZigBee coordinator, router, or end device supporting the binding table.

2.5.4.8.1 Optional and Mandatory Attributes Within Binding Manager

The Binding Manager is an optional object for all ZigBee Device Types.

If the Binding Manager is present, all requests are optional for all ZigBee logical device types. Responses shall be supported on devices which implement a binding table cache, and on devices which correspond to the source address for the binding table entries held on those devices.

If the Binding Manager is not present, all requests and all responses for all ZigBee logical device types shall not be supported. Table 2.147 summarizes Binding Manager attributes.

Table 2.147 Binding Manager Attributes

Attribute	M/O	Type
End_Device_Bind_req	O	Public
End_Device_Bind_rsp	O	Public
Bind_req	O	Public
Bind_rsp	O	Public
Unbind_req	O	Public
Unbind_rsp	O	Public
Bind_Register_req	O	Public
Bind_Register_rsp	O	Public
Replace_Device_req	O	Public
Replace_Device_rsp	O	Public
Store_Bkup_Bind_Entry_req	O	Public
Store_Bkup_Bind_Entry_rsp	O	Public
Remove_Bkup_Bind_req	O	Public
Remove_Bkup_Bind_rsp	O	Public
Backup_Bind_Table_req	O	Public
Backup_Bind_Table_rsp	O	Public
Recover_Bind_Table_req	O	Public
Recover_Bind_Table_rsp	O	Public
Backup_Source_Bind_req	O	Public
Backup_Source_Bind_rsp	O	Public
Recover_Source_Bind_req	O	Public
Recover_Source_Bind_rsp	O	Public

Attribute	M/O	Type
APSME-BIND.request	O	Private
APSME-BIND.confirm	O	Private
APSME-UNBIND.request	O	Private
APSME-UNBIND.confirm	O	Private

2.5.4.9 Network Manager

The Network Management function supports:

- Network Discovery
- Network Formation
- Permit/Disable Associations
- Association and Disassociation
- Route Discovery
- Network Reset
- Radio Receiver State Enable/Disable
- Get and Set of Network Management Information Block Data
- Detecting and reporting interference
- Receive network interference reports and change network channels if the particular node is identified as the network manager for the overall PAN

Network Management performs the above functions with NLME-SAP primitives (see Chapter 3).

2.5.4.9.1 Optional and Mandatory Attributes Within Network Manager

The Network Manager is a mandatory object for all ZigBee Device Types.

The Network Discovery, Get, and Set attributes (both requests and confirms) are mandatory for all ZigBee logical device types.

If the ZigBee logical device type is ZigBee Coordinator, the NWK Formation request and confirm, the NWK Leave request, NWK Leave indication, NWK Leave confirm, NWK Join indication, NWK Permit Joining request, NWK Permit Joining confirm, NWK Route Discovery request, and NWK Route Discovery confirm shall be supported. The NWK Direct Join request and NWK Direct Join confirm may be supported. The NWK Join request and the NWK Join confirm shall not be supported.

If the ZigBee logical device type is ZigBee Router, the NWK Formation request and confirm shall not be supported except if forming distributed networks. Additionally, the NWK Start Router request, NWK Start Router confirm, NWK Join request, NWK Join confirm, NWK Join indication, NWK Leave request, NWK Leave confirm, NWK Leave indication, NWK Permit Joining request, NWK Permit Joining confirm, NWK Route Discovery request, and NWK Route Discovery confirm shall be supported. The NWK Direct Join request and NWK Direct Join confirm may be supported.

If the ZigBee logical device type is ZigBee End Device, the NWK Formation request and confirm plus the NWK Start Router request and confirm shall not be supported. Additionally, the NWK Join indication and NWK Permit Joining request shall not be supported. The NWK Join request, NWK Join confirm, NWK Leave request, NWK Leave indication, NWK Leave confirm shall be supported.

For all ZigBee logical devices types, the NWK Sync request, indication and confirm plus NWK reset request and confirm plus NWK route discovery request and confirm shall be optional. Table 2.148 summarizes Network Manager Attributes. See Chapter 3 for a description of any of the primitives listed in Table 2.148.

For all ZigBee logical device types, reception of the NWK Network Status indication shall be supported, but no action is required in this version of the specification.

Table 2.148 Network Manager Attributes

Attribute	M/O	Type
NLME-GET.request	M	Private
NLME-GET.confirm	M	Private
NLME-SET.request	M	Private
NLME-SET.confirm	M	Private
NLME-NETWORK-DISCOVERY.request	M	Public
NLME-NETWORK-DISCOVERY.confirm	M	Public
NLME-NETWORK-FORMATION.request	O	Private
NLME-NETWORK-FORMATION.confirm	O	Private
NLME-START-ROUTER.request	O	Private
NLME-START-ROUTER.confirm	O	Private
NLME-JOIN.request	O	Private
NLME-JOIN.confirm	O	Private
NLME-JOIN.indication	O	Private
NLME-PERMIT-JOINING.request	O	Public
NLME-PERMIT-JOINING.confirm	O	Public
NLME-DIRECT-JOIN.request	O	Public
NLME-DIRECT-JOIN.confirm	O	Public
NLME_LEAVE.request	M	Public
NLME-LEAVE.confirm	M	Public

Attribute	**M/O**	**Type**
NLME_LEAVE.indication	M	Public
NLME-RESET.request	O	Private
NLME-RESET.confirm	O	Private
NLME-SYNC.request	O	Public
NLME-SYNC.indication	O	Public
NLME-SYNC.confirm	O	Public
NLME-NWK-STATUS.indication	M	Private
NLME-ROUTE-DISCOVERY.request	O	Public
NLME-ROUTE-DISCOVERY.confirm	O	Private
NLME-ED-SCAN.request	O	Private
NLME-ED-SCAN.confirm	O	Private
NLME-START-BACKOFF.request	O	Private

A single device in the network can become the Network Channel Manager. The operation of the network channel manager is described in Annex E. All other devices in the network are responsible for tracking message delivery failures and reporting interference in accordance with Annex E.

2.5.4.10 Node Manager

The Node Manager supports the ability to request and respond to management functions. These management functions only provide visibility to external devices regarding the operating state of the device receiving the request.

2.5.4.11 Group Manager

The Group Manager supports the ability to include application objects within groups or to remove application objects from groups. The group management functions operate only on application objects within the local device. Mechanisms to manage groups on other devices are beyond the scope of this document.

2.5.5 Configuration Attributes

This attribute is used to represent the minimum mandatory and/or optional attributes used as configuration attributes for a device.

Table 2.149 Configuration Attributes

Attribute	M/O	Type
:Config_Node_Descriptor	M	Public
:Config_Power_Descriptor	M	Public
:Config_Simple_Descriptors	M	Public
:Config_NWK_Scan_Attempts	M	Private
:Config_NWK_Time_btwn_Scans	M	Private
:Config_Complex_Descriptor	O	Public
:Config_User_Descriptor	O	Public
:Config_Max_Bind	O	Private
:Config_EndDev_Bind_Timeout	O	Private
:Config_Permit_Join_Duration	O	Public
:Config_NWK_Security_Level	O	Private
:Config_NWK_Secure_All_Frames	O	Private
:Config_NWK_Leave_removeChildren	O	Private
:Config_NWK_BroadcastDeliveryTime	O	Private
:Config_NWK_TransactionPersistenceTime	O	Private
:Config_NWK_indirectPollRate	O	Private
:Config_Max_Assoc	O	Private
:Config_NWK_Join_Direct_Addrs	O	Public
:Config_Parent_Link_Retry_Threshold	O	Public
:Config_Rejoin_Interval	O	Public
:Config_Max_Rejoin_Interval	O	Public

5909 ### 2.5.5.1 Configuration Attribute Definitions

5910 **Table 2.150 Configuration Attribute Definitions**

Attribute	Description	When Updated
:Config_Node_Descriptor	Contents of the Node Descriptor for this device (see section 2.3.2.3).	The :Config_Node_Descriptor is either created when the application is first loaded or initialized with a commissioning tool prior to when the device begins operations in the network. It is used for service discovery to describe node features to external inquiring devices.
:Config_Power_Descriptor	Contents of the Power Descriptor for this device (see section 2.3.2.4).	The :Config_Power_Descriptor is either created when the application is first loaded or initialized with a commissioning tool prior to when the device begins operations in the network. It is used for service discovery to describe node power features to external inquiring devices.
:Config_Simple_Descriptors	Contents of the Simple Descriptor(s) for each active endpoint for this device (see section 2.3.2.5).	The :Config_Simple_Descriptors are created when the application is first loaded and are treated as "read-only." The Simple Descriptor are used for service discovery to describe interfacing features to external inquiring devices.
:Config_NWK_Scan_Attempts	Integer value representing the number of scan attempts to make before the NWK layer decides which ZigBee coordinator or router to associate with (see section 2.5.4.5). This attribute has default value of 5 and valid values between 1 and 255.	The :Config_NWK_Scan_Attempts is employed within ZDO to call the NLME-NETWORK-DISCOVERY.request primitive the indicated number of times (for routers and end devices).

Attribute	Description	When Updated
:Config_NWK_Time_btwn_ Scans	Integer value representing the time duration (in OctetDurations) between each NWK discovery attempt described by :Config_NWK_Scan_Attempts (see section 2.5.4.5). This attribute has a default value of 0xc35 OctetDurations (100 milliseconds on 2.4GHz) and valid values between 1 and 0x1f3fe1 OctetDurations (65535 milliseconds on 2.4GHz).	The Config_NWK_Time_btwn_Scans is employed within ZDO to provide a time duration between the NLME-NETWORK-DISCOVERY.request attempts.
:Config_Complex_Descriptor	Contents of the (optional) Complex Descriptor for this device (see section 2.3.2.6).	The :Config_Complex_Descriptor is either created when the application is first loaded or initialized with a commissioning tool prior to when the device begins operations in the network. It is used for service discovery to describe extended device features for external inquiring devices.
:Config_User_Descriptor	Contents of the (optional) User Descriptor for this device (see section 2.3.2.7).	The :Config_User_Descriptor is either created when the application is first loaded or initialized with a commissioning tool prior to when the device begins operations in the network. It is used for service discovery to provide a descriptive character string for this device to external inquiring devices.
:Config_Max_Bind	A constant which describes the maximum number of binding entries permitted.	The :Config_Max_Bind is a maximum number of supported Binding Table entries for this device.
:Config_EndDev_Bind_Timeout	Timeout value in seconds employed in End Device Binding (see section 2.4.3.2).	The :Config_EndDev_Bind_Timeout is employed only on ZigBee Coordinators and used to determine whether end device bind requests have been received within the timeout window.

Attribute	Description	When Updated
:Config_Permit_Join_Duration	Permit Join Duration value set by the NLME-PERMIT-JOINING.request primitive (see Chapter 3).	The default value for :Config_Permit_Join_Duration is 0x00, however, this value can be established differently according to the needs of the profile.
:Config_NWK_Security_Level	Security level of the network (see Chapter 3).	This attribute is used only on the Trust Center and is used to set the level of security on the network.
:Config_NWK_Secure_All_Frames	If all network frames should be secured (see Chapter 3).	This attribute is used only on the Trust Center and is used to determine if network layer security shall be applied to all frames in the network.
:Config_NWK_Leave_removeChildren	Sets the policy as to whether child devices are to be removed if the device is asked to leave the network via NLME-LEAVE (see Chapter 3).	The policy for setting this attribute is found in the Stack Profile employed.
:Config_NWK_BroadcastDeliveryTime	See Chapter 3, Table 3-57.	The value for this configuration attribute is established in the Stack Profile.
:Config_NWK_TransactionPersistenceTime	See Table 3-44. This attribute is mandatory for the ZigBee coordinator and ZigBee routers and not used for ZigBee End Devices.	The value for this configuration attribute is established in the Stack Profile.
:Config_NWK_Alt_protocol_version	Sets the list of protocol version numbers, other than the current protocol version number, that the device may choose to employ in a PAN that it joins. This attribute is applicable only to ZigBee routers or end devices. The protocol version numbers in the list must refer to older versions of the ZigBee Specification.	:Config_NWK_ Alt_protocol_version permits ZigBee routers and ZigBee end devices to join networks discovered that employ an earlier version of the ZigBee Specification; Since this attribute is optional, devices may also be created omitting this attribute which require only the current version of the ZigBee Specification; This attribute would be omitted in cases where certain features are required that are contained only in the current specification or where code size is limited in the device.

Attribute	Description	When Updated
:Config_NWK_indirectPollRate	Sets the poll rate, in milliseconds, for the device to request indirect transmission messages from the parent.	The value for this configuration attribute is established by the application profile deployed on the device.
:Config_Max_Assoc	Sets the maximum allowed associations, either of routers, end devices, or both, to a parent router or coordinator.	The value for this configuration attribute is established by the stack profile in use on the device. Note that for some stack profiles, the maximum associations may have a dimension which provides for separate maximums for router associations and end device associations.
:Config_NWK_Join_Direct_Addrs	Consists of the following fields: DeviceAddress - 64-bit IEEE address for the device to be direct joined CapabilityInformation - Operating capabilities of the device to be direct joined Link Key- If security is enabled, link key for use in the key-pair descriptor for this new device (see Table 4.39) See section 3.2.2.14 for details.	:Config_NWK_Join_Direct_Addrs permits the ZigBee Coordinator or Router to be pre-configured with a list of addresses to be direct joined.
:Config_Parent_Link_Retry_Threshold	Contents of the link retry threshold for parent link (see section 2.5.4.5.5.2)	The Config_Parent_Link_Retry_Threshold is either created when the application is first loaded or initialized with a commissioning tool. It is used for the ZED to decide how many times it should retry to connect to the parent router before initiating the rejoin process.
:Config_Rejoin_Interval	Contents of the rejoin interval (see section 2.5.4.5.5.2).	The :Config_Rejoin_Interval is either created when the application is first loaded or initialized with a commissioning tool. It is used by the ZED to decide how often it should initiate the rejoin process.

Attribute	Description	When Updated
:Config_MAX_Rejoin_Interval	Contents of the maximal rejoin interval (see section 2.5.4.5.5.2).	The :Config_MAX_Rejoin_Interval is either created when the application is first loaded or initialized with a commissioning tool. It is used by the ZED to set the maximum value permitted for :Config_Rejoin_Interval during the rejoin procedure.

This page intentionally left blank.

CHAPTER 3 NETWORK SPECIFICATION

3.1 General Description

3.1.1 Network (NWK) Layer Overview

The network layer is required to provide functionality to ensure correct operation of the IEEE 802.15.4 MAC sub-layer and to provide a suitable service interface to the application layer. To interface with the application layer, the network layer conceptually includes two service entities that provide the necessary functionality. These service entities are the data service and the management service. The NWK layer data entity (NLDE) provides the data transmission service via its associated SAP, the NLDE-SAP, and the NWK layer management entity (NLME) provides the management service via its associated SAP, the NLME-SAP. The NLME utilizes the NLDE to achieve some of its management tasks and it also maintains a database of managed objects known as the network information base (NIB).

3.1.2 Network Layer Data Entity (NLDE)

The NLDE shall provide a data service to allow an application to transport application protocol data units (APDU) between two or more devices. The devices themselves must be located on the same network.

The NLDE will provide the following services:

- **Generation of the Network level PDU (NPDU):** The NLDE shall be capable of generating an NPDU from an application support sub-layer PDU through the addition of an appropriate protocol header.
- **Topology-specific routing:** The NLDE shall be able to transmit an NPDU to an appropriate device that is either the final destination of the communication or the next step toward the final destination in the communication chain.
- **Security:** The ability to ensure both the authenticity and confidentiality of a transmission.

3.1.2.1 Network Layer Management Entity (NLME)

The NLME shall provide a management service to allow an application to interact with the stack.

The NLME shall provide the following services:

- **Configuring a new device:** this is the ability to sufficiently configure the stack for operation as required. Configuration options include beginning an operation as a ZigBee coordinator or joining an existing network.
- **Starting a network:** this is the ability to establish a new network.
- **Joining, rejoining and leaving a network:** this is the ability to join, rejoin or leave a network as well as the ability of a ZigBee coordinator or ZigBee router to request that a device leave the network.
- **Addressing:** this is the ability of ZigBee coordinators and routers to assign addresses to devices joining the network.

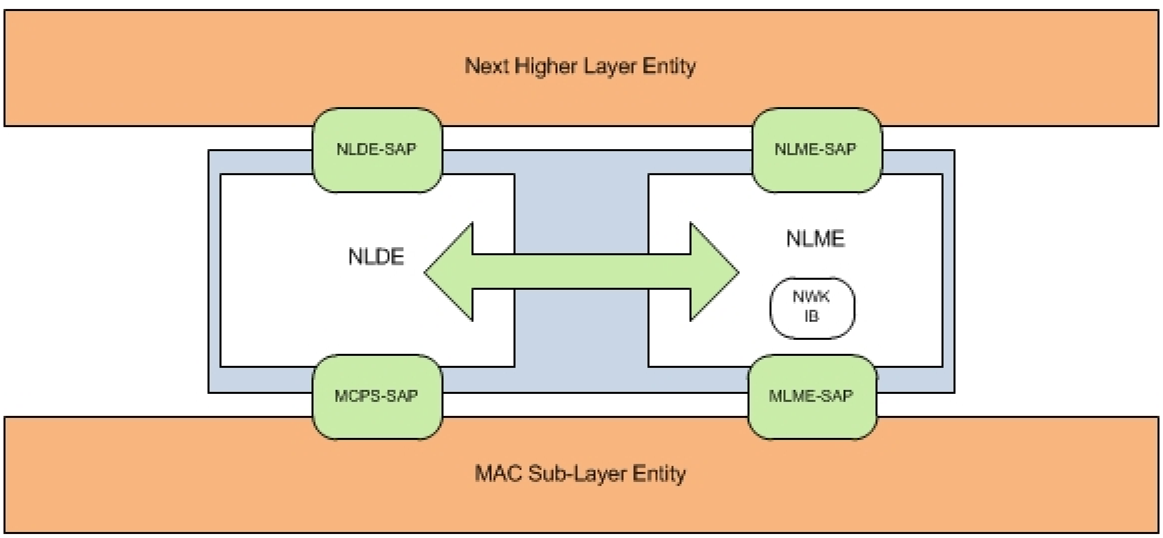
- **Neighbor discovery:** this is the ability to discover, record, and report information pertaining to the one-hop neighbors of a device.
- **Route discovery:** this is the ability to discover and record paths through the network, whereby messages may be efficiently routed.
- **Reception control:** this is the ability for a device to control when the receiver is activated and for how long, enabling MAC sub-layer synchronization or direct reception.
- **Routing:** this is the ability to use different routing mechanisms such as unicast, broadcast, multicast or many to one to efficiently exchange data in the network.

3.2 Service Specification

Figure 3.1 depicts the components and interfaces of the NWK layer.

The NWK layer provides two services, accessed through two service access points (SAPs). These are the NWK data service, accessed through the NWK layer data entity SAP (NLDE-SAP), and the NWK management service, accessed through the NWK layer management entity SAP (NLME-SAP). These two services provide the interface between the application and the MAC sub-layer, via the MCPS-SAP and MLME-SAP interfaces (See [B1]). In addition to these external interfaces, there is also an implicit interface between the NLME and the NLDE that allows the NLME to use the NWK data service.

Figure 3.1 The NWK Layer Reference Model



3.2.1 NWK Data Service

The NWK layer data entity SAP (NLDE-SAP) supports the transport of application protocol data units (APDUs) between peer application entities. Table 3.1 lists the primitives supported by the NLDE-SAP and the sections in which these primitives are discussed.

Table 3.1 NLDE-SAP Primitives

NLDE-SAP Primitive	Request	Confirm	Indication
NLDE-DATA	3.2.1.1	3.2.1.2	3.2.1.3

3.2.1.1 NLDE-DATA.request

This primitive requests the transfer of a data PDU (NSDU) from the local APS sub-layer entity to a single or multiple peer APS sub-layer entities.

3.2.1.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLDE-DATA.request           {
                            DstAddrMode,
                            DstAddr,
                            NsduLength,
                            Nsdu,
                            NsduHandle,
                            UseAlias,
                            AliasSrcAddr,
                            AliasSeqNumber,
                            Radius,
                            NonmemberRadius,
                            DiscoverRoute,
                            SecurityEnable
                            }
```

Table 3.2 specifies the parameters for the NLDE-DATA.request primitive. Support of the additional parameters UseAlias, AliasSrcAddr, AliasSeqNumb in the NLDE-DATA.request primitive is required if GP feature is to be supported by the implementation.

Table 3.2 NLDE-DATA.request Parameters

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x01 or 0x02	The type of destination address supplied by the DstAddr parameter. This may have one of the following two values: 0x01=16-bit multicast group address 0x02=16-bit network address of a device or a 16-bit broadcast address
DstAddr	16-bit Address	0x0000-0xffff	Destination address.

Name	Type	Valid Range	Description
NsduLength	Integer	0 to *aMaxPHYPacketSize* - *(nwkcMACFrameOverhead* + *nwkcMinHeaderOverhead)*	The number of octets comprising the NSDU to be transferred.
Nsdu	Set of Octets	-	The set of octets comprising the NSDU to be transferred.
NsduHandle	Integer	0x00 – 0xff	The handle associated with the NSDU to be transmitted by the NWK layer entity.
UseAlias	Boolean	TRUE or FALSE	The next higher layer MAY use the UseAlias parameter to request alias usage by NWK layer for the current frame. If the *UseAlias* parameter has a value of FALSE, meaning no alias usage, then the parameters *AliasSrcAddr* and *AliasSeqNumb* will be ignored. Otherwise, a value of TRUE denotesthat the values supplied in *AliasSrcAddr* and *AliasSeqNumb* are to be used.
AliasSrcAddr	16-bit address	Any valid device address except a broadcast address	The source address to be used for this NSDU. If the *UseAlias* parameter has a value of FALSE, the AliasSrcAddr parameter is ignored.
AliasSeqNumb	integer	0x00-0xff	The sequence number to be used for this NSDU. If the *UseAlias* parameter has a value of FALSE, the *AliasSeqNumb* parameter is ignored.
Radius	Unsigned Integer	0x00 – 0xff	The distance, in hops, that a frame will be allowed to travel through the network.
NonmemberRadius	Integer	0x00 – 0x07	The distance, in hops, that a multicast frame will be relayed by nodes not a member of the group. A value of 0x07 is treated as infinity.

Name	Type	Valid Range	Description
DiscoverRoute	Integer	0x00 – 0x01	The DiscoverRoute parameter may be used to control route discovery operations for the transit of this frame (see section 3.6.3.5): 0x00 = suppress route discovery 0x01 = enable route discovery
SecurityEnable	Boolean	TRUE or FALSE	The SecurityEnable parameter may be used to enable NWK layer security processing for the current frame. If the *nwkSecurityLevel* attribute of the NIB has a value of 0, meaning no security, then this parameter will be ignored. Otherwise, a value of TRUE denotes that the security processing specified by the security level will be applied, and a value of FALSE denotes that no security processing will be applied.

3.2.1.1.2 When Generated

This primitive is generated by a local APS sub-layer entity whenever a data PDU (NSDU) is to be transferred to a peer APS sub-layer entity.

3.2.1.1.3 Effect on Receipt

If this primitive is received on a device that is not currently associated, the NWK layer will issue an NLDE-DATA.confirm primitive with a status of INVALID_REQUEST.

On receipt of this primitive, the NLDE first constructs an NPDU in order to transmit the supplied NSDU. If, during processing, the NLDE issues the NLDE-DATA.confirm primitive prior to transmission of the NSDU, all further processing is aborted. In constructing the new NPDU, the destination address field of the NWK header will be set to the value provided in the DstAddr parameter. If the UseAlias parameter has a value of TRUE, the source address field of the NWK header of the frame will be set to the value provided in the AliasSrcAddr parameter. If the UseAlias parameter has a value of FALSE, then the source address field will have the value of the *macShortAddress* attribute in the MAC PIB. The discover route sub-field of the frame control field of the NWK header will be set to the value provided in the DiscoverRoute parameter. If the supplied Radius parameter does not have a value of zero, then the radius field of the NWK header will be set to the value of the Radius parameter. If the Radius parameter has a value of zero, then the radius field of the NWK header will be set to twice the value of the *nwkMaxDepth* attribute of the NIB. If the UseAlias parameter has a value of TRUE, the sequence number field of the NWK header of the frame will be set to the value provided in the AliasSeqNumb parameter. If the UseAlias parameter has a value of FALSE, then the NWK layer will generate a sequence number for the frame as described in section 3.6.2.1 and the sequence number field of the NWK header of the frame will be set to this sequence number value. The multicast flag field of the NWK header will be set according to the value of the DstAddrMode parameter. If the DstAddrMode parameter has a value of 0x01, the NWK header will contain a multicast control field whose fields will be set as follows:

- The multicast mode field will be set to 0x01 if this node is a member of the group specified in the DstAddr parameter.
- Otherwise, the multicast mode field will be set to 0x00.
- The non-member radius and the max non-member radius fields will be set to the value of the NonmemberRadius parameter.

Once the NPDU is constructed, the NSDU is routed using the procedure described in section Upon Receipt of a Unicast Frame if it is a unicast, section 3.6.5 if it is a broadcast, or section 3.6.6.2 if it is a multicast. When the routing procedure specifies that the NSDU is to be transmitted, this is accomplished by issuing the MCPS-DATA.request primitive with both the SrcAddrMode and DstAddrMode parameters set to 0x02, indicating the use of 16-bit network addresses. The SrcPANId and DstPANId parameters should be set to the current value of *macPANId* from the MAC PIB. The SrcAddr parameter will be set to the value of *macShortAddr* from the MAC PIB. The value of the DstAddr parameter is the next hop address determined by the routing procedure. If the message is a unicast, bit b0 of the TxOptions parameter should be set to 1 denoting that an acknowledgement is required. On receipt of the MCPS-DATA.confirm primitive on a unicast, the NLDE issues the NLDE-DATA.confirm primitive with a status equal to that received from the MAC sub-layer. Upon transmission of a MCPS-DATA.confirm primitive, in the case of a broadcast or multicast, the NLDE immediately issues the NLDE-DATA.confirm primitive with a status of success.

If the *nwkSecurityLevel* NIB attribute has a non-zero value and the SecurityEnable parameter has a value of TRUE, then NWK layer security processing will be applied to the frame before transmission as described in clause 4.3. Otherwise, no security processing will be performed at the NWK layer for this frame. The security processing SHALL always be performed using device's own extended 64-bit IEEE address and Outgoing Frame Counter attribute of the NIB, and those values SHALL be put into the auxiliary NWK header of the frame, even if UseAlias parameter has a value of TRUE. If security processing is performed and it fails for any reason, then the frame is discarded and the NLDE issues the NLDE-DATA.confirm primitive with a Status parameter value equal to that returned by the security suite.

3.2.1.2 NLDE-DATA.confirm

This primitive reports the results of a request to transfer a data PDU (NSDU) from a local APS sub-layer entity to a single peer APS sub-layer entity.

3.2.1.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

NLDE-DATA.confirm	{
	Status
	NsduHandle,
	TxTime
	}

Table 3.3 specifies the parameters for the NLDE-DATA.confirm primitive.

Table 3.3 NLDE-DATA.confirm Parameters

Name	Type	Valid Range	Description
Status	Status	INVALID_REQUEST, MAX_FRM_COUNTER, NO_KEY, BAD_CCM_OUTPUT, ROUTE_ERROR, BT_TABLE_FULL, FRAME_NOT_BUFFERED or any status values returned from security suite or the MCPS-DATA.confirm primitive (see [B1])	The status of the corresponding request.
NsduHandle	Integer	0x00 – 0xff	The handle associated with the NSDU being confirmed.
TxTime	Integer	Implementation specific	A time indication for the transmitted packet based on the local clock. The time should be based on the same point for each transmitted packet in a given implementation. This value is only provided if *nwkTimeStamp* is set to TRUE.

3.2.1.2.2 When Generated

This primitive is generated by the local NLDE in response to the reception of an NLDE-DATA.request primitive.

The Status field will reflect the status of the corresponding request, as described in section 3.2.1.1.3.

3.2.1.2.3 Effect on Receipt

On receipt of this primitive, the APS sub-layer of the initiating device is notified of the result of its request to transmit. If the transmission attempt was successful, the Status parameter will be set to SUCCESS. Otherwise, the Status parameter will indicate the error.

3.2.1.3 NLDE-DATA.indication

This primitive indicates the transfer of a data PDU (NSDU) from the NWK layer to the local APS sub-layer entity.

3.2.1.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLDE-DATA.indication      {
                          DstAddrMode,
                          DstAddr,
                          SrcAddr,
                          NsduLength,
                          Nsdu,
                          LinkQuality
                          RxTime
                          SecurityUse
                          }
```

Table 3.4 specifies the parameters for the NLDE-DATA.indication primitive.

Table 3.4 NLDE-DATA.indication Parameters

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x01 or 0x02	The type of destination address supplied by the DstAddr parameter. This may have one of the following two values: 0x01=16-bit multicast group address 0x02=16-bit network address of a device or a 16-bit broadcast address
DstAddr	16-bit Address	0x0000-0xffff	The destination address to which the NSDU was sent.
SrcAddr	16-bit Device address	Any valid device address except a broadcast address	The individual device address from which the NSDU originated.
NsduLength	Integer	0 to *aMaxPHYPacketSize* – (*nwkcMACFrameOverhead* + *nwkcMinHeaderOverhead*)	The number of octets comprising the NSDU being indicated.
Nsdu	Set of octets	–	The set of octets comprising the NSDU being indicated.

Name	Type	Valid Range	Description
LinkQuality	Integer	0x00 – 0xff	The link quality indication delivered by the MAC on receipt of this frame as a parameter of the MCPS-DATA.indication primitive (see [B1]).
RxTime	Integer	Implementation specific	A time indication for the received packet based on the local clock. The time should be based on the same point for each received packet on a given implementation. This value is only provided if *nwkTimeStamp* is set to TRUE.
SecurityUse	Boolean	TRUE or FALSE	An indication of whether the received data frame is using security. This value is set to TRUE if security was applied to the received frame or FALSE if the received frame was unsecured.

3.2.1.3.2 When Generated

This primitive is generated by the NLDE and issued to the APS sub-layer on receipt of an appropriately addressed data frame from the local MAC sub-layer entity.

3.2.1.3.3 Effect on Receipt

On receipt of this primitive, the APS sub-layer is notified of the arrival of data at the device.

3.2.2 NWK Management Service

The NWK layer management entity SAP (NLME-SAP) allows the transport of management commands between the next higher layer and the NLME. Table 3.5 lists the primitives supported by the NLME through the NLME-SAP interface and the sections containing details on each of these primitives.

Table 3.5 Summary of the Primitives Accessed Through the NLME-SAP

	Section Number in this Specification			
Name	**Request**	**Indication**	**Response**	**Confirm**
NLME-NETWORK-DISCOVERY	3.2.1.1			3.2.2.2
NLME-NETWORK-FORMATION	3.2.2.3			3.2.2.4
NLME-PERMIT-JOINING	3.2.2.5			3.2.2.6
NLME-START-ROUTER	3.2.2.7			3.2.2.8

	Section Number in this Specification			
Name	**Request**	**Indication**	**Response**	**Confirm**
NLME-ED-SCAN	3.2.2.9			3.2.2.10
NLME-JOIN	3.2.2.11	3.2.2.12		3.2.2.13
NLME-DIRECT-JOIN	3.2.2.14			3.2.2.15
NLME-LEAVE	3.2.2.16	3.2.2.17		3.2.2.18
NLME-RESET	3.2.2.19			3.2.2.20
NLME-SYNC	3.2.2.22			3.2.2.24
NLME-SYNC-LOSS		3.2.2.23		
NLME-GET	3.2.2.26			3.2.2.27
NLME-SET	3.2.2.28			3.2.2.29
NLME-NWK-STATUS		3.2.2.30		
NLME-ROUTE-DISCOVERY	3.2.2.32			3.2.2.32

3.2.2.1 NLME-NETWORK-DISCOVERY.request

This primitive allows the next higher layer to request that the NWK layer discover networks currently operating within the POS.

3.2.2.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-NETWORK-DISCOVERY.request  {
                                ScanChannels,
                                ScanDuration
                                }
```

Table 3.6 specifies the parameters for the NLME-NETWORK-DISCOVERY.request primitive.

Table 3.6 NLME-NETWORK-DISCOVERY.request Parameters

Name	Type	Valid Range	Description
ScanChannels	Bitmap	32-bit field	The five most significant bits (b27,..., b31) are reserved. The 27 least significant bits (b0, b1,... b26) indicate which channels are to be scanned (1 = scan, 0 = do not scan) for each of the 27 valid channels (see [B1]).
ScanDuration	Integer	0x00 – 0x0e	A value used to calculate the length of time to spend scanning each channel: The time spent scanning each channel is (*aBaseSuperframeDuration* * (2^n + 1)) symbols, where n is the value of the ScanDuration parameter. For more information on MAC sub-layer scanning (see [B1]).

3.2.2.1.2 When Generated

This primitive is generated by the next higher layer of a ZigBee device and issued to its NLME to request the discovery of networks operating within the device's personal operating space (POS).

3.2.2.1.3 Effect on Receipt

On receipt of this primitive, the NWK layer will attempt to discover networks operating within the device's POS by performing an active scan over the channels specified in the ScanChannels argument and using channel page zero, for the period specified in the ScanDuration parameter. The scan is performed by means of the MLME-SCAN.request primitive.

On receipt of the MLME-SCAN.confirm primitive, the NLME issues the NLME-NETWORK-DISCOVERY.confirm primitive containing the information about the discovered networks with a Status parameter value equal to that returned with the MLME-SCAN.confirm.

3.2.2.2 NLME-NETWORK-DISCOVERY.confirm

This primitive reports the results of a network discovery operation.

3.2.2.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-NETWORK-DISCOVERY.confirm          {
                                        Status
                                        NetworkCount,
                                        NetworkDescriptor,
                                        }
```

Table 3.7 describes the arguments of the NLME-NETWORK-DISCOVERY.confirm primitive.

Table 3.7 NLME-NETWORK-DISCOVERY.confirm Parameters

Name	Type	Valid Range	Description
Status	Status	Any status value returned with the MLME-SCAN.confirm primitive.	See [B1].
NetworkCount	Integer	0x00 – 0xff	Gives the number of networks discovered by the search.
NetworkDescriptor	List of network descriptors	The list contains the number of elements given by the NetworkCount parameter	A list of descriptors, one for each of the networks discovered. Table 3.8 gives a detailed account of the contents of each item.

Table 3.8 gives a detailed account of the contents of a network descriptor from the NetworkDescriptor parameter.

Table 3.8 Network Descriptor Information Fields

Name	Type	Valid Range	Description
ExtendedPANId	Integer	0x0000000000000001 - 0xfffffffffffffffe	The 64-bit PAN identifier of the network.
LogicalChannel	Integer	Selected from the available logical channels supported by the PHY (see [B1]).	The current logical channel occupied by the network.
StackProfile	Integer	0x00 – 0x0f	A ZigBee stack profile identifier indicating the stack profile in use in the discovered network.
ZigBeeVersion	Integer	0x00 – 0x0f	The version of the ZigBee protocol in use in the discovered network.
BeaconOrder	Integer	0x00 – 0x0f	This specifies how often the MAC sub-layer beacon is to be transmitted by a given device on the network. For a discussion of MAC sub-layer beacon order see [B1].
SuperframeOrder	Integer	0x00 – 0x0f	For beacon-oriented networks, that is, beacon order < 15, this specifies the length of the active period of the superframe. For a discussion of MAC sub-layer superframe order

Name	Type	Valid Range	Description
			see [B1].
PermitJoining	Boolean	TRUE or FALSE	A value of TRUE indicates that at least one ZigBee router on the network currently permits joining, *i.e.* its NWK has been issued an NLME-PERMIT-JOINING primitive and, the time limit if given, has not yet expired.
RouterCapacity	Boolean	TRUE or FALSE	This value is set to true if the device is capable of accepting join requests from router-capable devices and set to FALSE otherwise.
EndDeviceCapacity	Boolean	TRUE or FALSE	This value is set to true if the device is capable of accepting join requests from end devices and set to FALSE otherwise.

3.2.2.2.2 When Generated

This primitive is generated by the NLME and issued to its next higher layer on completion of the discovery task initiated by an NLME-NETWORK-DISCOVERY.request primitive.

3.2.2.2.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of a network search.

3.2.2.3 NLME-NETWORK-FORMATION.request

This primitive allows the next higher layer to request that the device start a new ZigBee network with itself as the coordinator and subsequently make changes to its superframe configuration.

3.2.2.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-NETWORK-FORMATION.request      {
                                    ScanChannels,
                                    ScanDuration,
                                    BeaconOrder,
                                    SuperframeOrder,
                                    BatteryLifeExtension
                                    DistributedNetwork
                                    DistributedNetworkAddress
                                    }
```

Table 3.9 specifies the parameters for the NLME-NETWORK-FORMATION.request primitive.

Table 3.9 NLME-NETWORK-FORMATION.request Parameters

Name	**Type**	**Valid Range**	**Description**
ScanChannels	Bitmap	32-bit field	The five most significant bits (b27,..., b31) are reserved. The 27 least significant bits (b0, b1,... b26) indicate which channels are to be scanned in preparation for starting a network (1=scan, 0=do not scan) for each of the 27 valid channels (see [B1]).
ScanDuration	Integer	0x00 – 0x0e	A value used to calculate the length of time to spend scanning each channel. The time spent scanning each channel is (*aBaseSuperframeDuration* * ($2n$ + 1)) symbols, where n is the value of the ScanDuration parameter (see [B1]).
BeaconOrder	Integer	0x00 – 0x0f	The beacon order of the network that the higher layers wish to form.
SuperframeOrder	Integer	0x00 – 0x0f	The superframe order of the network that the higher layers wish to form.
BatteryLifeExtension	Boolean	TRUE or FALSE	If this value is TRUE, the NLME will request that the ZigBee coordinator is started supporting battery life extension mode; If this value is FALSE, the NLME will request that the ZigBee coordinator is started without supporting battery life extension mode.

DistributedNetwork	Boolean	TRUE or FALSE	If this value is TRUE then it indicates that distributed network security will be used and therefore it is permissible for a ZigBee router to form the network. If FALSE, then this primitive may only be called by the ZigBee Coordinator.
DistributedNetworkAddress	Integer	0x0001 – 0xFFF7	The address the device will use when forming a distributed network.

3.2.2.3.2 When Generated

This primitive is generated by the next higher layer of a ZigBee coordinator-capable device and issued to its NLME to request the initialization of itself as the ZigBee coordinator of a new network.

3.2.2.3.3 Effect on Receipt

If DistributedNetwork is set to FALSE and the device is not capable of being a ZigBee coordinator, the NLME issues the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to INVALID_REQUEST. If DistributedNetwork is set to TRUE and the device is not capable of being a ZigBee router then NLME issues the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to INVALID_REQUEST. If DistributedNetwork is set to TRUE and the DistributedNetworkAddress is outside the valid range then processing shall fail with the Status parameter set to INVALID_REQUEST.

The NLME requests that the MAC sub-layer first perform an energy detection scan and then an active scan on the specified set of channels and using channel page zero. To do this, the NLME issues the MLME-SCAN.request primitive to the MAC sub-layer with the ScanType parameter set to indicate an energy detection scan and then issues the primitive again with the ScanType parameter set to indicate an active scan. After the completion of the active scan, on receipt of the MLME-SCAN.confirm primitive from the MAC sub-layer, the NLME selects a suitable channel. The NWK layer will pick a PAN identifier that does not conflict with that of any network known to be operating on the chosen channel. Once a suitable channel and PAN identifier are found, the NLME will choose an address as follows. If DistributedNetwork is set to FALSE, it shall use 0x0000 as the 16-bit short MAC address. If DistributedNetwork is set to TRUE then it shall use DistributedNetworkAddress as the 16-bit short MAC address. It shall inform the MAC sub-layer of the newly chosen address. To do this, the NLME issues the MLME-SET.request primitive to the MAC sub-layer to set the MAC PIB attribute *macShortAddress*. If the NIB attribute *nwkExtendedPANId* is equal to 0x0000000000000000, this attribute will be initialized with the value of the MAC constant *macExtendedAddress*. If no suitable channel or PAN identifier can be found, the NLME issues the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to STARTUP_FAILURE.

If only a single channel is provided in the higher layer request, the NLME does not need to request an energy scan prior to starting the network. An active scan is still conducted to ensure a PAN identifier is selected that does not conflict with that of any network known to be operating.

Next, the NLME issues the MLME-START.request primitive to the MAC sub-layer. The PANCoordinator parameter of the MLME-START.request primitive is set to TRUE. The BeaconOrder, SuperframeOrder, and BatteryLifeExtension parameters will have the same values as those given to the NLME-NETWORK-FORMATION.request. The CoordRealignment parameter in the MLME-START.request primitive is set to FALSE if the primitive is issued to start a new PAN. The CoordRealignment parameter is set to TRUE if the primitive is issued to change any of the PAN configuration attributes on an existing PAN. On receipt of the associated MLME-START.confirm primitive, the NLME issues the NLME-NETWORK-FORMATION.confirm primitive to the next higher layer with the status returned from the MLME-START.confirm primitive.

3.2.2.4 NLME-NETWORK-FORMATION.confirm

This primitive reports the results of the request to initialize a ZigBee coordinator in a network.

3.2.2.4.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-NETWORK-FORMATION.confirm          {
                                        Status
                                        }
```

Table 3.10 specifies the parameters for the NLME-NETWORK-FORMATION.confirm primitive.

Table 3.10 NLME-NETWORK-FORMATION.confirm Parameters

Name	Type	Valid Range	Description
Status	Status	INVALID_REQUEST, STARTUP_FAILURE or any status value returned from the MLME-START.confirm primitive	The result of the attempt to initialize a ZigBee coordinator.

3.2.2.4.2 When Generated

This primitive is generated by the NLME and issued to its next higher layer in response to an NLME-NETWORK-FORMATION.request primitive. This primitive returns a status value of INVALID_REQUEST, STARTUP_FAILURE or any status value returned from the MLME-START.confirm primitive. Conditions under which these values may be returned are described in section 3.2.2.3.3.

3.2.2.4.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to initialize the device as a ZigBee coordinator. If the NLME has been successful, the Status parameter will be set to SUCCESS. Otherwise, the Status parameter indicates the error.

3.2.2.5 NLME-PERMIT-JOINING.request

This primitive allows the next higher layer of a ZigBee coordinator or router to set its MAC sub-layer association permit flag for a fixed period when it may accept devices onto its network.

3.2.2.5.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-PERMIT-JOINING.request          {
```


PermitDuration
}

Table 3.11 specifies the parameters for the NLME-PERMIT-JOINING.request primitive.

Table 3.11 NLME-PERMIT-JOINING.request Parameters

Name	Type	Valid Range	Description
PermitDuration	Integer	0x00 – 0xff	The length of time in seconds during which the ZigBee coordinator or router will allow associations. The value 0x00 and 0xff indicate that permission is disabled or enabled, respectively, without a specified time limit.

3.2.2.5.2 When Generated

This primitive is generated by the next higher layer of a ZigBee coordinator or router and issued to its NLME whenever it would like to permit or prohibit the joining of the network by new devices.

3.2.2.5.3 Effect on Receipt

It is only permissible that the next higher layer of a ZigBee coordinator or router issue this primitive. On receipt of this primitive by the NWK layer of a ZigBee end device, the NLME-PERMIT-JOINING.confirm primitive returns a status of INVALID_REQUEST.

On receipt of this primitive with the PermitDuration parameter set to 0x00, the NLME sets the MAC PIB attribute, *macAssociationPermit*, to FALSE by issuing the MLME-SET.request primitive to the MAC sub-layer. Once the MLME-SET.confirm primitive is received, the NLME issues the NLME-PERMIT-JOINING.confirm primitive with a status equal to that received from the MAC sub-layer.

On receipt of this primitive with the PermitDuration parameter set to 0xff, the NLME sets the MAC PIB attribute, *macAssociationPermit,* to TRUE by issuing the MLME-SET.request primitive to the MAC sub-layer. Once the MLME-SET.confirm primitive is received, the NLME issues the NLME-PERMIT-JOINING.confirm primitive with a status equal to that received from the MAC sub-layer.

On receipt of this primitive with the PermitDuration parameter set to any value other than 0x00 or 0xff, the NLME sets the MAC PIB attribute, *macAssociationPermit*, to TRUE as described above. Following the receipt of the MLME-SET.confirm primitive, the NLME starts a timer to expire after PermitDuration seconds. Once the timer is set, the NLME issues the NLME-PERMIT-JOINING.confirm primitive with a status equal to that received by the MAC sub-layer. On expiration of the timer, the NLME sets *macAssociationPermit* to FALSE by issuing the MLME-SET.request primitive.

Every NLME-PERMIT-JOINING.request primitive issued by the next higher layer supersedes all previous requests.

3.2.2.6 NLME-PERMIT-JOINING.confirm

This primitive allows the next higher layer of a ZigBee coordinator or router to be notified of the results of its request to permit the acceptance of devices onto the network.

3.2.2.6.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

NLME-PERMIT-JOINING.confirm {
Status

}

Table 3.12 specifies the parameters for the NLME-PERMIT-JOINING.confirm primitive.

Table 3.12 NLME-PERMIT-JOINING.confirm Parameters

Name	Type	Valid Range	Description
Status	Status	INVALID_REQUEST or any status returned from the MLME-SET.confirm primitive (see [B1]).	The status of the corresponding request

3.2.2.6.2 When Generated

This primitive is generated by the initiating NLME of a ZigBee coordinator or router and issued to its next higher layer in response to an NLME-PERMIT-JOINING.request. The Status parameter either indicates the status received from the MAC sub-layer or an error code of INVALID_REQUEST. The reasons for these status values are described in section 3.2.2.5.

3.2.2.6.3 Effect on Receipt

On receipt of this primitive, the next higher layer of the initiating device is notified of the results of its request to permit devices to join the network.

3.2.2.7 NLME-START-ROUTER.request

This primitive allows the next higher layer of a ZigBee router to initiate the activities expected of a ZigBee router including the routing of data frames, route discovery, and the accepting of requests to join the network from other devices.

3.2.2.7.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-START-ROUTER.request  {
                           BeaconOrder,
                           SuperframeOrder,
                           BatteryLifeExtension
                           }
```

Table 3.13 specifies the parameters for NLME-START-ROUTER.request.

Table 3.13 NLME-START-ROUTER.request Parameters

Name	Type	Valid Range	Description
BeaconOrder	Integer	0x00 – 0x0f	The beacon order of the network that the higher layers wish to form.

SuperframeOrder	Integer	0x00 – 0x0f	The superframe order of the network that the higher layers wish to form.
BatteryLifeExtension	Boolean	TRUE or FALSE	If this value is TRUE, the NLME will request that the ZigBee router is started supporting battery life extension mode; If this value is FALSE, the NLME will request that the ZigBee router is started without supporting battery life extension mode.

3.2.2.7.2 When Generated

This primitive is generated by the next higher layer of a new device and issued to its NLME to request the initialization of itself as a ZigBee router.

3.2.2.7.3 Effect on Receipt

On receipt of this primitive by a device that is not already joined to a ZigBee network as a router, the NLME issues the NLME-START-ROUTER.confirm primitive with the Status parameter set to INVALID_REQUEST.

On receipt of this primitive by the NLME of a device that is joined to a ZigBee network as a router, the NLME issues the MLME-START.request primitive to the MAC sub-layer. The BeaconOrder, SuperframeOrder, and BatteryLifeExtension parameters of the MLME-START.request primitive will have the values given by the corresponding parameters of the NLME-START-ROUTER.request. The CoordRealignment parameter in the MLME-START.request primitive is set to FALSE if the primitive is issued to start as a router for the first time. The CoordRealignment parameter is set to TRUE thereafter if the primitive is issued to change any of the PAN configuration attributes.

On receipt of the associated MLME-START.confirm primitive, the NLME issues the NLME-START-ROUTER.confirm primitive to the next higher layer with the status returned from the MLME-START.confirm primitive. If, and only if, the status returned from the MLME-START.confirm primitive is SUCCESS, the device may then begin to engage in the activities expected of a ZigBee router including the routing of data frames, route discovery, and the accepting of requests to join the network from other devices. Otherwise, the device is expressly forbidden to engage in these activities.

3.2.2.8 NLME-START-ROUTER.confirm

This primitive reports the results of the request to initialize a ZigBee router.

3.2.2.8.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-START-ROUTER.confirm {
                          Status
                          }
```

Table 3.14 specifies the parameters for NLME-START-ROUTER.confirm.

Table 3.14 NLME-START-ROUTER.confirm Parameters

Name	Type	Valid Range	Description
Status	Status	INVALID_REQUEST or any status value returned from the MLME-START.confirm primitive.	The result of the attempt to initialize a ZigBee router.

3.2.2.8.2 When Generated

This primitive is generated by the NLME and issued to its next higher layer in response to an NLME-START-ROUTER.request primitive. This primitive returns a status value of INVALID_REQUEST or any status value returned from the MLME-START.confirm primitive. Conditions under which these values may be returned are described in section 3.2.2.7.3.

3.2.2.8.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to initialize a ZigBee router. If the NLME has been successful, the Status parameter will be set to SUCCESS. Otherwise, the Status parameter indicates the error.

3.2.2.9 NLME-ED-SCAN.request

This primitive allows the next higher layer to request an energy scan to evaluate channels in the local area.

3.2.2.9.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-ED-SCAN.request          {
                              ScanChannels,
                              ScanDuration,
                              }
```

Table 3.15 specifies the parameters for the service primitive.

Table 3.15 NLME-ED-SCAN.request Parameters

Name	Type	Valid Range	Description
ScanChannels	Bitmap	32-bit field	The five most significant bits (b27,..., b31) are reserved. The 27 least significant bits (b0, b1,... b26) indicate which channels are to be scanned (1=scan, 0=do not scan) for each of the 27 valid channels (see [B1]).

ScanDuration	Integer	0x00-0x0e	A value used to calculate the length of time to spend scanning each channel. The time spent scanning each channel is (*aBaseSuperframeDuration* * (2^n + 1)) symbols, where *n* is the value of the ScanDuration parameter [B1].

3.2.2.9.2 When Generated

The next higher layer of a device generates this primitive to request to conduct an energy scan of channels.

3.2.2.9.3 Effect on Receipt

On receipt of this primitive by a device that is currently joined to a network, the device will temporarily stop operating on the network to conduct an energy scan. To do this, the NLME issues the MLME-SCAN.request primitive to the MAC sub-layer with the ScanType parameter set to indicate an energy detection scan and the ScanChannels and ScanDuration from the NLME request and the ChannelPage set to zero.

3.2.2.10 NLME-ED-SCAN.confirm

This primitive provides the next higher layer results from an energy scan.

3.2.2.10.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-ED-SCAN.confirm          {
                              Status
                              EnergyDetectList
                              }
```

Table 3.16 specifies the parameters for the service primitive.

Table 3.16 NLME-ED-SCAN.confirm

Name	Type	Valid Range	Description
Status	Status	SUCCESS, or any valid code from the MAC	The status of the request.
EnergyDetectList	List of integers	0x00-0xff for each integer	The list of energy measurements in accordance with [B1], one for each channel.

3.2.2.10.2 When Generated

This primitive is generated by the NLME of a ZigBee device in response to an NLME-ED-SCAN.request. The status indicates the status received from the MAC sub-layer primitive MLME-SCAN.confirm. EnergyDetectList contains the ED scan result (List of integers, 0x00 - 0xff) for the channels scanned in accordance with IEEE 802.15.4-2003.

3.2.2.10.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of an ED scan.

3.2.2.11 NLME-JOIN.request

This primitive allows the next higher layer to request to join or rejoin a network, or to change the operating channel for the device while within an operating network.

3.2.2.11.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

NLME-JOIN.request	{
	ExtendedPANId,
	RejoinNetwork,
	ScanChannels,
	ScanDuration,
	CapabilityInformation,
	SecurityEnable
	}

Table 3.17 specifies the parameters for the NLME-JOIN.request primitive.

Table 3.17 NLME-JOIN.request

Name	**Type**	**Valid Range**	**Description**
ExtendedPANId	Integer	0x0000000000000001 – 0xfffffffffffffffe	The 64-bit PAN identifier of the network to join.
RejoinNetwork	Integer	0x00 – 0x03	This parameter controls the method of joining the network. The parameter is 0x00 if the device is requesting to join a network through association. The parameter is 0x01 if the device is joining directly or rejoining the network using the orphaning procedure. The parameter is 0x02 if the device is joining the network using the NWK rejoining procedure. The parameter is 0x03 if the device is to change the operational network channel to that identified in the ScanChannels parameter.
ScanChannels	Bitmap	32-bit field	The five most significant bits (b27,..., b31) are reserved. The 27 least significant bits (b0, b1,... b26) indicate which channels are to be scanned (1=scan, 0=do not scan) for each of the 27 valid channels (see [B1]).
ScanDuration	Integer	0x00-0x0e	A value used to calculate the length of time to spend scanning each channel. The time spent scanning each channel is (*aBaseSuperframeDuration* * $(2^n + 1)$) symbols, where n is the value of the ScanDuration parameter [B1].
CapabilityInformation	Bitmap	See Table 3.52.	The operating capabilities of the device being directly joined.
SecurityEnable	Boolean	TRUE or FALSE	If the value of RejoinNetwork is 0x02 and this is TRUE than the device will try to rejoin securely. Otherwise, this is set to FALSE.

3.2.2.11.2 When Generated

The next higher layer of a device generates this primitive to request to:

- Join a network using the MAC association procedure.
- Join or rejoin a network using the orphaning procedure.
- Join or rejoin a network using the NWK rejoin procedure.
- Switch the operating channel for a device that is joined to a network.

3.2.2.11.3 Effect on Receipt

On receipt of this primitive by a device that is currently joined to a network and with the RejoinNetwork parameter equal to 0x00, the NLME issues an NLME-JOIN.confirm primitive with the Status parameter set to INVALID_REQUEST.

On receipt of this primitive by a device that is not currently joined to a network and with the RejoinNetwork parameter equal to 0x00, the device attempts to join the network specified by the 64-bit ExtendedPANId parameter as described in section 3.6.1.4.1.1.

Whether joining or rejoining, the device shall set the nwkParentInformation in the NIB to 0.

If a device receives this primitive and the RejoinNetwork parameter is equal to 0x01, then it attempts to join or rejoin the network using orphaning as described in section 3.6.1.4.3.2.

On receipt of this primitive with the RejoinNetwork parameter is equal to 0x02, the device attempts to rejoin the network with 64-bit extended PAN ID given by the ExtendedPANId parameter. The procedure for rejoining is given in section 3.6.1.4.2.

Once the device has successfully joined a network, it will set the value of the *nwkExtendedPANId* NIB attribute to the extended PAN identifier of the network to which it joined.

If a device receives this primitive and the RejoinNetwork parameter is equal to 0x03, and the device supports setting the value of phyCurrentChannel then the device attempts to switch the operating channel to that provided in the ScanChannels parameter. If more than one channel is provided in the ScanChannels parameter, the NLME issues an NLME-JOIN.confirm primitive with the Status parameter set to INVALID_REQUEST. If the channel change is performed successfully, then the device issues a NLME-JOIN.confirm with the Status parameter set to SUCCESS. If the device does not support the setting of phyCurrentChannel directly, then the NLME issues a NLME-JOIN.confirm primitive with the Status parameter set to UNSUPPORTED_ATTRIBUTE.

If the MAC layer returned an error status during the channel change then this status shall be reported in the status field of the NLME-JOIN.confirm primitive.

3.2.2.12 NLME-JOIN.indication

This primitive allows the next higher layer of a ZigBee coordinator or ZigBee router to be notified when a new device has successfully joined its network by association or rejoined using the NWK rejoin procedure as described in section 3.6.1.4.3.

3.2.2.12.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-JOIN.indication        {
                            NetworkAddress,
                            ExtendedAddress,
                            CapabilityInformation,
                            RejoinNetwork
                            SecureRejoin
```


}

Table 3.18 specifies the parameters for the NLME-JOIN.indication primitive.

Table 3.18 NLME-JOIN.indication Parameters

Name	Type	Valid Range	Description
NetworkAddress	Network address	0x0001 – 0xfff7	The network address of an entity that has been added to the network.
ExtendedAddress	64-bit IEEE address	Any 64-bit, IEEE address	The 64-bit IEEE address of an entity that has been added to the network.
CapabilityInformation	Bitmap	See [B1]	Specifies the operational capabilities of the joining device.
RejoinNetwork	Integer	0x00 - 0x02	The RejoinNetwork parameter indicating the method used to join the network. The parameter is 0x00 if the device joined through association. The parameter is 0x01 if the device joined directly or rejoined using orphaning. The parameter is 0x02 if the device used NWK rejoin.
SecureRejoin	Boolean	TRUE or FALSE	This parameter will be TRUE if the rejoin was performed in a secure manner. Otherwise, this parameter will be FALSE.

3.2.2.12.2 When Generated

This primitive is generated by the NLME of a ZigBee coordinator or router and issued to its next higher layer on successfully adding a new device to the network using the MAC association procedure as shown in Figure 3.37, or on allowing a device to rejoin the network using the NWK rejoining procedure as shown in Figure 3.42.

3.2.2.12.3 Effect on Receipt

On receipt of this primitive, the next higher layer of a ZigBee coordinator or ZigBee router is notified that a new device has joined its network.

3.2.2.13 NLME-JOIN.confirm

This primitive allows the next higher layer to be notified of the results of its request to join a network.

3.2.2.13.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-JOIN.confirm        {
                          Status,
                          NetworkAddress,
                          ExtendedPANID,
                          ActiveChannel
                          }
```

Table 3.19 specifies the parameters for the NLME-JOIN.confirm primitive.

Table 3.19 NLME-JOIN.confirm

Name	**Type**	**Valid Range**	**Description**
Status	Status	INVALID_REQUEST, NOT_PERMITTED, NO_NETWORKS or any status value returned from the MLME-ASSOCIATE.confirm primitive, the MLME-SCAN.confirm primitive or the PLME-SET.confirm	The status of the corresponding request.
NetworkAddress	Integer	0x0001 – 0xfff7 and 0xffff	The 16-bit network address that was allocated to this device. This parameter will be equal to 0xffff if the join attempt was unsuccessful.
ExtendedPANID	Integer	0x0000000000000001 – 0xfffffffffffffffe	The 64-bit extended PAN identifier for the network of which the device is now a member.
ActiveChannel	Integer	The number of any channel supported by the PHY (see [B1]).	The value of *phyCurrentChannel* attribute of the PHY PIB, which is equal to the current channel of the network that has been joined.

3.2.2.13.2 When Generated

This primitive is generated by the initiating NLME and issued to its next higher layer in response to an NLME-JOIN.request primitive. If the request was successful, the Status parameter will have a value of SUCCESS. Otherwise, the Status parameter indicates an error code of INVALID_REQUEST, NOT_PERMITTED, NO_NETWORKS or any status value returned from either the MLME-ASSOCIATE.confirm primitive, the MLME-SCAN.confirm primitive or the PLME-SET.confirm primitive. The reasons for these status values are fully described in section 3.2.2.11.3.

3.2.2.13.3 Effect on Receipt

On receipt of this primitive, the next higher layer of the initiating device is notified of the results of its request to join a network using the MAC sub-layer association procedure, to join directly using the MAC sub-layer orphaning procedure, or to re-join a network once it has been orphaned.

3.2.2.14 NLME-DIRECT-JOIN.request

This optional primitive allows the next higher layer of a ZigBee coordinator or router to request to directly join another device to its network.

3.2.2.14.1 Semantics of the Service Primitive

The semantics of this optional primitive are as follows:

```
NLME-DIRECT-JOIN.request        {
                                DeviceAddress,
                                CapabilityInformation
                                }
```

Table 3.20 specifies the parameters for the NLME-DIRECT-JOIN.request primitive.

Table 3.20 NLME-DIRECT-JOIN.request Parameters

Name	Type	Valid Range	Description
DeviceAddress	64-bit IEEE address	Any 64-bit IEEE address	The IEEE address of the device to be directly joined.
CapabilityInformation	Bitmap	See Table 3.52.	The operating capabilities of the device being directly joined.

3.2.2.14.2 When Generated

The next higher layer of a ZigBee coordinator or router generates this primitive to add a new device directly to its network. This process is completed without any over the air transmissions.

3.2.2.14.3 Effect on Receipt

On receipt of this primitive, the NLME will attempt to add the device specified by the DeviceAddress parameter to its neighbor table. The CapabilityInformation parameter will contain a description of the device being joined. The alternate PAN coordinator bit is set to 0 in devices implementing this specification. The device type bit is set to 1 if the device is a ZigBee router, or to 0 if it is an end device. The power source bit is set to 1 if the device is receiving power from the alternating current mains or to 0 otherwise. The receiver on when idle bit is set to 1 if the device does not disable its receiver during idle periods, or to 0 otherwise. The security capability bit is set to 1 if the device is capable of secure operation, or to 0 otherwise.

If the NLME successfully adds the device to its neighbor table, the NLME issues the NLME-DIRECT-JOIN.confirm primitive with a status of SUCCESS. If the NLME finds that the requested device is already present in its neighbor tables, the NLME issues the NLME-DIRECT-JOIN.confirm primitive with a status of ALREADY_PRESENT. If no capacity is available to add a new device to the device list, the NLME issues the NLME-DIRECT-JOIN.confirm primitive with a status of NEIGHBOR_TABLE_FULL.

3.2.2.15 NLME-DIRECT-JOIN.confirm

This primitive allows the next higher layer of a ZigBee coordinator or router to be notified of the results of its request to directly join another device to its network.

3.2.2.15.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-DIRECT-JOIN.confirm    {
                            Status,
                            DeviceAddress
                            }
```

Table 3.21 specifies the parameters for the NLME-DIRECT-JOIN.confirm primitive.

Table 3.21 NLME-DIRECT-JOIN.confirm Parameters

Name	Type	Valid Range	Description
Status	Status	SUCCESS, ALREADY_PRESENT, NEIGHBOR_TABLE_FULL	The status of the corresponding request.
DeviceAddress	64-bit IEEE address	Any 64-bit, IEEE address	The 64-bit IEEE address in the request to which this is a confirmation.

3.2.2.15.2 When Generated

This primitive is generated by the initiating NLME and issued to its next higher layer in response to an NLME-DIRECT-JOIN.request primitive. If the request was successful, the Status parameter indicates a successful join attempt. Otherwise, the Status parameter indicates an error code of ALREADY_PRESENT or NEIGHBOR_TABLE_FULL. The reasons for these status values are fully described in section 3.2.2.14.3.

3.2.2.15.3 Effect on Receipt

On receipt of this primitive, the next higher layer of the initiating device is notified of the results of its request to directly join another device to a network.

3.2.2.16 NLME-LEAVE.request

This primitive allows the next higher layer to request that it or another device leaves the network.

3.2.2.16.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-LEAVE.request          {
                            DeviceAddress,
                            RemoveChildren,
                            Rejoin
                            }
```


Table 3.22 specifies the parameters for the NLME-LEAVE.request primitive.

Table 3.22 NLME-LEAVE.request Parameters

Name	Type	Valid Range	Description
DeviceAddress	Device address	Any 64-bit, IEEE address	The 64-bit IEEE address of the entity to be removed from the network or NULL if the device removes itself from the network.
RemoveChildren	Boolean	TRUE or FALSE	This parameter has a value of TRUE if the device being asked to leave the network is also being asked to remove its child devices, if any. Otherwise, it has a value of FALSE.
Rejoin	Boolean	TRUE or FALSE	This parameter has a value of TRUE if the device being asked to leave from the current parent is requested to rejoin the network. Otherwise, the parameter has a value of FALSE. Note that the Rejoin parameter is set by the application so cannot be relied upon by the networking layer to indicate whether a Join or Rejoin request will be accepted in the future.

3.2.2.16.2 When Generated

The next higher layer of a device generates this primitive to request to leave the network. The next higher layer of a ZigBee coordinator or router may also generate this primitive to remove a device from the network.

3.2.2.16.3 Effect on Receipt

On receipt of this primitive by the NLME of a device that is not currently joined to a network, the NLME issues the NLME-LEAVE.confirm primitive with a status of INVALID_REQUEST. On receipt of this primitive by the NLME of a device that is currently joined to a network, with the DeviceAddress parameter equal to the local device’s IEEE address or NULL, the NLME will remove itself from the network using the procedure described in section 3.6.1.10.1, and the value of the Rejoin parameter shall be copied into the Network Leave command frame that is generated. If the Rejoin parameter is set to TRUE, no further action is taken. If the Rejoin parameter is set to FALSE the NLME will then clear its routing table and route discovery table and issue an MLME-RESET.request primitive to the MAC sub-layer. The NLME will also set the relationship field of the neighbor table entry corresponding to its former parent to 0x03, indicating no relationship. If the NLME-LEAVE.request primitive is received with the DeviceAddress parameter equal to NULL and the RemoveChildren parameter equal to TRUE, then the NLME will attempt to remove the device's children, as described in section 3.6.1.10.3.

On receipt of this primitive by a ZigBee coordinator or ZigBee router and with the DeviceAddress parameter not equal to NULL and not equal to the local device's IEEE address, the NLME determines whether the specified device is in the Neighbor Table and the device type is 0x02 (Zigbee End device). If the requested device does not exist or the device type is not 0x02, the NLME issues the NLME-LEAVE.confirm primitive with a status of UNKNOWN_DEVICE. If the requested device exists, the NLME will attempt to remove it from the network using the procedure described in section 3.6.1.10.3. If the RemoveChildren parameter is equal to TRUE then the device will be requested to remove its children as well. Following the removal, the NLME will issue the NLME-LEAVE.confirm primitive with the DeviceAddress equal to the 64-bit IEEE address of the removed device and the Status parameter equal to the status delivered by the MCPS-DATA.confirm primitive. If the relationship field of the neighbor table entry corresponding to the leaving device has a value of 0x01 then it will be changed to 0x04 indicating previous child. If it has a value of 0x05, indicating that the child has not yet authenticated, it will be changed to 0x03, indicating no relationship.

3.2.2.17 NLME-LEAVE.indication

This primitive allows the next higher layer of a ZigBee device to be notified if that ZigBee device has left the network or if a neighboring device has left the network.

3.2.2.17.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-LEAVE.indication       {
                            DeviceAddress,
                            Rejoin
                            }
```

Table 3.23specifies the parameters for the NLME-LEAVE.indication primitive.

Table 3.23 NLME-LEAVE.indication Parameters

Name	Type	Valid Range	Description
DeviceAddress	64-bit IEEE address	Any 64-bit, IEEE address	The 64-bit IEEE address of an entity that has removed itself from the network or NULL in the case that the device issuing the primitive has been removed from the network by its parent.
Rejoin	Boolean	TRUE or FALSE	This parameter has a value of TRUE if the device being asked to leave the current parent is requested to rejoin the network. Otherwise, this parameter has a value of FALSE.

3.2.2.17.2 When Generated

This primitive is generated by the NLME of a ZigBee coordinator or ZigBee router and issued to its next higher layer on receipt of a broadcast leave command pertaining to a device on its PAN. It is also generated by the NLME of a ZigBee router or end device and issued to its next higher layer to indicate that it has been successfully removed from the network by its associated router or ZigBee coordinator.

3.2.2.17.3 Effect on Receipt

On receipt of this primitive, the next higher layer of a ZigBee coordinator or ZigBee router is notified that a device, formerly on the network, has left. The primitive can also inform the next higher layer of a ZigBee router or end device that it has been removed from the network by its associated ZigBee router or ZigBee coordinator parent. In this case, the value of the Rejoin parameter indicates to the next higher layer whether the peer entity on the parent device wishes the device that has been removed to rejoin the same network.

When the local device receives a NLME-LEAVE.indication with Rejoin set to FALSE it shall remove any persistent data within the stack related to the leaving device.

3.2.2.18 NLME-LEAVE.confirm

This primitive allows the next higher layer of the initiating device to be notified of the results of its request for itself or another device to leave the network.

3.2.2.18.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-LEAVE.confirm        {
                          Status,
                          DeviceAddress
                          }
```

Table 3.24 specifies the parameters for the NLME-LEAVE.confirm primitive.

Table 3.24 NLME-LEAVE.confirm Parameters

Name	Type	Valid Range	Description
Status	Status	SUCCESS, INVALID_REQUEST, UNKNOWN_DEVICE or any status returned by the MCPS-DATA.confirm primitive	The status of the corresponding request.
DeviceAddress	64-bit IEEE address	Any 64-bit, IEEE address	The 64-bit IEEE address in the request to which this is a confirmation or null if the device requested to remove itself from the network.

3.2.2.18.2 When Generated

This primitive is generated by the initiating NLME and issued to its next higher layer in response to an NLME-LEAVE.request primitive. If the request was successful, the Status parameter indicates a successful leave attempt. Otherwise, the Status parameter indicates an error code of INVALID_REQUEST, UNKNOWN_DEVICE or a status delivered by the MCPS-DATA.confirm primitive. The reasons for these status values are fully described in section 3.2.2.16.3.

3.2.2.18.3 Effect on Receipt

On receipt of this primitive, the next higher layer of the initiating device is notified of the results of its request for itself or a child device to leave the network.

3.2.2.19 NLME-RESET.request

This primitive allows the next higher layer to request the NWK layer to perform a reset operation.

3.2.2.19.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-RESET.request          {
                            WarmStart
                            }
```

Table 3.25 specifies the parameters for this primitive.

Table 3.25 NLME-RESET.request Parameters

Name	Type	Valid Range	Description
WarmStart	Boolean	TRUE or FALSE	This parameter has a value of FALSE if the request is expected reset all stack values to their initial default values. If this value is TRUE, the device is expected to resume operations according to the NIB settings prior to the call.

3.2.2.19.2 When Generated

This primitive is generated by the next higher layer and issued to its NLME to request the NWK layer be reset to its initial condition, or that it resume operations according to its current NIB values prior to this primitive being issued.

3.2.2.19.3 Effect on Receipt

On receipt of this primitive, where the WarmStart parameter has a value of FALSE, the NLME issues the MLME-RESET.request primitive to the MAC sub-layer with the SetDefaultPIB parameter set to TRUE. On receipt of the corresponding MLME-RESET.confirm primitive, the NWK layer resets itself by clearing all internal variables, routing table and route discovery table entries and by setting all NIB attributes to their default values. Once the NWK layer is reset, the NLME issues the NLME-RESET.confirm with the Status parameter set to SUCCESS if the MAC sub-layer was successfully reset or DISABLE_TRX_FAILURE otherwise.

On receipt of this primitive where WarmStart is set to TRUE, the network layer should not modify any NIB values, but rather should resume normal network operations and consider itself part of the network specified in the NIB. Routing table values and neighbor table values should be cleared. The method by which the network and MAC layers attributes are pre-loaded is left to the implementer.

If this primitive is issued to the NLME of a device that is currently joined to a network, any required leave attempts using the NLME-LEAVE.request primitive should be made *a priori* at the discretion of the next higher layer.

3.2.2.20 NLME-RESET.confirm

This primitive allows the next higher layer of the initiating device to be notified of the results of the request to reset the NWK layer.

3.2.2.20.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

NLME-RESET.confirm	{
	Status
	}

Table 3.26 specifies the parameters for this primitive.

Table 3.26 NLME-RESET.confirm Parameters

Name	Type	Valid Range	Description
Status	Status	Any status value returned from the MLME-RESET.confirm primitive (see [B1])	The result of the reset operation

3.2.2.20.2 When Generated

This primitive is generated by the NLME and issued to its next higher layer in response to an NLME-RESET.request primitive. If the request was successful, the Status parameter will have a value of SUCCESS. Otherwise, the Status parameter will indicate an error code of DISABLE_TRX_FAILURE. The reasons for these status values are fully described in section 3.2.2.19.3.

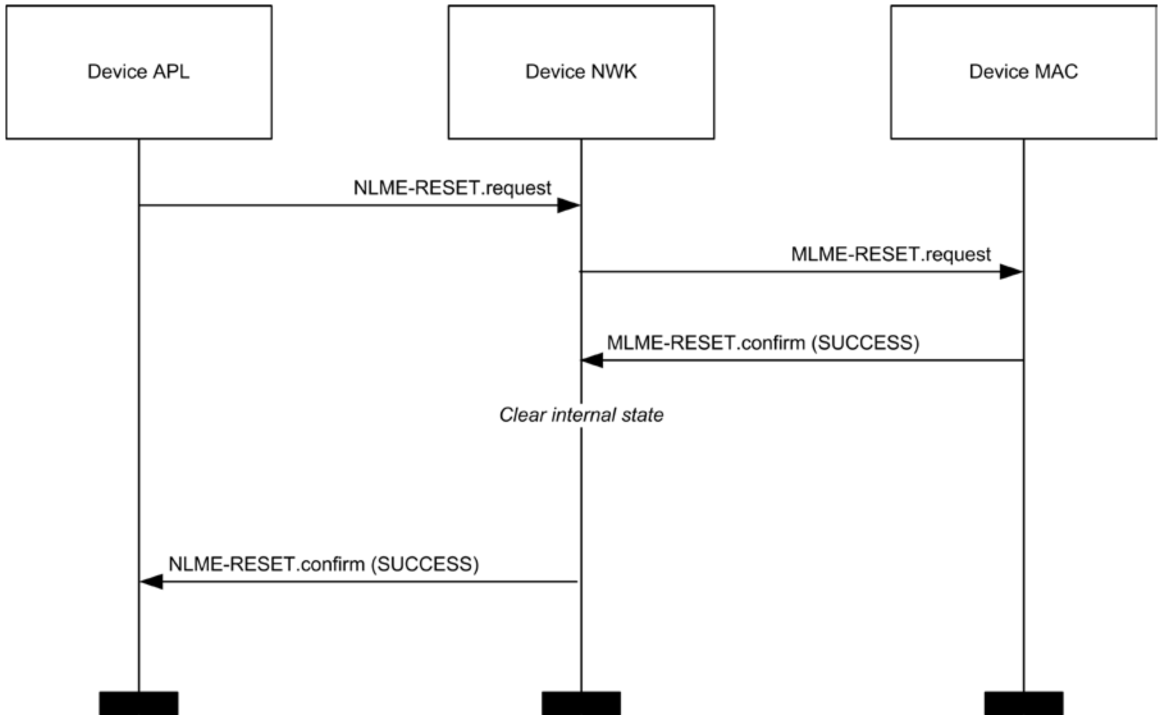
3.2.2.20.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to reset the NWK layer.

3.2.2.21 Network Layer Reset Message Sequence Chart

Figure 3.2 illustrates the sequence of messages necessary for resetting the NWK layer.

Figure 3.2 Message Sequence Chart for Resetting the Network Layer



3.2.2.22 NLME-SYNC.request

This primitive allows the next higher layer to synchronize or extract data from its ZigBee coordinator or router.

3.2.2.22.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-SYNC.request          {
                           Track
                           }
```

Table 3.27 specifies the parameters for this primitive.

Table 3.27 NLME-SYNC.request Parameters

Name	Type	Valid Range	Description
Track	Boolean	TRUE or FALSE	Whether or not the synchronization should be maintained for future beacons.

3.2.2.22.2 When Generated

This primitive is generated whenever the next higher layer wishes to achieve synchronization or check for pending data at its ZigBee coordinator or router.

3.2.2.22.3 Effect on Receipt

If the Track parameter is set to FALSE and the device is operating on a non-beacon enabled network, the NLME issues the MLME-POLL.request primitive to the MAC sub-layer. On receipt of the corresponding MLME-POLL.confirm primitive, the NLME issues the NLME-SYNC.confirm primitive with the Status parameter set to the value reported in the MLME-POLL.confirm.

If the Track parameter is set to FALSE and the device is operating on a beacon enabled network, the NLME first sets the *macAutoRequest* PIB attribute in the MAC sub-layer to TRUE by issuing the MLME-SET.request primitive. It then issues the MLME-SYNC.request primitive with the TrackBeacon parameter set to FALSE. The NLME then issues the NLME-SYNC.confirm primitive with the Status parameter set to SUCCESS.

If the Track parameter is set to TRUE and the device is operating on a non-beacon enabled network, the NLME will issue the NLME-SYNC.confirm primitive with a Status parameter set to INVALID_PARAMETER.

If the Track parameter is set to TRUE and the device is operating on a beacon enabled network, the NLME first sets the *macAutoRequest* PIB attribute in the MAC sub-layer to TRUE by issuing the MLME-SET.request primitive. It then issues the MLME-SYNC.request primitive with the TrackBeacon parameter set to TRUE. The NLME then issues the NLME-SYNC.confirm primitive with the Status parameter set to SUCCESS.

3.2.2.23 NLME-SYNC-LOSS.indication

This primitive allows the next higher layer to be notified of the loss of synchronization at the MAC sub-layer.

3.2.2.23.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

NLME-SYNC-LOSS.indication	{
	}

This primitive has no parameters.

3.2.2.23.2 When Generated

This primitive is generated upon receipt of a loss of synchronization notification from the MAC sub-layer via the MLME-SYNC-LOSS.indication primitive with a LossReason of BEACON_LOST. This follows a prior NLME-SYNC.request primitive being issued to the NLME.

3.2.2.23.3 Effect on Receipt

The next higher layer is notified of the loss of synchronization with the beacon.

3.2.2.24 NLME-SYNC.confirm

This primitive allows the next higher layer to be notified of the results of its request to synchronize or extract data from its ZigBee coordinator or router.

3.2.2.24.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

NLME-SYNC.confirm	{
	Status
	}

Table 3.28 specifies the parameters for this primitive.

Table 3.28 NLME-SYNC.confirm Parameters

Name	Type	Valid Range	Description
Status	Status	SUCCESS, SYNC_FAILURE, INVALID_PARAMETER, or any status value returned from the MLME_POLL.confirm primitive (see [B1])	The result of the request to synchronize with the ZigBee coordinator or router.

3.2.2.24.2 When Generated

This primitive is generated by the initiating NLME and issued to its next higher layer in response to an NLME-SYNC.request primitive. If the request was successful, the Status parameter indicates a successful synchronization attempt. Otherwise, the Status parameter indicates an error code. The reasons for these status values are fully described in section 3.2.2.22.2.

3.2.2.24.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to synchronize or extract data from its ZigBee coordinator or router. If the NLME has been successful, the Status parameter will be set to SUCCESS. Otherwise, the Status parameter indicates the error.

3.2.2.25 Message Sequence Charts for Synchronization

Figure 3.3 and Figure 3.4 illustrate the sequence of messages necessary for a device to successfully syn-chronize with a ZigBee coordinator or ZigBee router. Figure 3.3 illustrates the case for a non-beaconing network, and Figure 3.4 illustrates the case for a beaconing network.

Figure 3.3 Message Sequence Chart for Synchronizing in a Non-Beaconing Network

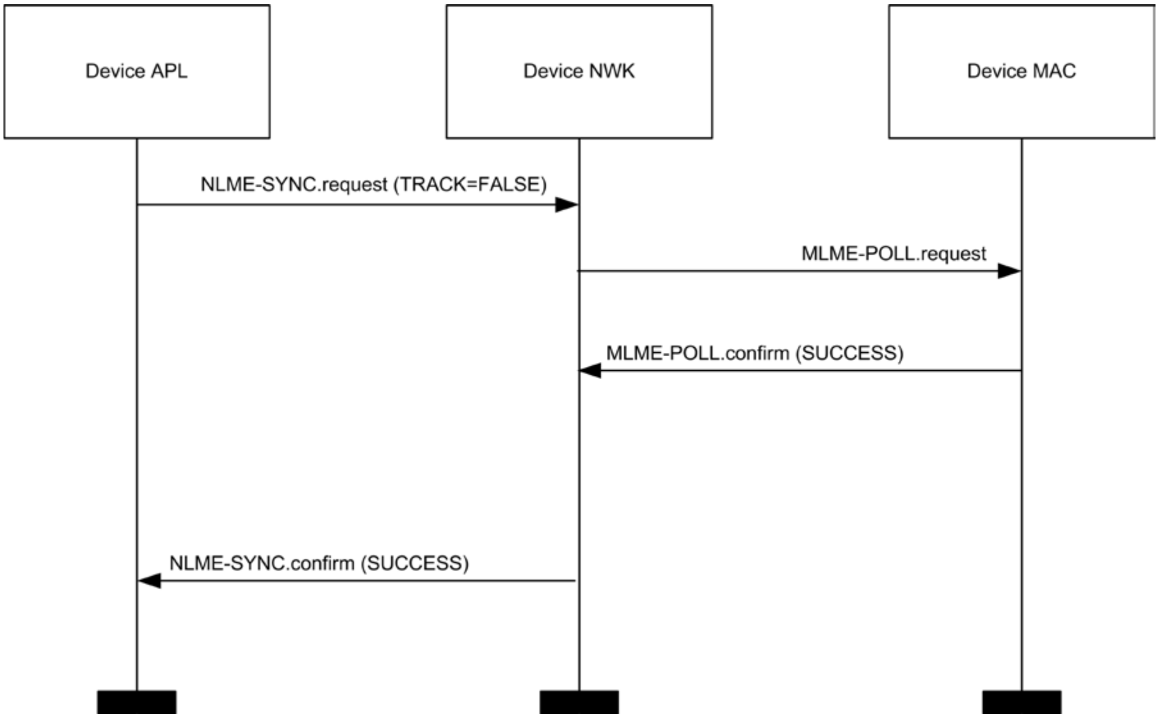
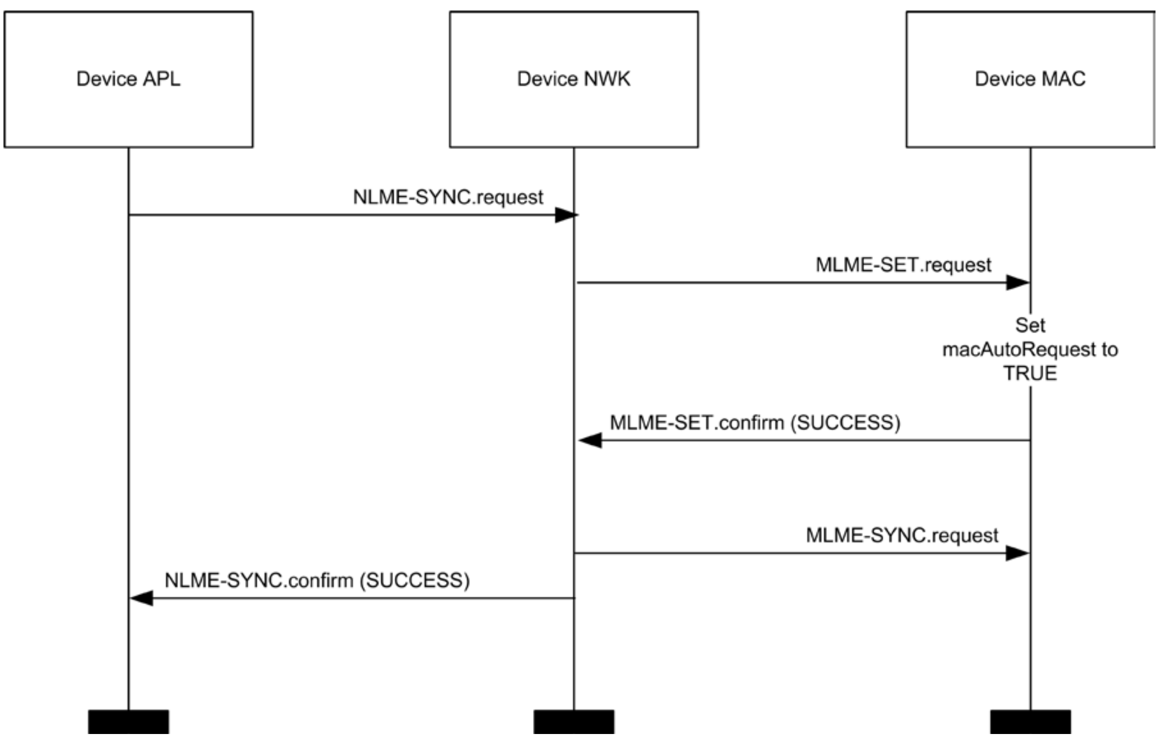


Figure 3.4 Message Sequence Chart for Synchronizing in a Beacon-Enabled Network



3.2.2.26 NLME-GET.request

This primitive allows the next higher layer to read the value of an attribute from the NIB.

3.2.2.26.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-GET.request        {
                        NIBAttribute
                        }
```

Table 3.29 specifies the parameters for this primitive.

Table 3.29 NLME-GET.request Parameters

Name	Type	Valid Range	Description
NIBAttribute	Integer	See Table 3.49.	The identifier of the NIB attribute to read.

3.2.2.26.2 When Generated

This primitive is generated by the next higher layer and issued to its NLME in order to read an attribute from the NIB.

3.2.2.26.3 Effect on Receipt

On receipt of this primitive, the NLME attempts to retrieve the requested NIB attribute from its database. If the identifier of the NIB attribute is not found in the database, the NLME issues the NLME-GET.confirm primitive with a status of UNSUPPORTED_ATTRIBUTE.

If the requested NIB attribute is successfully retrieved, the NLME issues the NLME-GET.confirm primitive with a status of SUCCESS and the NIB attribute identifier and value.

3.2.2.27 NLME-GET.confirm

This primitive reports the results of an attempt to read the value of an attribute from the NIB.

3.2.2.27.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-GET.confirm        {
                        Status,
                        NIBAttribute,
                        NIBAttributeLength,
                        NIBAttributeValue
                        }
```


Table 3.30 specifies the parameters for this primitive.

Table 3.30 NLME-GET.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS or UNSUPPORT-ED_ATTRIBUTE	The results of the request to read a NIB attribute value.
NIBAttribute	Integer	See Table 3.49.	The identifier of the NIB attribute that was read.
NIBAttributeLength	Integer	0x0000 – 0xffff	The length, in octets, of the attribute value being returned.
NIBAttributeValue	Various	Attribute specific (see Table 3.49)	The value of the NIB attribute that was read.

3.2.2.27.2 When Generated

This primitive is generated by the NLME and issued to its next higher layer in response to an NLME-GET.request primitive. This primitive returns either a status of SUCCESS, indicating that the request to read a NIB attribute was successful, or an error code of UNSUPPORTED_ATTRIBUTE. The reasons for these status values are fully described in section 3.2.2.26.3.

3.2.2.27.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to read a NIB attribute. If the request to read a NIB attribute was successful, the Status parameter will be set to SUCCESS. Otherwise, the Status parameter indicates the error.

3.2.2.28 NLME-SET.request

This primitive allows the next higher layer to write the value of an attribute into the NIB.

3.2.2.28.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-SET.request          {
                          NIBAttribute,
                          NIBAttributeLength,
                          NIBAttributeValue
                          }
```


Table 3.31 specifies the parameters for this primitive.

Table 3.31 NLME-SET.request Parameters

Name	Type	Valid Range	Description
NIBAttribute	Integer	See Table 3.49.	The identifier of the NIB attribute to be written.
NIBAttributeLength	Integer	0x0000 – 0xffff	The length, in octets, of the attribute value being set.
NIBAttributeValue	Various	Attribute specific (see Table 3.49)	The value of the NIB attribute that should be written.

3.2.2.28.2 When Generated

This primitive is to be generated by the next higher layer and issued to its NLME in order to write the value of an attribute in the NIB.

3.2.2.28.3 Effect on Receipt

On receipt of this primitive the NLME attempts to write the given value to the indicated NIB attribute in its database. If the NIBAttribute parameter specifies an attribute that is not found in the database, the NLME issues the NLME-SET.confirm primitive with a status of UNSUPPORTED_ATTRIBUTE. If the NIBAttributeValue parameter specifies a value that is out of the valid range for the given attribute, the NLME issues the NLME-SET.confirm primitive with a status of INVALID_PARAMETER.

If the requested NIB attribute is successfully written, the NLME issues the NLME-SET.confirm primitive with a status of SUCCESS.

3.2.2.29 NLME-SET.confirm

This primitive reports the results of an attempt to write a value to a NIB attribute.

3.2.2.29.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-SET.confirm          {
                          Status,
                          NIBAttribute
                          }
```


Table 3.32 specifies the parameters for this primitive.

Table 3.32 NLME-SET.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS, INVALID_PARAMETER or UNSUPPORTED_ATTRIBUTE	The result of the request to write the NIB attribute.
NIBAttribute	Integer	See Table 3.49.	The identifier of the NIB attribute that was written.

3.2.2.29.2 When Generated

This primitive is generated by the NLME and issued to its next higher layer in response to an NLME-SET.request primitive. This primitive returns a status of either SUCCESS, indicating that the requested value was written to the indicated NIB attribute, or an error code of INVALID_PARAMETER or UNSUPPORTED_ATTRIBUTE. The reasons for these status values are fully described in section 3.2.2.28.3.

3.2.2.29.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to write the value of a NIB attribute. If the requested value was written to the indicated NIB attribute, the Status parameter will be set to SUCCESS. Otherwise, the Status parameter indicates the error.

3.2.2.30 NLME-NWK-STATUS.indication

This primitive allows the next higher layer to be notified of network failures.

3.2.2.30.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-NWK-STATUS.indication  {
                            Status,
                            NetworkAddr
                            }
```

Table 3.33 specifies the parameters for this primitive.

Table 3.33 NLME-NWK-STATUS.indication Parameters

Name	Type	Valid Range	Description
Status	Status	Any network status code (see Table 3.42)	The error code associated with the failure.
NetworkAddr	Integer	0x0000 – 0xfff7	The 16-bit network address of the device associated with the status information.

3.2.2.30.2 When Generated

This primitive is generated by the NWK layer on a device and passed to the next higher layer when one of the following occurs:

- The device has failed to discover or repair a route to the destination given by the NetworkAddr parameter.
- The NWK layer on that device has failed to deliver a frame to its end device child with the 16-bit network address given by the NetworkAddr parameter, due to one of the reasons given in Table 3.42.
- The NWK layer has received a network status command frame destined for the device. In this case, the values of the NetworkAddr and Status parameters will reflect the values of the destination address and error code fields of the command frame.

3.2.2.30.3 Effect on Receipt

The next higher layer is notified of a failure to communicate with the identified device.

3.2.2.31 NLME-ROUTE-DISCOVERY.request

This primitive allows the next higher layer to initiate route discovery.

3.2.2.31.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-ROUTE-DISCOVERY.request     {
                                 DstAddrMode,
                                 DstAddr,
                                 Radius
                                 NoRouteCache
                                 }
```

Table 3.34 specifies the parameters for this primitive.

Table 3.34 NLME-ROUTE-DISCOVERY.request Parameters

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x00 – 0x02	A parameter specifying the kind of destination address provided. The DstAddrMode parameter may take one of the following three values: 0x00 = No destination address 0x01 = 16-bit network address of a multicast group 0x02 = 16-bit network address of an individual device

Name	Type	Valid Range	Description
DstAddr	16-bit network address	Any network address or multicast address	The destination of the route discovery. If the DstAddrMode parameter has a value of 0x00 then no DstAddr will be supplied. This indicates that the route discovery will be a many-to-one discovery with the device issuing the discovery command as a target. If the DstAddrMode parameter has a value of 0x01, indicating multicast route discovery then the destination will be the 16-bit network address of a multicast group. If the DstAddrMode parameter has a value of 0x02, this indicates unicast route discovery. The DstAddr will be the 16-bit network address of a device to be discovered.
Radius	Integer	0x00 – 0xff	This optional parameter describes the number of hops that the route request will travel through the network.
NoRouteCache	Boolean	TRUE or FALSE	In the case where DstAddrMode has a value of zero, indicating many-to-one route discovery, this flag determines whether the NWK should establish a route record table. TRUE = no route record table should be established FALSE = establish a route record table

3.2.2.31.2 When Generated

This primitive is generated by the next higher layer of a ZigBee coordinator or router and issued to its NLME to request the initiation of route discovery.

3.2.2.31.3 Effect on Receipt

On receipt of this primitive by the NLME of a ZigBee end device, the NLME will issue the NLME-ROUTE-DISCOVERY.confirm primitive to the next higher layer with a status value of INVALID_REQUEST.

On receipt of this primitive by the NLME with the DstAddrMode parameter not equal to 0x00 and the DstAddr parameter equal to a broadcast address, the NLME will issue the NLME-ROUTE-DISCOVERY.confirm primitive to the next higher layer with a status value of INVALID_REQUEST.

On receipt of this primitive by the NLME of a ZigBee router or ZigBee coordinator with no routing capacity and with the DstAddrMode parameter equal to 0x01 or 0x02, the NLME will issue the NLME-ROUTE-DISCOVERY.confirm to the next higher layer with a status value of ROUTE_ERROR and a NetworkStatusCode value of 0x04 indicating no routing capacity.

On receipt of this primitive by a ZigBee router or ZigBee coordinator that has routing capacity, with the DstAddrMode parameter equal to 0x02, the NLME will initiate discovery of a unicast route between the current device and the network device with the 16-bit network address given by the DstAddr parameter. The procedure for initiating discovery of a unicast route is outlined in section 3.6.3.5.1.

On receipt of this primitive by a ZigBee router or ZigBee coordinator that has routing capacity, with the DstAddrMode parameter equal to 0x01, the NLME will first check to see if the device is a member of the multicast group identified by the DstAddr parameter by checking if the *nwkGroupIDTable* attribute of the NIB contains an entry corresponding to the destination address. If the device is a member of the multicast group, then the NLME will immediately issue the NLME-ROUTE-DISCOVERY.confirm primitive with a status value of SUCCESS and discontinue further processing of the NLME-ROUTE-DISCOVERY.request primitive. If the device is not a member of the multicast group, the NLME will initiate discovery of a unicast route between the current device and the multicast group identified by the DstAddr parameter.

On receipt of this primitive on a ZigBee router or ZigBee coordinator with the DstAddrMode parameter equal to 0x00, the NLME will initiate many-to-one route discovery. The procedure for initiating many-to-one route discovery is outlined in section 3.6.3.5.1.

In each of the three cases of actual route discovery described above, the NLME will initiate route discovery by attempting to transmit a route discovery command frame using the MCPS-DATA.request primitive of the MAC sub-layer. If a value has been supplied for the optional Radius parameter, that value will be placed in the Radius field of the NWK header of the outgoing frame. If a value has not been supplied then the radius field of the NWK header will be set to twice the value of the *nwkMaxDepth* attribute of the NIB as would be the case for data frame transmissions. If the MAC sub-layer fails, for any reason, to transmit the route request command frame, the NLME will issue the ROUTE-DISCOVERY.confirm primitive to the next higher layer with a Status parameter value equal to that returned by the MCPS-DATA.confirm. If the route discovery command frame is sent successfully and if the DstAddrMode parameter has a value of 0x00, indicating many-to-one route discovery, the NLME will immediately issue the ROUTE-DISCOVERY.confirm primitive with a value of SUCCESS. Otherwise, the NLME will wait until either a route reply command frame is received or the route discovery operation times out as described in section 3.6.3.5. If a route reply command frame is received before the route discovery operation times out, the NLME will issue the NLME-ROUTE-DISCOVERY.confirm primitive to the next higher layer with a status value of SUCCESS. If the operation times out, it will issue the NLME_ROUTE-DISCOVERY.confirm primitive with a Status of ROUTE_ERROR and with a Network-StatusCode value reflecting the reason for failure as described in Table 3.42.

3.2.2.32 NLME_ROUTE-DISCOVERY.confirm

This primitive informs the next higher layer about the results of an attempt to initiate route discovery.

3.2.2.32.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME_ROUTE-DISCOVERY.confirm    {
                                Status,
                                NetworkStatusCode
                                }
```

Table 3.35 specifies the parameters for the NLME-ROUTE-DISCOVERY.confirm primitive.

Table 3.35 NLME_ROUTE-DISCOVERY.confirm Parameters

Name	Type	Valid Range	Description
Status	status value	INVALID_REQUEST, ROUTE_ERROR or any status value returned by the MCPS-DATA.confirm primitive	The status of an attempt to initiate route discovery.
NetworkStatusCode	Network status code	See Table 3.42.	In the case where the Status parameter has a value of ROUTE_ERROR, this code gives further information about the kind of error that occurred. Otherwise, it should be ignored.

3.2.2.32.2 When Generated

This primitive is generated by the NLME and passed to the next higher layer as a result of an attempt to initiate route discovery.

3.2.2.32.3 Effect on Receipt

The next higher layer is informed of the status of its attempt to initiate route discovery. Possible values for the Status parameter and the circumstances under which they are generated are described in section 3.2.2.32.3.

3.3 Frame Formats

This section specifies the format of the NWK frame (NPDU). Each NWK frame consists of the following basic components:

- A NWK header, which comprises frame control, addressing and sequencing information
- A NWK payload, of variable length, which contains information specific to the frame type

The frames in the NWK layer are described as a sequence of fields in a specific order. All frame formats in this section are depicted in the order in which they are transmitted by the MAC sub-layer, from left to right, where the leftmost bit is transmitted first. Bits within each field are numbered from 0 (leftmost and least significant) to k-1 (rightmost and most significant), where the length of the field is k bits. Fields that are longer than a single octet are sent to the MAC sub-layer in the order from the octet containing the lowest-numbered bits to the octet containing the highest-numbered bits.

3.3.1 General NPDU Frame Format

The NWK frame format is composed of a NWK header and a NWK payload. The fields of the NWK header appear in a fixed order. The NWK frame shall be formatted as illustrated in Figure 3.5.

Figure 3.5 General NWK Frame Format

Octets: 2	2	2	1	1	0/8	0/8	0/1	Variable	Variable
Frame control	Destination address	Source address	Radius	Sequence number	Destination IEEE Address	Source IEEE Address	Multicast control	Source route subframe	Frame payload
NWK Header									Payload

3.3.1.1 Frame Control Field

The frame control field is 16 bits in length and contains information defining the frame type, addressing and sequencing fields and other control flags. The frame control field shall be formatted as illustrated in Figure 3.6.

Figure 3.6 Frame Control Field

Bits: 0-1	2-5	6-7	8	9	10	11	12	13	14-15
Frame type	Protocol version	Discover route	Multicast flag	Security	Source Route	Destination IEEE Address	Source IEEE Address	End Device Initiator	Reserved

Table 3.36shows the allowable frame control sub-field configurations for NWK data frames. Note that all frames listed below will have a frame type sub-field equal to 00 indicating data and a protocol version sub-field reflecting the version of the ZigBee specification implemented.

Table 3.36 Allowable Frame Control Sub-Field Configurations

Data Transmission Method	Discover Route	Multicast	Security	Destination IEEE Address	Source IEEE Address
Unicast	00 or 01	0	0 or 1	0 or 1	0 or 1
Broadcast	00	0	0 or 1	0	0 or 1
Multicast	00	1	0 or 1	0	0 or 1
Source routed	00	0	0 or 1	0 or 1	0 or 1

3.3.1.1.1 Frame Type Sub-Field

The frame type sub-field is 2 bits in length and shall be set to one of the non-reserved values listed in Table 3.37.

Table 3.37 Values of the Frame Type Sub-Field

Frame Type Value b_1 b_0	Frame Type Name
00	Data
01	NWK command
10	Reserved
11	Inter-PAN

3.3.1.1.2 Protocol Version Sub-Field

The protocol version sub-field is 4 bits in length and shall be set to a number reflecting the ZigBee NWK protocol version in use. The protocol version in use on a particular device shall be made available as the value of the NWK constant *nwkcProtocolVersion*.

3.3.1.1.3 Discover Route Sub-Field

The discover route sub-field may be used to control route discovery operations for the transit of this frame (see section 3.6.3.5).

Table 3.38 Values of the Discover Route Sub-Field

Discover Route Field Value	Field Meaning
0x00	Suppress route discovery
0x01	Enable route discovery
0x02, 0x03	Reserved

For NWK layer command frames, the discover route sub-field shall be set to 0x00 indicating suppression of route discovery.

3.3.1.1.4 Multicast Flag Sub-Field

The multicast flag sub-field is 1 bit in length and has the value 0 if the frame is a unicast or broadcast frame and the value 1 if it is a multicast frame. The multicast control field of the NWK header shall be present only if the multicast flag has the value 1.

3.3.1.1.5 Security Sub-Field

The security sub-field shall have a value of 1 if, and only if, the frame is to have NWK security operations enabled. If security for this frame is implemented at another layer or disabled entirely, it shall have a value of 0.

3.3.1.1.6 Source Route Sub-Field

The source route sub-field shall have a value of 1 if and only if a source route subframe is present in the NWK header. If the source route subframe is not present, the source route sub-field shall have a value of 0.

3.3.1.1.7 Destination IEEE Address Sub-Field

The destination IEEE address sub-field shall have a value of 1 if, and only if, the NWK header is to include the full IEEE address of the destination.

3.3.1.1.8 Source IEEE Address Sub-Field

The source IEEE address sub-field shall have a value of 1 if, and only if, the NWK header is to include the full IEEE address of the source device.

3.3.1.1.9 End Device Initiator

If the source of the message is an end device and the *nwkParentInformation* field of the NIB is a value other than 0, then this sub-field shall be set to 1. Otherwise this sub-field shall be set to 0. After validating the source (see section 3.6.2.2), a router parent device shall clear this field when relaying a message sent by one of its end device children.

3.3.1.2 Destination Address Field

The destination address field shall always be present and shall be 2 octets in length. If the multicast flag sub-field of the frame control field has the value 0, the destination address field shall hold the 16-bit network address of the destination device or a broadcast address (see Table 3.59). If the multicast flag sub-field has the value 1, the destination address field shall hold the 16-bit Group ID of the destination multicast group. Note that the network address of a device shall be set to the value of the *macShortAddress* attribute of the MAC PIB.

3.3.1.3 Source Address Field

The source address field shall always be present. It shall always be 2 octets in length and shall hold the network address of the source device of the frame. Note that the network address of a device shall be set to value of the *macShortAddress* attribute of the MAC PIB.

3.3.1.4 Radius Field

The radius field shall always be present. It will be 1 octet in length and specifies the range of a radius-limited transmission. The field shall be decremented by 1 by each receiving device.

3.3.1.5 Sequence Number Field

The sequence number field is present in every frame and is 1 octet in length. The sequence number value shall be incremented by 1 with each new frame transmitted. The values of the source address and sequence number fields of a frame, taken as a pair, may be used to uniquely identify a frame within the constraints imposed by the sequence number's one-octet range. For more details on the use of the sequence number field, see section 3.6.2.

3.3.1.6 Destination IEEE Address Field

The destination IEEE address field, if present, contains the 64-bit IEEE address corresponding to the 16-bit network address contained in the destination address field of the NWK header. Upon receipt of a frame containing a 64-bit IEEE address, the contents of the *nwkAddressMap* and neighbor table should be checked for consistency, and updated if necessary. Section 3.6.1.9.2 describes the actions to take in detecting address conflicts. If the 16-bit network address is a broadcast or multicast address then the destination IEEE address field shall not be present.

3.3.1.7 Source IEEE Address Field

The source IEEE address field, if present, contains the 64-bit IEEE address corresponding to the 16-bit network address contained in the source address field of the NWK header. Upon receipt of a frame containing a 64-bit IEEE address, the contents of the *nwkAddressMap* and Neighbor Table should be checked for consistency, and updated if necessary. Section 3.6.1.9.2 describes the actions to take in detecting address conflicts.

3.3.1.8 Multicast Control Field

The multicast control sub-field is 1 octet in length and shall only be present if the multicast flag sub-field has a value of 1. It is divided into three sub-fields as illustrated in Figure 3.7.

Figure 3.7 Multicast Control Field Format

Bits: 0 – 1	**2 – 4**	**5 – 7**
Multicast mode	NonmemberRadius	MaxNonmemberRadius

3.3.1.8.1 Multicast Mode Sub-Field

The multicast mode sub-field indicates whether the frame is to be transmitted using member or non-member mode. Member mode is used to propagate multicasts between the devices that are members of the destination group. Non-member mode is used to transmit a multicast frame from a device that is not a member of the multicast group to a device that is a member of the multicast group.

Table 3.39 Values of the Multicast Mode Sub-Field

Multicast Mode Field Value	Field Meaning
0x0	Non-member mode
0x1	Member mode
0x2	Reserved
0x3	Reserved

3.3.1.8.2 NonmemberRadius Sub-Field

The nonmemberradius sub-field indicates the range of a member mode multicast when relayed by devices that are not members of the destination group. Receiving devices that are members of the destination group will set this field to the value of the MaxNonmemberRadius sub-field. The originating device and receiving devices that are not members of the destination group will discard the frame if the NonmemberRadius sub-field has value 0 and will decrement the field if the NonmemberRadius sub-field has a value in the range 0x01 through 0x06. A value of 0x07 indicates an infinite range and is not decremented.

The value of the NonmemberRadius sub-field will never exceed the value of the MaxNonmemberRadius sub-field.

3.3.1.8.3 MaxNonmemberRadius Sub-Field

The maximum value of the NonmemberRadius sub-field for this frame.

3.3.1.9 Source Route Subframe Field

The source route subframe field shall only be present if the source route sub-field of the frame control field has a value of 1. It is divided into three sub-fields as illustrated in Figure 3.8.

Figure 3.8 Source Route Subframe Format

Octets: 1	1	Variable
Relay count	Relay index	Relay list

3.3.1.9.1 Relay Count Sub-Field

The relay count sub-field indicates the number of relays contained in the relay list sub-field of the source route subframe.

3.3.1.9.2 Relay Index

The relay index sub-field indicates the index of the next relay in the relay list sub-field to which the packet will be transmitted. This field is initialized to 1 less than the relay count by the originator of the packet, and is decremented by 1 by each receiving relay.

3.3.1.9.3 Relay List Sub-Field

The relay list sub-field shall contain the list of relay addresses. The relay closest to the destination shall be listed first. The relay closest to the originator shall be listed last.

3.3.1.9.4 Frame Payload Field

The frame payload field has a variable length and contains information specific to individual frame types.

3.3.2 Format of Individual Frame Types

There are two defined NWK frame types: data and NWK command. Each of these frame types is discussed in the following sections.

3.3.2.1 Data Frame Format

The data frame shall be formatted as illustrated in Figure 3.9.

Figure 3.9 Data Frame Format

Octets: 2	Variable	Variable
Frame control	Routing fields	Data payload
NWK header		NWK payload

The order of the fields of the data frame shall conform to the order of the general NWK frame format as illustrated in Figure 3.5.

3.3.2.1.1 Data Frame NWK Header Field

The data frame NWK header field shall contain the frame control field and an appropriate combination of routing fields as required.

In the frame control field, the frame type sub-field shall contain the value that indicates a data frame, as shown in Table 3.37. All other sub-fields shall be set according to the intended use of the data frame.

The routing fields shall contain an appropriate combination of address and broadcast fields, depending on the settings in the frame control field (see Figure 3.6).

3.3.2.1.2 Data Payload Field

The data frame data payload field shall contain the sequence of octets that the next higher layer has requested the NWK layer to transmit.

3.3.2.2 NWK Command Frame Format

The NWK command frame shall be formatted as illustrated in Figure 3.10.

Figure 3.10 NWK Command Frame Format

Octets: 2	**Variable**	**1**	**Variable**
Frame control	Routing fields	NWK command identifier	NWK command payload
NWK header		NWK payload	

The order of the fields of the NWK command frame shall conform to the order of the general NWK frame as illustrated in Figure 3.5.

3.3.2.2.1 NWK Command Frame NWK Header Field

The NWK header field of a NWK command frame shall contain the frame control field and an appropriate combination of routing fields as required.

In the frame control field, the frame type sub-field shall contain the value that indicates a NWK command frame, as shown in Table 3.37. All other sub-fields shall be set according to the intended use of the NWK command frame.

The routing fields shall contain an appropriate combination of address and broadcast fields, depending on the settings in the frame control field.

3.3.2.2.2 NWK Command Identifier Field

The NWK command identifier field indicates the NWK command being used. This field shall be set to one of the non-reserved values listed in Table 3.40.

3.3.2.2.3 NWK Command Payload Field

The NWK command payload field of a NWK command frame shall contain the NWK command itself.

3.4 Command Frames

The command frames defined by the NWK layer are listed in Table 3.40. The following sections detail how the NLME shall construct the individual commands for transmission.

For each of these commands, the following applies to the NWK header fields unless specifically noted in the clause on NWK header in each command:

- The frame type sub-field of the NWK frame control field shall be set to indicate that this frame is a NWK command frame.
- The discover route sub-field in the NWK header shall be set to suppress route discovery (see Table 3.38).
- The source address field in the NWK header shall be set to the address of the originating device.

Table 3.40 NWK Command Frames

Command Frame Identifier	Command Name	Reference
0x01	Route request	3.4.1
0x02	Route reply	3.4.2
0x03	Network Status	3.4.3
0x04	Leave	3.4.4
0x05	Route Record	3.4.5
0x06	Rejoin request	3.4.6
0x07	Rejoin response	3.4.7
0x08	Link Status	3.4.8
0x09	Network Report	3.4.9
0x0a	Network Update	3.4.10
0x0b	End Device Timeout Request	3.4.11
0x0c	End Device Timeout Response	3.4.12
0x0d – 0xff	Reserved	—

3.4.1 Route Request Command

The route request command allows a device to request other devices within radio range to engage in a search for a particular destination device and establish a state within the network that will allow messages to be routed to that destination more easily and economically in the future. The payload of a route request command shall be formatted as illustrated in Figure 3.11.

Figure 3.11 Route Request Command Frame Format

Octets: 1	1	2	1	0/8
Command options	Route request identifier	Destination address	Path cost	Destination IEEE Address
NWK command payload				

3.4.1.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in IEEE 802.15.4-2003 [B1], the following information shall be included in the MAC frame header:

- The destination PAN identifier shall be set to the PAN identifier of the device sending the route request command.
- The destination address shall be set to the broadcast address of 0xffff.
- The source address and PAN identifier shall be set to the network address and PAN identifier of the device sending the route request command, which may or may not be the device from which the command originated.
- The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. Because the frame is broadcast, no acknowledgment request shall be specified.
- The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

3.4.1.2 NWK Header Fields

In order for this route request to reach its destination and for the route discovery process to complete correctly, the following information must be provided:

- The destination address in the NWK header shall be set to the broadcast address for all routers and the coordinator (see Table 3.59).
- The source IEEE address sub-field of the frame control field shall be set to 1 and the source IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the originator of the frame.

3.4.1.3 NWK Payload Fields

The NWK frame payload contains a command identifier field, a command options field, the route request identifier field, the address of the intended destination, an up-to-date summation of the path cost, and the destination IEEE address.

The command frame identifier shall contain the value indicating a route request command frame.

3.4.1.3.1 Command Options Field

The format of the 8-bit command options field is shown in Figure 3.12.

Figure 3.12 Route Request Command Options Field

Bit: 0-2	3-4	5	6	7
Reserved	Many-to-one	Destination IEEE address	Multicast	Reserved

3.4.1.3.1.1 Many-to-One

The many-to-one field shall have one of the non-reserved values shown in Table 3.41.

Table 3.41 Many-to-One Field Values

Value	Description
0	The route request is not a many-to-one route request.
1	The route request is a many-to-one route request and the sender supports a route record table.
2	The route request is a many-to-one route request and the sender does not support a route record table.
3	Reserved

3.4.1.3.1.2 Destination IEEE Address

The destination IEEE address field is a single-bit field. It shall have a value of 1 if, and only if, the command frame contains the destination IEEE address. The Destination IEEE Address field should always be added if it is known.

3.4.1.3.1.3 Multicast Sub-Field

The multicast sub-field is a single-bit field. It shall have a value of 1 if, and only if, the command frame is a request for a route to a multicast group, in which case the destination address field contains the Group ID of the desired group.

3.4.1.3.2 Route Request Identifier

The route request identifier is an 8-bit sequence number for route requests and is incremented by 1 every time the NWK layer on a particular device issues a route request.

3.4.1.3.3 Destination Address

The destination address shall be 2 octets in length and represents the intended destination of the route request command frame.

3.4.1.3.4 Path Cost

The path cost field is eight bits in length and is used to accumulate routing cost information as a route request command frame moves through the network (see section 3.6.3.5.2).

3.4.1.3.5 Destination IEEE Address

The destination IEEE address shall be 8 octets in length and represents the IEEE address of the destination of the route request command frame. It shall be present only if the destination IEEE address sub-field of the command frame options field has a value of 1.

3.4.2 Route Reply Command

The route reply command allows the specified destination device of a route request command to inform the originator of the route request that the request has been received. It also allows ZigBee routers along the path taken by the route request to establish state information that will enable frames sent from the source device to the destination device to travel more efficiently. The payload of the route reply command shall be formatted as illustrated in Figure 3.13.

Figure 3.13 Route Reply Command Format

Octets: 1	1	2	2	1	0/8	0/8
Command options	Route request identifier	Originator address	Responder address	Path cost	Originator IEEE address	Responder IEEE address
NWK command payload						

3.4.2.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in IEEE 802.15.4-2003 [B1], the following information shall be included in the MAC frame header:

- The destination MAC address and PAN identifier shall be set to the network address and PAN identifier, respectively, of the first hop in the path back to the originator of the corresponding route request command frame. The destination PAN identifier shall be the same as the PAN identifier of the originator.
- The source MAC address and PAN identifier shall be set to the network address and PAN identifier of the device sending the route reply command, which may or may not be the device from which the command originated.
- The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. The transmission options shall be set to require acknowledgment. The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

3.4.2.2 NWK Header Fields

In order for this route reply to reach its destination and for the route discovery process to complete correctly, the following information must be provided:

- The source address in the NWK header shall be set to the 16-bit network address of the device transmitting the frame.
- The destination address field in the NWK header shall be set to the network address of the first hop in the path back to the originator of the corresponding route request.
- Since this is a NWK layer command frame, the source IEEE address sub-field of the frame control field shall be set to 1 and the source IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the originator of the frame. The destination IEEE address sub-field of the frame control field shall also have a value of 1 and the destination IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the first hop in the path back to the originator of the corresponding route request.
- The Sequence Number field in the NWK header shall be created for every hop during the route reply process. The Radius Field shall be set to *nwkMaxDepth* * 2 by the target of the route request. Every hop during the Route Reply process shall decrement the radius by 1. If the value of the radius in the received Route Reply message is 1, the relaying router shall set the radius of the message to 1. The Sequence Number shall be created as if it were a new frame from the device transmitting the frame replacing the sequence number with the device's next available sequence number. The Route Reply frame is not a forwarded frame, but is newly created by each hop during the route reply process.

3.4.2.3 NWK Payload Fields

The NWK frame payload contains a command identifier field, a command options field, the route request identifier, originator and responder addresses and an up-to-date summation of the path cost.

The command frame identifier shall contain the value indicating a route reply command frame.

3.4.2.3.1 Command Options Field

The format of the 8-bit command options field is shown in Figure 3.14.

Figure 3.14 Route Reply Command Options Field

Bit: 0 – 3	**4**	**5**	**6**	**7**
Reserved	Originator IEEE address	Responder IEEE address	Multicast	Reserved

3.4.2.3.1.1 Originator IEEE Address

The originator IEEE address sub-field is a single-bit field. It shall have a value of 1 if and only if the originator IEEE address field is included in the payload. This bit shall be set when *nwkUniqueAddr* is FALSE.

3.4.2.3.1.2 Responder IEEE Address

The responder IEEE address sub-field is a single-bit field. It shall have a value of 1 if, and only if, the responder IEEE address field is included in the payload. This bit shall be set when *nwkUniqueAddr* is FALSE and the multicast sub-field is set to 0.

3.4.2.3.1.3 Multicast Sub-Field

The multicast sub-field is a single-bit field. It shall have a value of 1 if and only if the command frame is a reply to a request for a route to a multicast group, in which case the responder address field contains the Group ID of the desired group.

3.4.2.3.2 Route Request Identifier

The route request identifier is the 8-bit sequence number of the route request to which this frame is a reply.

3.4.2.3.3 Originator Address

The originator address field shall be 2 octets in length and shall contain the 16-bit network address of the originator of the route request command frame to which this frame is a reply.

3.4.2.3.4 Responder Address

The responder address field shall be 2 octets in length and shall always be the same as the value in the destination address field of the corresponding route request command frame.

3.4.2.3.5 Path Cost

The path cost field is used to sum link cost as the route reply command frame transits the network (see section 3.6.3.5.3.

3.4.2.3.6 Originator IEEE Address

The originator IEEE address field shall be 8 octets in length and shall contain the 64-bit address of the originator of the route request command frame to which this frame is a reply. This field shall only be present if the originator IEEE address sub-field of the command options field has a value of 1.

3.4.2.3.7 Responder IEEE Address

The responder IEEE address field shall be 8 octets in length and shall contain the 64-bit address of the destination of the route request command frame to which this frame is a reply. This field shall only be present if the responder IEEE address sub-field of the command options field has a value of 1.

3.4.3 Network Status Command

A device uses the network status command to report errors and other conditions arising in the NWK layer of a particular device to the peer NWK layer entities of other devices in the network. The NWK status command may be also used to diagnose network problems, for example address conflicts. The payload of a network status command shall be formatted as illustrated in Figure 3.15.

Figure 3.15 Network Status Command Frame Format

Octets: 1	**2**
Status code	Destination address
NWK command payload	

3.4.3.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in IEEE 802.15.4-2003 [B1], the following information shall be provided:

- The destination MAC address and PAN identifier shall be set to the network address and PAN identifier, respectively, of the first hop in the path to the destination of the command frame or to the broadcast address 0xffff in the case where the command frame is being broadcast at the NWK layer.
- The source MAC address and PAN identifier shall be set to the network address and PAN identifier of the device sending the network status command.
- The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. The transmission options shall not be set to require acknowledgement if the destination MAC address is the broadcast address 0xffff.
- The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

3.4.3.2 NWK Header Fields

Network status commands may be either unicast or broadcast. The fields of the NWK header shall be set as follows:

- The source address field shall always be set to the 16-bit network address of the device originating the command frame.
- The source IEEE address sub-field of the frame control field shall be set to 1 and the source IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the originator of the frame.
- When sent in response to a routing error, the destination address field in the NWK header shall be set to the same value as the source address field of the data frame that encountered a forwarding failure.
- If and only if, the network status command frame is not broadcast, the destination IEEE address sub-field of the frame control field shall have a value of 1 and the destination IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE corresponding to the 16-bit network address in the destination address field if this IEEE address is known.

3.4.3.3 NWK Payload Fields

The NWK frame payload of the network status command frame contains a command frame identifier field, a status code field and a destination address field as described below. The command frame identifier shall be set to specify the network status command frame as defined in Table 3.40.

3.4.3.3.1 Status Code

The status code shall be set to one of the non-reserved values shown in Table 3.42.

Table 3.42 Status Codes for Network Status Command Frame

Value	Status Code
0x00	No route available
0x01	Tree link failure
0x02	Non-tree link failure
0x03	Low battery level
0x04	No routing capacity
0x05	No indirect capacity
0x06	Indirect transaction expiry
0x07	Target device unavailable
0x08	Target address unallocated
0x09	Parent link failure
0x0a	Validate route
0x0b	Source route failure
0x0c	Many-to-one route failure
0x0d	Address conflict
0x0e	Verify addresses
0x0f	PAN identifier update
0x10	Network address update
0x11	Bad frame counter
0x12	Bad key sequence number
0x13 – 0xff	Reserved

These status codes are used both as values for the status code field of a network status command frame and as values of the Status parameter of the NLME-NWK-STATUS.indication primitive. A brief explanation of each follows:

- **No route available:** Route discovery and/or repair has been attempted and no route to the intended destination address has been discovered.
- **Tree link failure:** The routing failure occurred as a result of the failure of an attempt to route the frame along the tree.
- **Non-tree link failure:** The routing failure did not occur as a result of an attempt to route along the tree.
- **Low battery level:** The frame was not relayed because the relaying device was running low on battery power.
- **No routing capacity:** The failure occurred because the relaying device has no routing capacity.
- **No indirect capacity:** The failure occurred as the result of an attempt to buffer a frame for a sleeping end device child and the relaying device had no buffer capacity to use.
- **Indirect transaction expiry:** A frame that was buffered on behalf of a sleeping end device child has been dropped as a result of a time-out.
- **Target device unavailable:** An end device child of the relaying device is for some reason unavailable.
- **Target address unallocated:** The frame was addressed to a non-existent end device child of the relaying device.
- **Parent link failure:** The failure occurred as a result of a failure in the RF link to the device’s parent. This status is only used locally on a device to indicate loss of communication with the parent.
- **Validate route:** The multicast route identified in the destination address field should be validated as described in section 3.6.3.6.
- **Source route failure:** Source routing has failed, probably indicating a link failure in one of the source route’s links.
- **Many-to-one route failure:** A route established as a result of a many-to-one route request has failed.
- **Address conflict:** The address in the destination address field has been determined to be in use by two or more devices.
- **Verify addresses:** The source device has the IEEE address in the Source IEEE address field and, if the Destination IEEE address field is present, the value it contains is the expected IEEE address of the destination.
- **PAN identifier update:** The operational network PAN identifier of the device has been updated.
- **Network address update:** The network address of the device has been updated.
- **Bad frame counter:** A frame counter reported in a received frame had a value less than or equal to that stored in *nwkSecurityMaterialSet*.
- **Bad key sequence number:** The key sequence number reported in a received frame did not match that of *nwkActiveKeySeqNumber*.

3.4.3.3.2 Destination Address

The destination address field is 2 octets in length and shall be present if, and only if, the network status command frame is being sent in response to a routing failure. In this case, it shall contain the destination address from the data frame that encountered the failure.

3.4.4 Leave Command

The leave command is used by the NLME to inform other devices on the network that a device is leaving the network or else to request that a device leave the network. The payload of the leave command shall be formatted as shown in Figure 3.16.

Figure 3.16 Leave Command Frame Format

1
Command options
NWK command payload

3.4.4.1 MAC Data Service Requirement

In order to transmit this command using the MAC data service, specified in IEEE 802.15.4-2003 [B1], the following information shall be provided:

- The destination MAC address and PAN identifier shall be set to the network address and PAN identifier, respectively, of the neighbor device to which the frame is being sent or else to the MAC broadcast address 0xffff in the case where the NWK header also contains a broadcast address.
- The source MAC address and PAN identifier shall be set to the network address and PAN identifier of the device sending the leave command.
- The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. Acknowledgment shall be requested.
- The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

3.4.4.2 NWK Header Fields

The NWK header fields of the leave command frame shall be set as follows:

- The source IEEE address sub-field of the frame control field shall be set to 1 and the source IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the originator of the frame.
- If the request sub-field of the command options field is set to 1 then the destination address field in the NWK header shall be set to the network address of the child device being requested to leave.
- If the request sub-field is set to 0 then the destination address field in the NWK header shall be set to 0xfffd so that the indication is received by devices with *macRxOnWhenIdle* equal to TRUE.
- The destination address sub-field of the frame control may be set to 0 or 1. The choice shall be based on whether the local device has knowledge of the IEEE address for the device being requested to leave. If the local device knows the IEEE address then the field shall be set to 1 and the destination IEEE address field shall be present..
- The radius field shall be set to 1.

3.4.4.3 NWK Payload Fields

The NWK payload of the leave command frame contains a command frame identifier field and a command options field. The command frame identifier field shall be set to specify the leave command frame as described in Table 3.40.

3.4.4.3.1 Command Options Field

The format of the 8-bit Command Options field is shown in Figure 3.17.

Figure 3.17 Leave Command Options Field

Bit: 0 – 4	5	6	7
Reserved	Rejoin	Request	Remove children

3.4.4.3.1.1 Rejoin Sub-Field

The Rejoin sub-field is a single-bit field. If the value of this sub-field is 1, the device that is leaving from its current parent will rejoin the network. If the value of this sub-field is 0, the device will not rejoin the network.

3.4.4.3.1.2 Request Sub-Field

The request sub-field is a single-bit field. If the value of this sub-field is 1, then the leave command frame is a request for another device to leave the network. If the value of this sub-field is 0, then the leave command frame is an indication that the sending device plans to leave the network.

3.4.4.3.1.3 Remove Children Sub-Field

The remove children sub-field is a single-bit field. If this sub-field has a value of 1, then the children of the device that is leaving the network will also be removed. If this sub-field has a value of 0, then the children of the device leaving the network will not be removed.

3.4.5 Route Record Command

The route record command allows the route taken by a unicast packet through the network to be recorded in the command payload and delivered to the destination device. The payload of the route record command shall be formatted as illustrated in Figure 3.18.

Figure 3.18 Route Record Command Format

Octets: 1	Variable
Relay count	Relay list
NWK command payload	

3.4.5.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in IEEE 802.15.4-2003 [B1], the following information shall be provided:

- The destination MAC address and PAN identifier shall be set to the network address and PAN identifier, respectively, of the neighbor device to which the frame is being sent.
- The source MAC address and PAN identifier shall be set to the network address and PAN identifier of the device sending the route record command.
- The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. Acknowledgment shall be requested.
- The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

3.4.5.2 NWK Header Fields

The NWK header fields of the route record command frame shall be set as follows:

- If the route record is being initiated as the result of a NLDE-DATA.request primitive from the next higher layer, the source address field shall be set to the 16-bit network address of the originator of the frame. If the route record is being initiated as a result of the relaying of a data frame on behalf of one of the device's end device children, the source address field shall contain the 16-bit network address of that end device child.
- The source IEEE address sub-field of the frame control field shall be set to 1 and the source IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address corresponding to the 16-bit network address contained in the source address field.
- The destination address field in the NWK header shall be set to the 16-bit network address of the concentrator device that is the destination of the frame.
- The destination IEEE address sub-field of the frame control field shall be set to 1, and the destination IEEE address field shall be set to the IEEE address of the concentrator device that is the destination of the frame, if this address is known.
- The Source Route sub-field of the frame control field shall be set to 0.

3.4.5.3 NWK Payload

The NWK frame payload contains a command identifier field, a relay count field, and a relay list field. The command frame identifier shall contain the value indicating a route record command frame.

3.4.5.3.1 Relay Count Field

This field contains the number of relays in the relay list field of the route record command. If the route record is being initiated as the result of a NLDE-DATA.request primitive from the next higher layer, the relay count field is initialized to 0. If the route record is being initiated as a result of the relaying of a data frame on behalf of one of the device's end device children, the relay count field is initialized to 1. In either case, it is incremented by each receiving relay.

3.4.5.3.2 Relay List Field

The relay list field is a list of the 16-bit network addresses of the nodes that have relayed the packet. If the route record is being initiated as a result of the relaying of a data frame on behalf of one of the device's end device children, the initiating device will initialize this field with its own 16-bit network address. Receiving relay nodes append their network address to the list before forwarding the packet.

3.4.6 Rejoin Request Command

The rejoin request command allows a device to rejoin its network. This is normally done in response to a communication failure, such as when an end device can no longer communicate with its original parent. The rejoin request command shall be formatted as shown in Figure 3.19.

Figure 3.19 Rejoin Request Command Frame Format

Octets: 1
Capability Information
NWK command payload

3.4.6.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in IEEE 802.15.4.-2003, [B1], the following information shall be provided:

- The destination address and PAN identifier shall be set to the network address and PAN identifier, respectively, of the prospective parent.
- The source MAC address and PAN identifier shall be set to the network address and PAN identifier of the device transmitting the rejoin command frame.
- The transmission options shall be set to require acknowledgement.
- The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

3.4.6.2 NWK Header Fields

The NWK header fields of the rejoin request command frame shall be set as follows:

- The source address field of the NWK header to the 16-bit network address shall be as follows. If the value of the *nwkNetworkAddress* in the NIB is within the valid range, then it shall use that value. If the value of the *nwkNetworkAddress* in the NIB is not within the valid range, then it shall randomly generate a value with the valid range, excluding the value of 0x0000, and use that.
- The source IEEE address sub-field of the frame control field shall be set to 1, and the source IEEE address field shall be set to the IEEE address of the device issuing the request.
- The destination address field in the NWK header shall be set to the 16-bit network address of the prospective parent.
- The destination IEEE address sub-field of the frame control field shall be set to 1, and the destination IEEE address field shall be set to the IEEE address of the prospective parent, if this address is known.
- The radius field shall be set to 1.

3.4.6.3 NWK Payload Fields

The NWK frame payload contains a command identifier field and a capability information field. The command frame identifier shall contain the value indicating a rejoin request command frame.

3.4.6.3.1 Capability Information Field

This one-octet field has the format of the capability information field in the association request command in [B1], which is also described in Table 3.52.

3.4.7 Rejoin Response Command

The rejoin response command is sent by a device to inform a child of its network address and rejoin status. The rejoin request command shall be formatted as shown in Figure 3.20.

Figure 3.20 Rejoin Response Command Frame Format

Octets: 2	**1**
Network address	Rejoin status
NWK command payload	

3.4.7.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in [B1], the following information shall be provided:

- The destination MAC address and PAN identifier shall be set to the network address and PAN identifier, respectively, of the device that sent the rejoin request to which this frame is a response.
- The source MAC address and PAN identifier shall be set to the network address and PAN identifier of the device that received and processed the rejoin request command frame.
- Acknowledgment shall be requested.
- The addressing mode and intra-PAN flags shall be set to support the addressing fields described here. The TXOptions shall request ‘indirect transmission’ to be used if the *Receiver on when idle* bit of the *nwkCapabilityInformation* contained in the corresponding rejoin request command is equal to 0x00. Otherwise, ‘direct transmission’ shall be used.

3.4.7.2 NWK Header Fields

The NWK header fields of the rejoin response command frame shall be set as follows:

- The source address field shall be set to the 16-bit network address of the device that is sending the response.
- The source IEEE address sub-field of the frame control field shall be set to 1 and the source IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the parent device that is sending the response.
- The destination address field of the NWK header shall be set to the current network address of the rejoining device, *i.e.* the device that sent the join request to which this frame is a response.
- The destination IEEE address sub-field of the frame control field shall have a value of 1 and the destination IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the child device that is source of the rejoin request command to which this frame is a response.
- The NWK layer will set the security of the rejoin response command frame to the same level as that of the received rejoin request command frame to which it is a response.

3.4.7.3 NWK Payload Fields

3.4.7.3.1 Network Address Field

If the rejoin was successful, this two-octet field contains the new network address assigned to the rejoining device. If the rejoin was not successful, this field contains the broadcast address (0xffff).

3.4.7.3.2 Rejoin Status Field

This field shall contain one of the non-reserved association status values specified in [B1].

3.4.8 Link Status Command

The link status command frame allows neighboring routers to communicate their incoming link costs to each other as described in section 3.6.3.4. Link status frames are transmitted as one-hop broadcasts without retries.

3.4.8.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in IEEE 802.15.4-2003 [B1], the following information shall be included in the MAC frame header:

- The destination PAN identifier shall be set to the PAN identifier of the device sending the link status command.

- The destination address must be set to the broadcast address of 0xffff.
- The source MAC address and PAN identifier shall be set to the network. address and PAN identifier of the device sending the link status command.
- The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. Because the frame is broadcast, no acknowledgment request shall be specified.
- The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

3.4.8.2 NWK Header Fields

The NWK header field of the link status command frame shall be set as follows:

- The source IEEE address sub-field of the frame control field shall be set to 1 and the source IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the originator of the frame.
- The destination address in the NWK header shall be set to the router-only broadcast address (see Table 3.59).
- The destination IEEE address sub-field of the frame control field shall have a value of 0 and the destination IEEE address field of the NWK header shall not be present.
- The radius field shall be set to 1.

3.4.8.3 NWK Payload Fields

The NWK command payload of the link status command shall be formatted as illustrated in Figure 3.21.

Figure 3.21 Link Status Command Format

Octets: 1	**Variable**
Command options	Link status list
NWK command payload	

3.4.8.3.1 Command Options Field

The format of the 8-bit command options field is shown in Figure 3.22.

Figure 3.22 Link Status Command Options Field

Bit: 0 – 4	**5**	**6**	**7**
Entry count	First frame	Last frame	Reserved

The entry count sub-field of the command options field indicates the number of link status entries present in the link status list. The first frame sub-field is set to 1 if this is the first frame of the sender's link status. The last frame sub-field is set to 1 if this is the last frame of the sender's link status. If the sender's link status fits into a single frame, the first frame and last frame bits shall both be set to 1.

3.4.8.3.2 Link Status List Field

An entry in the link status list is formatted as shown in Figure 3.23.

Figure 3.23 Link Status Entry

Octets: 2	1
Neighbor network address	Link status

Link status entries are sorted in ascending order by network address. If all router neighbors do not fit in a single frame, multiple frames are sent. When sending multiple frames, the last network address in the link status list for frame N is equal to the first network address in the link status list for frame N+1.

Each link status entry contains the network address of a router neighbor, least significant octet first, followed by the link status octet. The incoming cost field contains the device's estimate of the link cost for the neighbor, which is a value between 1 and 7. The outgoing cost field contains the value of the outgoing cost field from the neighbor table.

The link status field in a link status entry is formatted as follows:

Bits: 0-2	3	4-6	7
Incoming cost	Reserved	Outgoing cost	Reserved

3.4.9 Network Report Command

The network report command allows a device to report network events to the device identified by the address contained in the *nwkManagerAddr* in the NIB. Such events are radio channel condition and PAN ID conflicts. The payload of a network report command shall be formatted as illustrated in Figure 3.24.

Figure 3.24 Network Report Command Frame Format

Octets: 1	8	Variable
Command options (see Figure 3.25)	EPID	Report information
NWK command payload		

3.4.9.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in [B1], the following information shall be included in the MAC frame header:

- The destination PAN identifier shall be set to the PAN identifier of the device sending the network report command.
- The destination address shall be set to the value of the next-hop address field in the routing table entry for which the destination address field has the same value as the *nwkManagerAddr* field in the NIB. If no such routing table entry exists, then the NWK may attempt route discovery as described in section 3.6.3.5.

- The source MAC address and PAN identifier shall be set to the network address and PAN identifier of the device sending the network report command, which may or may not be the device from which the command originated.
- The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security. The transmission options shall be set to require acknowledgment.

3.4.9.2 NWK Header Fields

The NWK header fields of the network report command frame shall be set as follows:

- The source IEEE address sub-field of the frame control field shall be set to 1and the source IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the originator of the frame.
- The destination address field in the NWK header shall be set to the 16-bit network address contained in the *nwkManagerAddr* attribute of the NIB.
- The destination IEEE address sub-field of the frame control field shall have a value of 1 and the destination IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the corresponding to the 16-bit network address contained in the *nwkManagerAddr* attribute of the NIB, if this IEEE address is known.

3.4.9.3 NWK Payload Fields

The NWK frame payload contains a command identifier field, a command options field, an EPID field, and a report information payload.

The command frame identifier shall contain the value indicating a network report command frame.

3.4.9.3.1 Command Options Field

The format of the 8-bit command options field is shown in Figure 3.25.

Figure 3.25 Network Report Command Options Field

Bits 0 - 4	5 - 7
Report information count	Report command identifier (see Figure 3.26)

3.4.9.3.1.1 Report Information Count Sub-Field

The report information count sub-field contains an integer indicating the number of records contained within the Report Information field. The size of a record depends in the value of the Report Command Identifier.

3.4.9.3.1.2 Report Command Identifier Sub-Field

The report command identifier sub-field contains an integer indicating the type of report information command. Figure 3.26 contains the values that can be inserted into this field.

Figure 3.26 Report Command Identifier Sub-Field

Report Command Identifier Value	Report Type
0x00	PAN identifier conflict
0x01 - 0x07	Reserved

3.4.9.3.2 EPID Field

The EPID field shall contain the 64-bit EPID that identifies the network that the reporting device is a member of.

3.4.9.3.3 Report Information

The report information field provides the information being reported, the format of this field depends upon the value of the Report Command Identifier sub-field.

3.4.9.3.3.1 PAN Identifier Conflict Report

If the value of the Report Command Identifier sub-field indicates a PAN identifier conflict report then the Report Information field will have the format shown in Figure 3.27.

Figure 3.27 PAN Identifier Conflict Report

Octets: 2	2	2
1st PAN ID	...	nth PAN ID

The PAN ID conflict report shall be made up of a list of 16-bit PAN identifiers that are operating in the neighborhood of the reporting device. The number of PAN identifiers in the PAN ID conflict report shall be equal to the value of the report information count sub-field of the command options field.

3.4.10 Network Update Command

The network update command allows the device identified by the *nwkManagerAddr* attribute of the NIB to broadcast the change of configuration information to all devices in the network. For example, broadcasting the fact that the network is about to change its short PAN identifier.

The payload of a network update command shall be formatted as illustrated in Figure 3.28.

Figure 3.28 Network Update Command Frame Format

Octets: 1	8	1	Variable
Command Options (see Figure 3.25)	EPID	Update Id	Update Information
NWK command payload			

3.4.10.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service specified in [B1], the following information shall be included in the MAC frame header:

- The destination PAN identifier shall be set to the old PAN identifier of the ZigBee coordinator in order for the command frame to reach network devices which have not received this update. The destination address shall be set according to the procedures for broadcast transmission outlined in section 3.6.5.
- The source MAC address and PAN identifier shall be set to the network address and the old PAN identifier of the device sending the network report command, which may or may not be the device from which the command originated.
- The frame control field shall be set to specify that the frame is a MAC data frame with MAC security disabled, since any secured frame originating from the NWK layer shall use NWK layer security.

3.4.10.2 NWK Header Fields

The NWK header fields of the network update command frame shall be set as follows:

- The source IEEE address sub-field of the frame control field shall be set to 1and the source IEEE address field of the NWK header shall be present and shall contain the 64-bit IEEE address of the originator of the frame.
- The destination address in the NWK header shall be set to the broadcast address 0xffff.
- The destination IEEE address sub-field of the frame control field shall have a value of 0 and the destination IEEE address field shall not be present in the NWK header.

3.4.10.3 NWK Payload Fields

The NWK frame payload contains a command identifier field, a command options field, an EPID field and an Update Information variable field.

The command frame identifier shall contain the value indicating a network update command frame.

3.4.10.3.1 Command Options Field

The format of the 8-bit command options field is shown in Figure 3.29.

Figure 3.29 Network Update Command Options Field

Bits 0 - 4	**5 - 7**
Update Information Count	Update Command identifier (see Figure 3.30)

3.4.10.3.1.1 Update Information Count Sub-Field

The update information count sub-field contains an integer indicating the number of records contained within the Update Information field. The size of a record depends on the value of the Update Command Identifier sub-field.

3.4.10.3.1.2 Update Command Identifier Sub-Field

The update command identifier sub-field contains an integer indicating the type of update information command. Figure 3.30 contains the values that can be inserted into this field.

Figure 3.30 Update Command Identifier Sub-Field

Update Command Identifier Value	Report Type
0x00	PAN Identifier Update
0x01 - 0x07	Reserved

3.4.10.3.2 EPID Field

The EPID field shall contain the 64bit EPID that identifies the network that is to be updated.

3.4.10.3.3 Update Id Field

The update Id field will reflect the current value of the *nwkUpdateId* attribute of the device sending the frame.

3.4.10.3.4 Update Information

The update information field provides the information being updated, the format of this field depends upon the value of the Update Command Identifier sub-field.

3.4.10.3.4.1 PAN Identifier Update

If the value of the Update Command Identifier sub-field indicates a PAN identifier update, then the Update Information field shall have the format shown in Figure 3.31.

Figure 3.31 PAN Identifier Update

Octets: 2
New PAN ID

The PAN identifier update shall be made up of a single 16-bit PAN identifier that is the new PAN identifier for this network to use. The Update Information count sub field shall be set equal to 1 as there is only a single PAN identifier contained within the Update Information field.

3.4.11 End Device Timeout Request Command

The End Device Timeout Request command is sent by an end device informing its parent of its timeout requirements. This allows the parent the ability to delete the child entry from the neighbor table if the child has not communicated with the parent in the specified amount of time.

The payload of an End Device Timeout Request command shall be formatted as illustrated in Figure 3.32.

Figure 3.32 Format of the End Device Timeout Request Command

Octets: 1	1
Request Timeout Enumeration	End Device Configuration

3.4.11.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in [B1], the following information shall be provided:

- The destination address and PAN identifier shall be set to the network address and PAN identifier, respectively, of the end device's parent.
- The source MAC address and PAN identifier shall be set to the network address and PAN identifier of the device transmitting the End Device Timeout Request command.
- The transmission options shall be set to require acknowledgement.
- The address mode and intra-PAN flags shall be set to support the addressing fields described here.

3.4.11.2 NWK Header fields

The NWK header fields of the End Device Timeout Request command frame shall be set as follows:

- The source address field of the NWK header shall be set to the 16-bit network address.
- The source IEEE address sub-field of the frame control field shall be set to 1, and the source IEEE address field shall be set to the IEEE address of the device issuing the request.
- The destination address field in the NWK header shall be set to the 16-bit network address of the parent.
- The destination IEEE address sub-field of the frame control field shall be set to 1, and the destination IEEE address field shall be set to the IEEE address of the parent.
- The radius field shall be set to 1.

3.4.11.3 NWK Payload Fields

The NWK frame payload contains a command identifier field and a capability

Table 3.43 Fields of the End Device Timeout Request

Name	Type	Valid Range	Description
Requested Timeout Enumeration	Enumerated type	0 - 14	The requested timeout enumeration. This will be converted into actual timeout value based on Table 3.44
End Device Configuration	Bitmask	0x00 – 0x00	This is an enumeration of the child's requested configuration.

3.4.11.3.1 Requested Timeout Field

The valid values for the requested timeout will be an enumerated type between 0 and 14. This will be converted to an actual timeout value according to Table 3.44 below

Table 3.44 Requested Timeout Enumerated Values

Requested Timeout Enumeration Value	Actual Timeout Value
0	10 seconds
1	2 minutes

2	4 minutes
3	8 minutes
4	16 minutes
5	32 minutes
6	64 minutes
7	128 minutes
8	256 minutes
9	512 minutes
10	1024 minutes
11	2048 minutes
12	4096 minutes
13	8192 minutes
14	16384 minutes

This allows for an actual timeout value between 10 seconds and 16384 minutes (~ 11 days).

3.4.11.3.2 End Device Configuration Field

This is a bitmask indicating the end device's requested configuration. At this time there are no enumerated bits in the configuration field. Devices adhering to this standard shall set the field to 0. To allow for future compatibility this field is left in place. Devices that receive the End Device Timeout Request message with an End Device Configuration field set to anything other than 0 shall reject the message. The receiving device shall send an End Device Timeout Response command with a status of 0x01 (INCORRECT_VALUE).

3.4.12 End Device Timeout Response Command

The End Device Timeout Response is sent by a router parent informing the end device whether it has accepted the timeout value that it was previously sent, and what its capabilities are.

Figure 3.33 Format of the End Device Timeout Response Command

Octets: 1	**1**
Status	Parent Information

3.4.12.1 MAC Data Service Requirements

In order to transmit this command using the MAC data service, specified in reference [B1], the following information shall be provided:

- The destination address and PAN identifier shall be set to the network address and PAN identifier, respectively, of the end device.
- The source MAC address and PAN identifier shall be set to the network address and PAN identifier of the device transmitting the End Device Timeout Response command.
- The transmission options shall be set to require acknowledgement.

- The address mode and intra-PAN flags shall be set to support the addressing fields described here.

3.4.12.2 NWK Header fields

The NWK header fields of the End Device Timeout Response command frame shall be set as follows:

- The source address field of the NWK header shall be set to the 16-bit network address.
- The source IEEE address sub-field of the frame control field shall be set to 1, and the source IEEE address field shall be set to the IEEE address of the device issuing the command.
- The destination address field in the NWK header shall be set to the 16-bit network address of the end device.
- The destination IEEE address sub-field of the frame control field shall be set to 1, and the destination IEEE address field shall be set to the IEEE address of the end device.
- The radius field shall be set to 1.

3.4.12.2.1 NWK Payload Fields

The NWK frame payload contains a command identifier field and a capability information field. The payload of the End Device Timeout Response are described in Table 3.45.

Table 3.45 Payload fields of the End Device Timeout Response

Name	Type	Valid Range	Description
Status	Enumeration	0 – 0xFF	The success or failure result of the previously received End Device Timeout Request command. See Table 3.46 for an enumeration of the status codes.
Parent Information	Bitmask	0 – 0xFF	This bitmask indicates the parent router's support information to the child device. The bitmask's values are described in Table 3.47

Table 3.46 Enumeration of the End Device Timeout Response Status

Status	Value	Description
SUCCESS	0x00	The End Device Timeout Request message was accepted by the parent.
INCORRECT_VALUE	0x01	The received timeout value in the End Device Timeout Request command was outside the allowed range.
Reserved	0x02 – 0xFF	Reserved for Future Use

Table 3.47 Values of the Parent Information Bitmask

Bits	Description
0	MAC Data Poll Keepalive Supported
1	End Device Timeout Request Keepalive Supported
2 – 15	Reserved for future use

3.5 Constants and NIB Attributes

3.5.1 NWK Constants

The constants that define the characteristics of the NWK layer are presented in Table 3.48.

Table 3.48 NWK Layer Constants

Constant	Description	Value
nwkcCoordinatorCapable	A Boolean flag indicating whether the device is capable of becoming the ZigBee coordinator. A value of 0x00 indicates that the device is not capable of becoming a coordinator while a value of 0x01 indicates that the device is capable of becoming a coordinator.	Configuration dependent
nwkcDefaultSecurityLevel	The default security level to be used (see Chapter 4).	Defined in stack profile
nwkcMinHeaderOverhead	The minimum number of octets added by the NWK layer to an NSDU.	0x08
nwkcProtocolVersion	The version of the ZigBee NWK protocol in the device.	0x02
nwkcWaitBeforeValidation	The number of OctetDurations, on the originator of a multicast route request, between receiving a route reply and sending a message to validate the route.	0x9c40 (0x500 msec on 2.4 GHz)
nwkcRouteDiscoveryTime	The number of OctetDurations until a route discovery expires.	0x4c4b4 (0x2710 msec on 2.4GHz)
nwkcMaxBroadcastJitter	The maximum broadcast jitter time measured in OctetDurations.	0x7d0 (0x40 msec on 2.4GHz)

Constant	Description	Value
nwkcInitialRREQRetries	The number of times the first broadcast transmission of a route request command frame is retried.	0x03
nwkcRREQRetries	The number of times the broadcast transmission of a route request command frame is retried on relay by an intermediate ZigBee router or ZigBee coordinator.	0x02
nwkcRREQRetryInterval	The number of OctetDurations between retries of a broadcast route request command frame.	0x1f02 (0xfe msec on 2.4Ghz)
nwkcMinRREQJitter	The minimum jitter, in OctetDurations, for broadcast retransmission of a route request command frame.	0x3f (2 msec on 2.4GHz)
nwkcMaxRREQJitter	The maximum jitter, in OctetDurations, for broadcast retransmission of a route request command frame.	0xfa0 (128 msec on 2.4GHz)
nwkcMACFrameOverhead	The size of the MAC header used by the ZigBee NWK layer.	0x0b

3.5.2 NWK Information Base

The NWK information base (NIB) comprises the attributes required to manage the NWK layer of a device. Each of these attributes can be read or written using the NLME-GET.request and NLME-SET.request primitives, respectively, except for attributes for which the Read Only column contains a value of Yes. In that case, the attributes value may be read using the NLME-GET.request primitive but may not be set using the NLME-SET.request primitive. Generally, these read-only attribute are set using some other mechanism. For example, the *nwkSequenceNumber* attribute is set as specified in section 3.6.2.1 and incremented every time the NWK layer sends a frame. The attributes of the NIB are presented in Table 3.49.

Table 3.49 NIB Attributes

Attribute	Id	Type	Read Only	Range	Description	Default
nwkSequenceNumber	0x81	Integer	Yes	0x00 – 0xff	A sequence number used to identify outgoing frames (see section 3.6.2).	Random value from within the range
nwkPassiveAckTimeout	0x82	Integer	No	0x000000 – 0xffffff	The maximum time duration in OctetDurations allowed for the parent and all child devices to retransmit a broadcast message (passive ac-	Defined in stack profile

Attribute	Id	Type	Read Only	Range	Description	Default
					knowledgment time-out).	
nwkMaxBroadcastRetries	0x83	Integer	No	0x00 – 0x5	The maximum number of retries allowed after a broadcast transmission failure.	0x03
nwkMaxChildren	0x84	Integer	No	0x00 – 0xff	The number of children a device is allowed to have on its current network Note that when *nwkAddrAlloc* has a value of 0x02, indicating stochastic addressing, the value of this attribute is implementation-dependent.	Defined in the stack profile
nwkMaxDepth	0x85	Integer	Yes	0x00 – 0xff	The depth a device can have.	Defined in stack profile
nwkMaxRouters	0x86	Integer	No	0x01-0xff	The number of routers any one device is allowed to have as children. This value is determined by the ZigBee coordinator for all devices in the network. If *nwkAddrAlloc* is 0x02 this value not used.	Defined in stack profile
nwkNeighborTable	0x87	Set	No	Variable	The current set of neighbor table entries in the device (see Table 3.53).	Null set

Attribute	Id	Type	Read Only	Range	Description	Default
nwkNetworkBroadcastDeliveryTime	0x88	Integer	No	0 – 0xffffffff	Time duration in OctetDurations that a broadcast message needs to encompass the entire network. This is a calculated quantity based on other NIB attributes. The formula is given in section 3.5.2.1.	Defined in stack profile
nwkReportConstantCost	0x89	Integer	No	0x00-0x01	If this is set to 0, the NWK layer shall calculate link cost from all neighbor nodes using the LQI values reported by the MAC layer; otherwise, it shall report a constant value.	0x00
Reserved	0x8a					
nwkRouteTable	0x8b	Set	No	Variable	The current set of routing table entries in the device (see Table 3.56).	Null set
nwkSymLink	0x8e	Boolean	No	TRUE or FALSE	The current route symmetry setting: TRUE means that routes are considered to be comprised of symmetric links. Backward and forward routes are created during one-route discovery and they are identical. FALSE indicates that routes are not consider to be comprised of symmetric links. Only the forward route is stored during route discovery.	FALSE
nwkCapabilityInformation	0x8f	Bit vector	Yes	See Table 3.52.	This field shall contain the device capability information established at network joining time.	0x00

Attribute	Id	Type	Read Only	Range	Description	Default
nwkAddrAlloc	0x90	Integer	No	0x00 - 0x02	A value that determines the method used to assign addresses: 0x00 = use distributed address allocation 0x01 = reserved 0x02 = use stochastic address allocation	0x00
nwkUseTreeRouting	0x91	Boolean	No	TRUE or FALSE	A flag that determines whether the NWK layer should assume the ability to use hierarchical routing: TRUE = assume the ability to use hierarchical routing. FALSE = never use hierarchical routing.	TRUE
nwkManagerAddr	0x92	Integer	No	0x0000 - 0xfff7	The address of the designated network channel manager function.	0x0000
nwkMaxSourceRoute	0x93	Integer	No	0x00 - 0xff	The maximum number of hops in a source route.	0x0c
nwkUpdateId	0x94	Integer	No	0x00 - 0xFF	The value identifying a snapshot of the network settings with which this node is operating with.	0x00
nwkTransactionPersistenceTime	0x95	Integer	No	0x0000 - 0xffff	The maximum time (in superframe periods) that a transaction is stored by a coordinator and indicated in its beacon. This attribute reflects the value of the MAC PIB attribute *macTransactionPersistenceTime* (see [B1]) and any changes made by the higher layer will be reflected in the MAC PIB attribute value as well.	0x01f4

Attribute	Id	Type	Read Only	Range	Description	Default
nwkNetworkAddress	0x96	Integer	No	0x0000 - 0xfff7	The 16-bit address that the device uses to communicate with the PAN. This attribute reflects the value of the MAC PIB attribute *macShortAddress* (see [B1]) and any changes made by the higher layer will be reflected in the MAC PIB attribute value as well.	0xffff
nwkStackProfile	0x97	Integer	No	0x00-0x0f	The identifier of the ZigBee stack profile in use for this device.	
nwkBroadcastTransactionTable	0x98	Set	Yes	-	The current set of broadcast transaction table entries in the device (see Table 3.60).	Null set
nwkGroupIDTable	0x99	Set	No	Variable	The set of group identifiers, in the range 0x0000 - 0xffff, for groups of which this device is a member.	Null Set
nwkExtendedPANID	0x9a	64-bit extended address	No	0x0000000000000000 - 0xfffffffffffffffe	The Extended PAN Identifier for the PAN of which the device is a member. The value 0x0000000000000000 means the Extended PAN Identifier is unknown.	0x0000000000000000
nwkUseMulticast	0x9b	Boolean	No	TRUE or FALSE	A flag determining the layer where multicast messaging occurs. TRUE = multicast occurs at the network layer. FALSE= multicast occurs at the APS layer and using the APS header.	TRUE
nwkRouteRecordTable	0x9c	Set	No	Variable	The route record table (see Table 3.50).	Null Set

Attribute	Id	Type	Read Only	Range	Description	Default
nwkIsConcentrator	0x9d	Boolean	No	TRUE or FALSE	A flag determining if this device is a concentrator. TRUE = Device is a concentrator. FALSE = Device is not a concentrator.	FALSE
nwkConcentratorRadius	0x9e	Integer	No	0x00 - 0xff	The hop count radius for concentrator route discoveries.	0x0000
nwkConcentratorDiscoveryTime	0x9f	Integer	No	0x00 - 0xff	The time in seconds between concentrator route discoveries. If set to 0x0000, the discoveries are done at start up and by the next higher layer only.	0x0000
nwkSecurityLevel	0xa0		No		Security attribute defined in Chapter 4.	
nwkSecurityMaterialSet	0xa1		No		Security attribute defined in Chapter 4.	
nwkActiveKeySeqNumber	0xa2		No		Security attribute defined in Chapter 4.	
nwkAllFresh	0xa3		No		Security attribute defined in Chapter 4.	
nwkLinkStatusPeriod	0xa6	Integer	No	0x00 - 0xff	The time in seconds between link status command frames.	0x0f
nwkRouterAgeLimit	0xa7	Integer	No	0x00 - 0xff	The number of missed link status command frames before resetting the link costs to zero.	3

Attribute	Id	Type	Read Only	Range	Description	Default
nwkUniqueAddr	0xa8	Boolean	No	TRUE or FALSE	A flag that determines whether the NWK layer should detect and correct conflicting addresses: TRUE = assume addresses are unique. FALSE = addresses may not be unique.	TRUE
nwkAddressMap	0xa9	Set	No	Variable	The current set of 64-bit IEEE to 16-bit network address map (see Table 3.51).	Null Set
nwkTimeStamp	0x8C	Boolean	No	TRUE or FALSE	A flag that determines if a time stamp indication is provided on incoming and outgoing packets. TRUE= time indication provided. FALSE = no time indication provided.	FALSE
nwkPANId	0x80	16-bit PAN ID	No	0x0000 - 0xffff	This NIB attribute should, at all times, have the same value as *macPANId*.	0xffff
nwkTxTotal	0x8D	Integer	No	0x0000 - 0xffff	A count of unicast transmissions made by the NWK layer on this device. Each time the NWK layer transmits a unicast frame, by invoking the MCPS-DATA.request primitive of the MAC sub-layer, it shall increment this counter. When either the NHL performs an NLME-SET.request on this attribute or if the value of *nwkTxTotal* rolls over past 0xffff the NWK layer shall reset to 0x00 each Transmit Failure field contained in the neighbor table.	0

Attribute	Id	Type	Read Only	Range	Description	Default
nwkLeaveRequestAllowed	0xAA	Boolean	No	TRUE or FALSE	This policy determines whether or not a remote NWK leave request command frame received by the local device is accepted.	TRUE
nwkParentInformation	0xAB	Bitmask	No	0x00 – 0xFF	The behavior depends upon whether the device is an FFD or RFD. For an RFD, this records the information received in an End Device Timeout Response command indicating the parent information. The bitmask values are defined in Table 3.47. For an FFD, this records the device's local capabilities.	0x00
nwkEndDeviceTimeoutDefault	0xAC	Integer	No	0x00 – 0xFF	This is an index into tableTable 3.44. It indicates the default timeout in minutes for any end device that does not negotiate a different timeout value.	8
nwkLeaveRequestWithoutRejoinAllowed	0xAD	Boolean	No	TRUE or FALSE	This policy determines whether a NWK leave request is accepted when the Rejoin bit in the message is set to FALSE	TRUE
nwkIeeeAddress	0xAE	64-bit address	Yes	0x00000000 00000001 – 0xFFFFFFFF FFFFFFFF	The IEEE address of the local device.	

7743

Table 3.50 Route Record Table Entry Format

Field Name	**Field Type**	**Valid Range**	**Reference**
Network Address	Integer	0x0000- 0xfff7	The destination network address for this route record.
Relay Count	Integer	0x0000 - 0xffff	The count of relay nodes from concentrator to the destination.
Path	Set of Network Addresses		The set of network addresses that represent the route in order from the concentrator to the destination.

Table 3.51 Network Address Map

64-bit IEEE Address	**16-bit Network address**
A valid 64-bit IEEE Address or Null if not known	0x0000 - 0xfff7

3.5.2.1 Broadcast Delivery Time

The total delivery time for a broadcast transmission, *i.e.* the time required for a broadcast to be delivered to every device in the network, shall be calculated according to the following formula:

```
nwkBroadcastDeliveryTime =    2*nwkMaxDepth*
                              ((0.05+(nwkcMaxBroadcastJitter/2))+
                              nwkPassiveAckTimeout*nwkBroadcastRetries/
                              1000)
```

3.6 Functional Description

3.6.1 Network and Device Maintenance

All ZigBee devices shall provide the following functionality:

- Join a network
- Leave a network
- Rejoin a network

Both ZigBee coordinators and routers shall provide the following additional functionality:

- Permit devices to join the network using the following:
  - Association indications from the MAC

    - Explicit join requests from the application
    - Rejoin requests
- Permit devices to leave the network using the following:
    - Network leave command frames
    - Explicit leave requests from the application
- Participate in assignment of logical network addresses
- Maintain a list of neighboring devices

ZigBee coordinators shall provide functionality to establish a new network. ZigBee routers and end devices shall provide the support of portability within a network.

3.6.1.1 Establishing a New Network

The procedure to establish a new network is initiated through use of the NLME-NETWORK-FORMATION.request primitive. Only devices for which the *nwkcCoordinatorCapable* constant has a value of 0x01, and which are not currently joined to a network shall attempt to establish a new network. If this procedure is initiated on any other device, the NLME shall terminate the procedure and notify the next higher layer of the illegal request. This is achieved by issuing the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to INVALID_REQUEST.

When this procedure is initiated, the NLME shall first request that the MAC sub-layer perform an energy detection scan over either a specified set of channels or, by default, the complete set of available channels, as dictated by the PHY layer (see [B1]), to search for possible interferers. A channel scan is initiated by issuing the MLME-SCAN.request primitive to the MAC sub-layer with the ScanType parameter set to energy detection scan. The results are communicated back via the MLME-SCAN.confirm primitive. This scan is not necessary if there is only one channel specified.

On receipt of the results from a successful energy detection scan, the NLME shall order the channels according to increasing energy measurement and discard those channels whose energy levels are beyond an acceptable level. The choice of an acceptable energy level is left to the implementation. The NLME shall then perform an active scan, by issuing the MLME-SCAN.request primitive with the ScanType parameter set to active scan and ChannelList set to the list of acceptable channels and ChannelPage set to zero, to search for other ZigBee devices. To determine the best channel on which to establish a new network, the NLME shall review the list of returned PAN descriptors and find the first channel with the lowest number of existing networks, favoring a channel with no detected networks.

If no suitable channel is found, the NLME shall terminate the procedure and notify the next higher layer of the startup failure. This is achieved by issuing the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to STARTUP_FAILURE.

If a suitable channel is found, the NLME shall select a PAN identifier for the new network. To do this the device shall choose a random PAN identifier less than 0xffff that is not already in use on the selected channel. Once the NLME makes its choice, it shall set the *macPANID* attribute in the MAC sub-layer to this value by issuing the MLME-SET.request primitive.

If no unique PAN identifier can be chosen, the NLME shall terminate the procedure and notify the next higher layer of the startup failure by issuing the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to STARTUP_FAILURE.

Once a PAN identifier is selected, the NLME shall select a 16-bit network address equal to 0x0000 and set the *nwkNetworkAddress* attribute of the NIB equal to the selected network address.

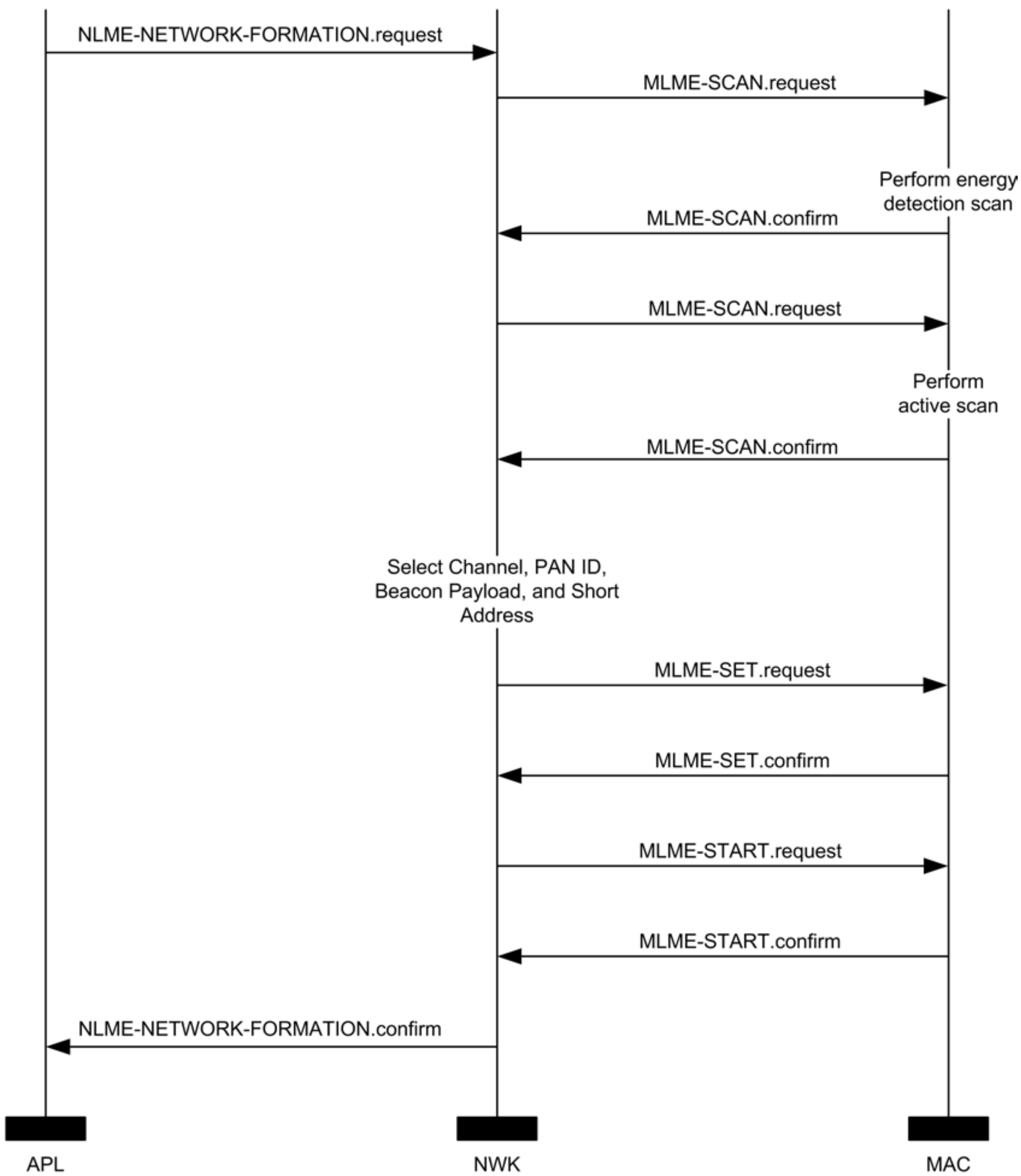
Once a network address is selected, the NLME shall check the value of the *nwkExtendedPANId* attribute of the NIB. If this value is 0x0000000000000000 this attribute is initialized with the value of the MAC constant *aExtendedAddress*.

Once the value of the *nwkExtendedPANId* is checked, the NLME shall begin operation of the new PAN by issuing the MLME-START.request primitive to the MAC sub-layer. The parameters of the MLME-START.request primitive shall be set according to those passed in the NLME-NETWORK-FORMATION.request, the results of the channel scan, and the chosen PAN identifier. The status of the PAN startup is communicated back via the MLME-START.confirm primitive.

On receipt of the status of the PAN startup, the NLME shall inform the next higher layer of the status of its request to initialize the ZigBee coordinator. This is achieved by issuing the NLME-NETWORK-FORMATION.confirm primitive with the Status parameter set to the primitive returned in the MLME-START.confirm from the MAC sub-layer.

The procedure to successfully start a new network is illustrated in the message sequence chart (MSC) shown in Figure 3.34.

Figure 3.34 Establishing a New Network



3.6.1.2 Permitting Devices to Join a Network

The procedure for permitting devices to join a network is initiated through the NLME-PERMIT-JOINING.request primitive. Only devices that are either the ZigBee coordinator or a ZigBee router shall attempt to permit devices to join the network.

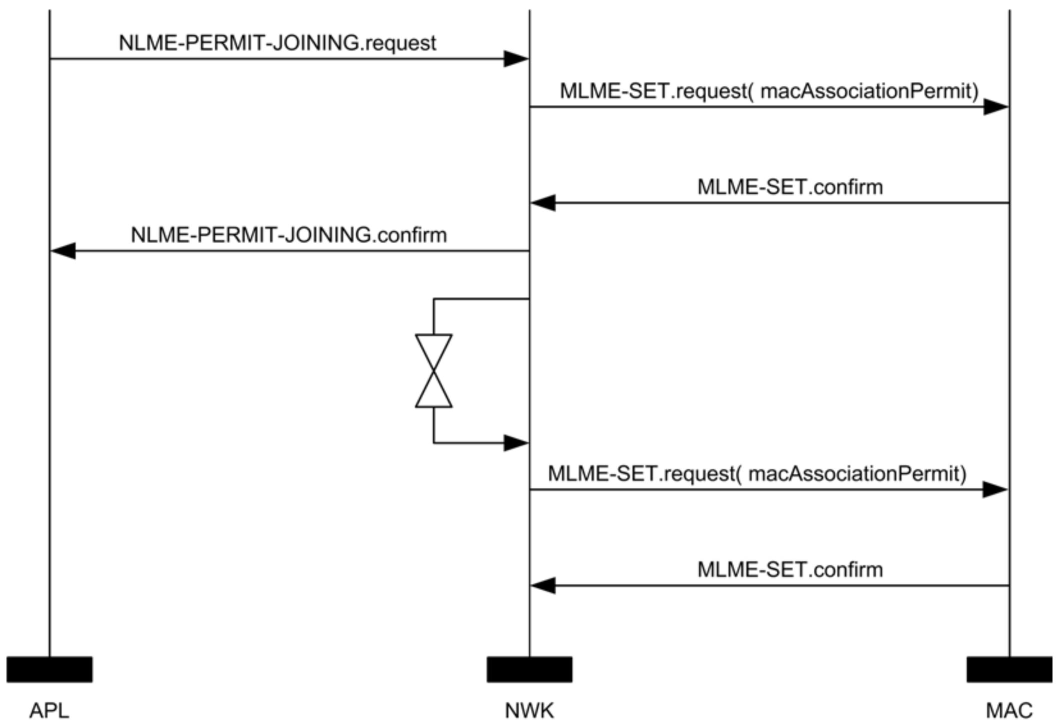
When this procedure is initiated with the PermitDuration parameter set to 0x00, the NLME shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to FALSE. A MAC sub-layer attribute setting is initiated by issuing the MLME-SET.request primitive.

When this procedure is initiated with the PermitDuration parameter set to a value between 0x01 and 0xfe, the NLME shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to TRUE. The NLME shall then start a timer to expire after the specified duration. On expiration of this timer, the NLME shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to FALSE.

When this procedure is initiated with the PermitDuration parameter set to 0xff, the NLME shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to TRUE for an unlimited amount of time, unless another NLME-PERMIT-JOINING.request primitive is issued.

The procedure for permitting devices to join a network is illustrated in the MSC shown in Figure 3.35.

Figure 3.35 Permitting Devices to Join a Network



3.6.1.3 Network Discovery

The NWK layer enables higher layers to discover what networks, if any, are operational in the POS of a device.

The procedure for network discovery shall be initiated by issuing the NLME-NETWORK-DISCOVERY.request primitive with the ScanChannels parameter set to indicate which channels are to be scanned for networks and the ScanDuration parameter set to indicate the length of time to be spent scanning each channel. Upon receipt of this primitive, the NWK layer shall issue an MLME-SCAN.request primitive asking the MAC sub-layer to perform an active scan.

Every beacon frame received during the scan having a non-zero length payload shall cause the MLME-BEACON-NOTIFY.indication primitive to be issued from the MAC sub-layer of the scanning device to its NLME. This primitive includes information such as the addressing information of the beaconing device, whether or not it is permitting association and the beacon payload. (See [B1] for the complete list of parameters). The NLME of the scanning device shall check the protocol ID field in the beacon payload and verify that it matches the ZigBee protocol identifier. If not, the beacon is ignored. Otherwise, the device shall copy the relevant information from each received beacon (see Figure 3.51 for the structure of the beacon payload) into its neighbor table (see Table 3.53 for the contents of a neighbor table entry).

Once the MAC sub-layer signals the completion of the scan by issuing the MLME-SCAN.confirm primitive to the NLME, the NWK layer shall issue the NLME-NETWORK-DISCOVERY.confirm primitive containing a description of each network that was heard. Every network description contains the ZigBee version, stack profile, Extended PAN Id, PAN Id, logical channel, and information on whether it is permitting joining (see Table 3.8).

3.6.1.4 Joining a Network

For purposes of the ensuing discussion, a parent-child relationship is formed when a device having membership in the network allows a new device to join. On joining, the new device becomes the child, while the first device becomes the parent.

3.6.1.4.1 Joining a Network Through Association

This section specifies the procedure a device (child) shall follow if it opts to join a network using the underlying association capabilities provided by the MAC, as well as the procedure a ZigBee coordinator or router (parent) shall follow upon receipt of an MLME-ASSOCIATE.request primitive from the MAC.

3.6.1.4.1.1 Child Procedure

The procedure for joining a network using the MAC layer association procedure should be preceded by network discovery as described in section 3.6.1.3. Upon receipt of the NLME-NETWORK-DISCOVERY.confirm primitive, the next higher layer shall either choose a network to join from the discovered networks or redo the network discovery. Once a network is selected, it shall then issue the NLME-JOIN.request with the RejoinNetwork parameter set to 0x00 and the JoinAsRouter parameter set to indicate whether the device wants to join as a routing device.

Only those devices that are not already joined to a network shall initiate the join procedure. If any other device initiates this procedure, the NLME shall terminate the procedure and notify the next higher layer of the illegal request by issuing the NLME-JOIN.confirm primitive with the Status parameter set to INVALID_REQUEST.

For a device that is not already joined to a network, the NLME-JOIN.request primitive shall cause the NWK layer to search its neighbor table for a suitable parent device, *i.e.* a device for which following conditions are true:

- The device belongs to the network identified by the ExtendedPANId parameter.
- The device is open to join requests and is advertising capacity of the correct device type.
- The link quality for frames received from this device is such that a link cost of at most 3 is produced when calculated as described in section 3.6.3.1.
- If the neighbor table entry contains a potential parent field for this device, that field shall have a value of 1 indicating that the device is a potential parent.
- The device shall have the most recent update id, where the determination of most recent needs to take into account that the update id will wrap back to zero. In particular the update id given in the beacon payload of the device should be greater than or equal to — again, compensating for wrap — the *nwkUpdateId* attribute of the NIB.

If the neighbor table contains no devices that are suitable parents, the NLME shall respond with an NLME-JOIN.confirm with a Status parameter of NOT_PERMITTED. If the neighbor table has more than one device that could be a suitable parent, the device which is at a minimum depth from the ZigBee coordinator may be chosen if and only if the *nwkStackProfile* is set to 1. If more than one device has a minimum depth, the NWK layer is free to choose from among them. If *nwkStackProfile* is not equal to 1, then the depth shall not be considered when choosing a suitable parent.

Once a suitable parent is identified the device shall set its *nwkParentInformation* value in the NIB to 0, then the NLME shall issue an MLME-ASSOCIATE.request primitive to the MAC sub-layer. The LogicalChannel parameter of the MLME-ASSOCIATE.request primitive shall be set to that found in the neighbor table entry corresponding to the coordinator address of the potential parent. The bit-fields of the CapabilityInformation parameter shall have the values shown in Table 3.52 and the capability information shall be stored as the value of the *nwkCapabilityInformation* NIB attribute (see Table 3.49). If more than one device meets these requirements, then the joining device may select the parent with the smallest network depth.

Table 3.52 Capability Information Bit-Fields

Bit	Name	Description
0	Alternate PAN coordinator	This field will always have a value of 0 in implementations of this specification.
1	Device type	This field will have a value of 1 if the joining device is a ZigBee router. It will have a value of 0 if the device is a ZigBee end device or else a router-capable device that is joining as an end device.
2	Power source	This field will be set to the value of lowest-order bit of the PowerSource parameter passed to the NLME-JOIN-request primitive. The values are: 0x01 = Mains-powered device 0x00 = other power source
3	Receiver on when idle	This field will be set to the value of the lowest-order bit of the RxOnWhenIdle parameter passed to the NLME-JOIN.request primitive. 0x01 = The receiver is enabled when the device is idle 0x00 = The receiver may be disabled when the device is idle
4 – 5	Reserved	This field will always have a value of 0 in implementations of this specification.
6	Security capability	This field shall have a value of 0. Note that this overrides the default meaning specified in [B1].

Bit	Name	Description
7	Allocate address	This field will have a value of 1 in implementations of this specification, indicating that the joining device must be issued a 16-bit network address, except in the case where a device has self-selected its address while using the NWK rejoin command to join a network for the first time in a secure manner. In this case, it shall have a value of 0.

Otherwise, the NLME issues the NLME-JOIN.confirm with the Status parameter set to the Status parameter value returned from the MLME-ASSOCIATE.confirm primitive.

If the RejoinNetwork parameter is 0x00 and the JoinAsRouter parameter is set to TRUE, the device will function as a ZigBee router in the network. If the JoinAsRouter parameter is FALSE, then it will join as an end device and not participate in routing.

The addressing parameters in the MLME-ASSOCIATE.request primitive (see Chapter 2) shall be set to contain the addressing information for the device chosen from the neighbor table. The status of the association is communicated back to the NLME via the MLME-ASSOCIATE.confirm primitive.

If the attempt to join was unsuccessful, the NWK layer shall receive an MLME-ASSOCIATE.confirm primitive from the MAC sub-layer with the Status parameter indicating the error. If the Status parameter indicates a refusal to permit joining on the part of the neighboring device (that is, PAN at capacity or PAN access denied), then the device attempting to join should set the Potential parent bit to 0 in the corresponding neighbor table entry to indicate a failed join attempt. Setting the Potential parent bit to 0 ensures that the NWK layer shall not issue another request to associate to the same neighboring device. The Potential parent bit should be set to 1 for every entry in the neighbor table each time an MLME-SCAN.request primitive is issued.

A join request may also be unsuccessful, if the potential parent is not allowing new routers to associate (for example, the maximum number of routers, *nwkMaxRouters* may already have associated with the device) and the joining device has set the JoinAsRouter parameter to TRUE. In this case, the NLME-JOIN.confirm primitive will indicate a status of NOT_PERMITTED. In this case, the child device’s application may wish to attempt to join again as an end device instead, by issuing another NLME-JOIN.request with the JoinAsRouter parameter set to FALSE.

If the attempt to join was unsuccessful, the NLME shall attempt to find another suitable parent from the neighbor table. If no such device could be found, the NLME shall issue the NLME-JOIN.confirm primitive with the Status parameter set to the value returned in the MLME-ASSOCIATE.confirm primitive.

If the attempt to join was unsuccessful and there is a second neighboring device that could be a suitable parent, the NWK layer shall initiate the MAC sub-layer association procedure with the second device. The NWK layer shall repeat this procedure until it either joins the PAN successfully or exhausts its options to join the PAN.

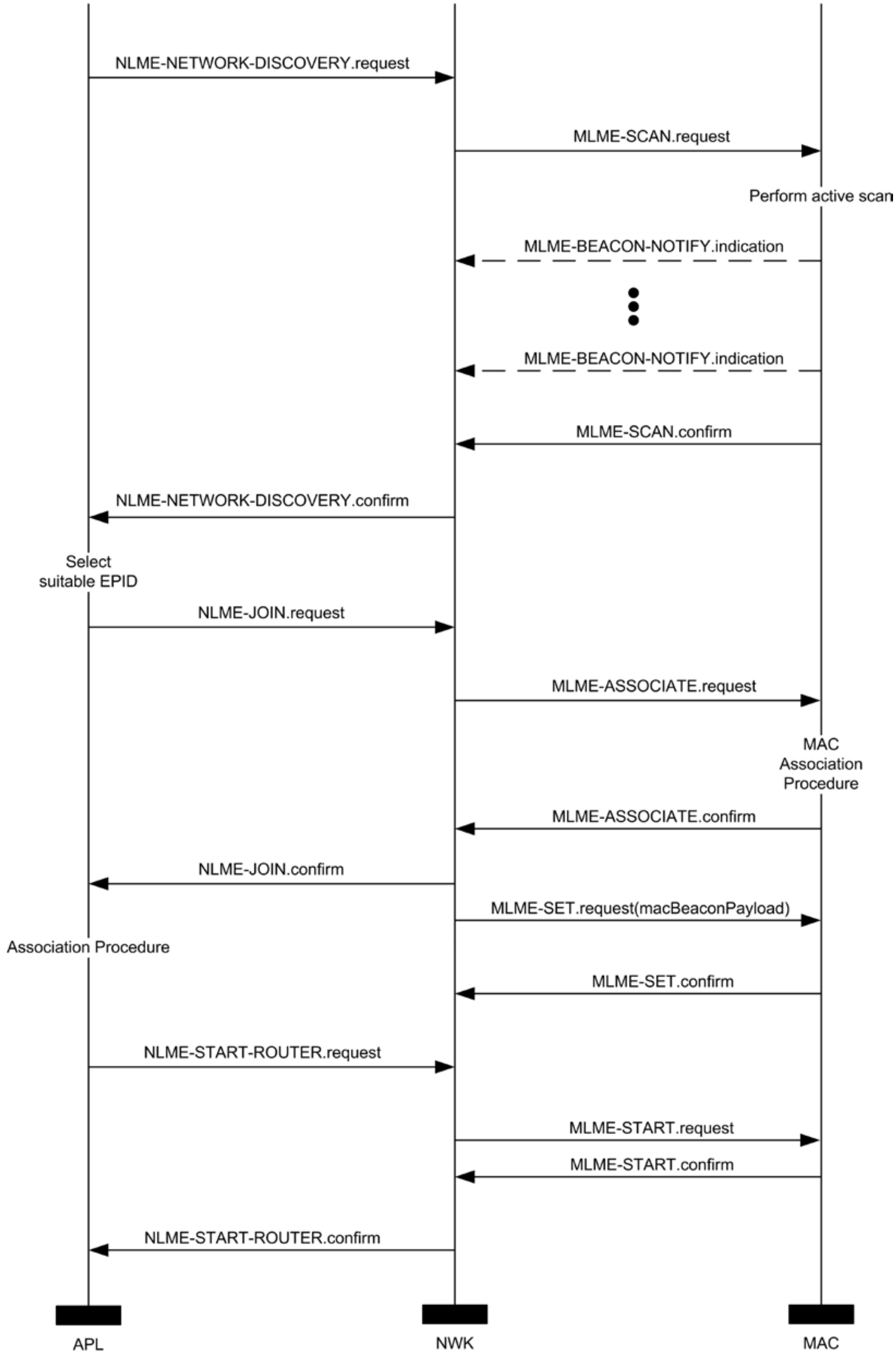
If the device cannot successfully join the PAN specified by the next higher layer, the NLME shall terminate the procedure by issuing the NLME-JOIN.confirm primitive with the Status parameter set to the value returned in the last received MLME-ASSOCIATE.confirm primitive. In this case, the device shall not receive a valid logical address and shall not be permitted to transmit on the network.

If the attempt to join was successful, the NWK shall issue the NLME-JOIN.confirm primitive with a status value of SUCCESS. In this case, the MLME-ASSOCIATE.confirm primitive received by the NWK layer shall contain a 16-bit logical address unique to that network which the child can use in future transmissions. The NWK layer shall then set the Relationship field in the corresponding neighbor table entry to indicate that the neighbor is its parent. By this time, the parent shall have added the new device to its neighbor table. Furthermore, the NWK layer will update the values of *nwkNetworkAddress*, *nwkUpdateId* and *mwkPANId* in the NIB.

If the device is attempting to join a secure network and it is a router, it will need to wait until its parent has authenticated it before transmitting beacons. The device shall therefore wait for an NLME-START-ROUTER.request primitive to be issued from the next higher layer. Upon receipt of this primitive, the NLME shall issue an MLME-START.request primitive if it is a router. If the NLME-START-ROUTER.request primitive is issued on an end device, the NWK layer shall issue an NLME-START-ROUTER.confirm primitive with the status value set to INVALID_REQUEST.

Once the device has successfully joined the network, if it is a router and the next higher layer has issued a NLME-START-ROUTER.request, the NWK layer shall issue the MLME-START.request primitive to its MAC sub-layer. The PANId, LogicalChannel, BeaconOrder and SuperframeOrder parameters shall be set equal to the corresponding values held in the neighbor table entry for its parent. The network depth is set to one more than the parent network depth unless the parent network depth has a value of 0x0f, *i.e.* the maximum value for the 4-bit device depth field in the beacon payload. In this case, the network depth shall also be set to 0x0f. The PANCoordinator and CoordRealignment parameters shall both be set to FALSE. Upon receipt of the MLME-START.confirm primitive, the NWK layer shall issue an NLME-START-ROUTER.confirm primitive with the same status value.

7962 **Figure 3.36 Procedure for Joining a Network Through Association**



7963

3.6.1.4.1.2 Parent Procedure

The procedure for a ZigBee coordinator or router to join a device to its network using the MAC sub-layer association procedure is initiated by the MLME-ASSOCIATE.indication primitive arriving from the MAC sub-layer. Only those devices that are either a ZigBee coordinator or a ZigBee router and that are permitting devices to join the network shall initiate this procedure. If this procedure is initiated on any other device, the NLME shall terminate the procedure.

When this procedure is initiated, the NLME of a potential parent shall first determine whether the device wishing to join already exists on its network. To do this, the NLME shall search its neighbor table in order to determine whether a matching 64-bit, extended address can be found. If an extended address match is found, the NLME shall check that the supplied DeviceCapabilities match the device type on record in the neighbor table. If the device type also matches the NLME, it shall then obtain the corresponding 16-bit network address and issue an association response to the MAC sub-layer. If a device type match is not found the NLME shall remove all records of the device in its neighbor table and restart processing of the MLME-ASSOCIATION.indication. If an extended address match is not found, the NLME shall, if possible, allocate a 16-bit network address for the new device. See section 3.6.1.6 and section 3.6.1.7 for an explanation of the address assignment mechanisms.

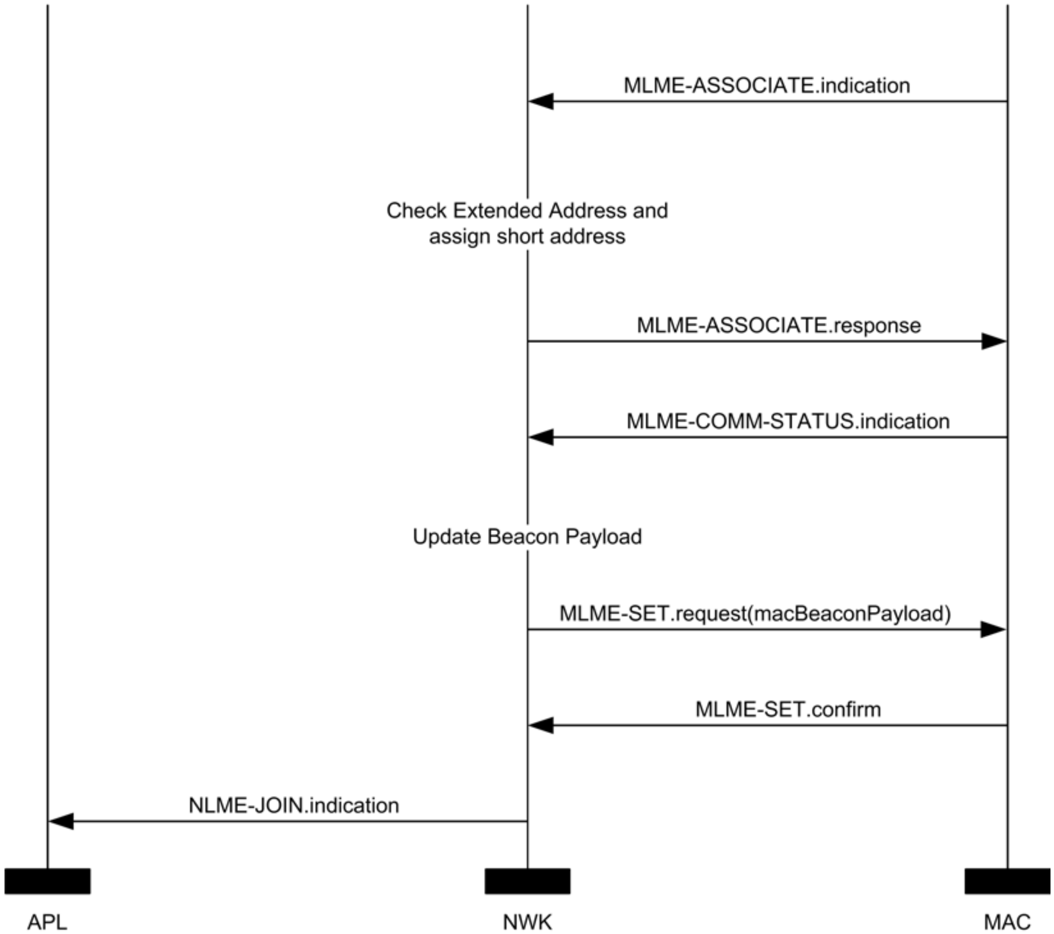
If the potential parent does not have the capacity to accept more children, the NLME shall terminate the procedure and indicate this fact in the subsequent MLME-ASSOCIATE.response primitive to the MAC sub-layer. The Status parameter of this primitive shall indicate that the PAN is at capacity.

If the request to join is granted, the NLME of the parent shall create a new entry for the child in its neighbor table using the supplied device information and indicate a successful association in the subsequent MLME-ASSOCIATE.response primitive to the MAC sub-layer. If the value of *nwkSecurityLevel* is 0x00, the relationship field of the new neighbor table entry shall be set to the value 0x01 indicating that the neighbor is a child; otherwise, it shall be set to 0x05 indicating an unauthenticated child. The status of the response transmission to the child is communicated back to the network layer via the MLME-COMM-STATUS.indication primitive.

If the transmission was unsuccessful (*i.e.* the MLME-COMM-STATUS.indication primitive contained a Status parameter not equal to SUCCESS), the NLME shall terminate the procedure. If the transmission was successful, the NLME shall notify the next higher layer that a child has just joined the network by issuing the NLME-JOIN.indication primitive.

The procedure for successfully joining a device to the network is illustrated in the MSC shown in Figure 3.37.

Figure 3.37 Procedure for Handling a Join Request



3.6.1.4.2 Joining or Rejoining a Network Using NWK Rejoin

Devices that have lost all connection to the network, for example a ZED that can no longer communicate successfully with its parent, can rejoin the network using the NWK rejoin request and NWK rejoin response commands. The rejoining procedure is identical to the association procedure described in the previous section, except that the MAC association procedure is replaced by an exchange involving the rejoin request and rejoin response commands, and, because NWK commands make use of NWK security, no authentication step is performed. Using these commands instead of the MAC procedure allows a device to rejoin a network that does not currently allow new devices to join.

Devices that are joining a network for the first time may also use a variant of this procedure as described in the following sections.

3.6.1.4.2.1 Child Procedure

The procedure for joining or rejoining a network using the NWK rejoin procedure shall be initiated by issuing the NLME-JOIN.request primitive, as shown in Figure 3.38, with the RejoinNetwork parameter set to 0x02 and the ExtendedPANId parameter set to the ExtendedPANId of the network to rejoin. The device type field of the CapabilityInformation parameter shall be set to 1 if the device is intended to join as a router and to 0 otherwise. If the value of the *nwkNetworkAddress* value in the NIB is within the valid range defined for that value, it shall use *nwkNetworkAddress* when issuing the Rejoin Request command. If the *nwkNetworkAddress* is NOT within the valid range, it shall randomly generate a short address within the valid range, excluding the value of 0x0000, and use that for the Rejoin Request command.

The ScanChannels parameter shall be set to indicate which channels are to be scanned to locate this network and the ScanDuration parameter set to indicate the length of time to be spent scanning each channel.

Upon receipt of this primitive, the NWK layer shall issue an MLME- SCAN.request primitive asking the MAC sub-layer to perform an active scan.

Every beacon frame received during the scan having a non-zero length payload shall cause the MLME-BEACON-NOTIFY.indication primitive to be issued from the MAC sub-layer of the scanning device to its NLME. The NLME of the scanning device shall check the ExtendedPANId contained within the beacon payload to see if it is of the correct value. If not, the beacon is ignored. Otherwise, the device shall copy the relevant information from each received beacon (see Figure 3.51 for the structure of the beacon payload) into its neighbor table (see Table 3.53 and Table 3.54 for the contents of a neighbor table entry).

Once the MAC sub-layer signals the completion of the scan by issuing the MLME-SCAN.confirm primitive to the NLME, the NWK layer shall search its neighbor table for a suitable parent device. A suitable parent device shall advertise device capacity of the type requested in the JoinAsRouter parameter, shall have the most recent update id, where the determination of most recent update id must take into account that the update id will wrap back to zero, and shall have a link cost (see section 3.6.3.1) of 3, at most. If the neighbor table contains no devices that are suitable parents, the NLME shall respond with an NLME-JOIN.confirm with a Status parameter of NOT_PERMITTED. If the neighbor table has more than one device that could be a suitable parent, the device which is at a minimum depth from the ZigBee coordinator shall be chosen.

Once a suitable parent is identified the device shall set its *nwkParentInformation* value in the NIB to 0, then the NLME shall construct a NWK rejoin request command frame. The destination address field of the NWK header shall have a value equal to the 16-bit network address of the parent candidate chosen from the neighbor table. The source address field of the NWK header shall be set to the value of the *nwkNetworkAddress* attribute of the NIB. Both the source IEEE address field and the destination IEEE address field shall be present in the NWK header. If the device is joining this network for the first time, and the value of the *nwkNetworkAddress* attribute of its NIB has a value of 0xffff indicating that it is not currently joined to a network, the device shall select a 16-bit network address for itself and set the *nwkNetworkAddress* attribute to this value. The address should be randomly selected according to the procedures outlined in section 3.6.1.7. In this case, and in any case where the *nwkAddrAlloc* attribute of the NIB has a value of 0x02 indicating stochastic addressing, the allocate address sub-field of the capability information field of the command payload shall be set to 0 indicating a self-selected address.

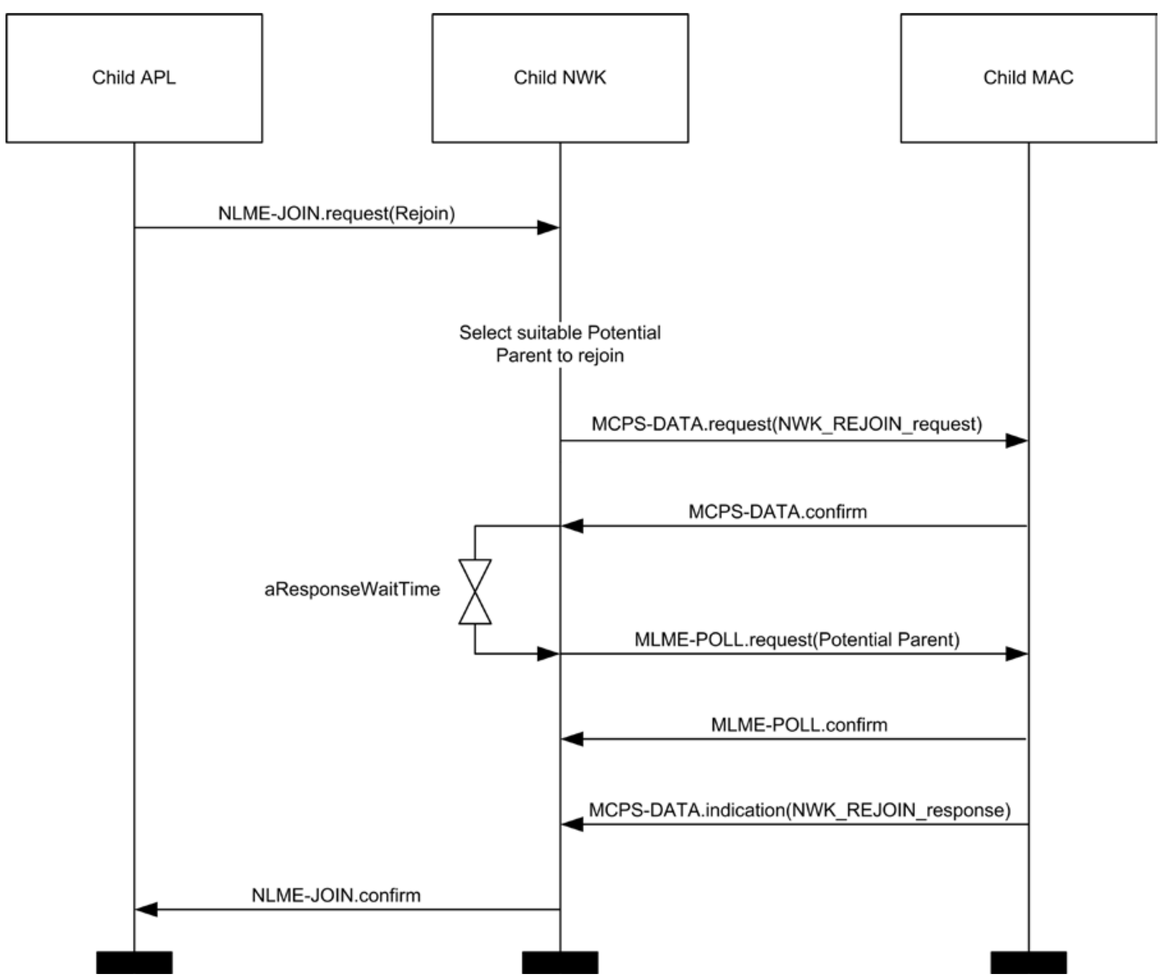
After the successful transmission of the rejoin request command using the MAC data service, the network layer shall load a countdown timer with a value of *aResponseWaitTim*e ([B1]). If this timer elapses before a rejoin response command frame is received, then the rejoin was unsuccessful. If the receiver on when idle field of the CapabilityInformation parameter is equal to 0, the device shall issue a MLME-POLL.request to the potential parent to retrieve the rejoin response command. If the receiver on when idle field is equal to 1, polling is not required.

Note: Polling more than once before *aResponseWaitTim*e ([B1]) elapses is permitted.

On receipt of a rejoin response command frame, after the above procedure or at any other time, the device shall check the destination IEEE address field and the source IEEE address fields of the command frame NWK header. If the destination IEEE address field is not equal in value to the IEEE address of the receiving device or if the source IEEE address field is not equal in value to the IEEE address of the most recent potential parent to which a rejoin request command frame was sent (or the current parent in the case of an unsolicited rejoin response), then the rejoin response command frame shall be discarded without further processing.

If the rejoin status field within the rejoin response command frame indicates a refusal to permit rejoining on the part of the neighboring device (that is, PAN at capacity or PAN access denied), then the device attempting to rejoin should set the potential parent bit to 0 in the corresponding neighbor table entry to indicate a failed join attempt. Setting the potential parent bit to 0 ensures that the NWK layer will not issue another request to rejoin to the same neighboring device. If the attempt to join was unsuccessful, the NLME shall attempt to find another suitable parent from the neighbor table. If no such device can be found, the NLME shall issue the NLME-JOIN.confirm primitive with the Status parameter set to NOT_PERMITTED. If the attempt to join is unsuccessful and there is a second neighboring device that could be a suitable parent, the NWK layer shall initiate the NWK rejoin procedure with the second device. The NWK layer shall repeat this procedure until it either rejoins the PAN successfully or exhausts its options to rejoin the PAN. If the device cannot successfully rejoin the PAN specified by the next higher layer, the NLME shall terminate the procedure by issuing the NLME-JOIN.confirm primitive with the Status parameter set to NOT_PERMITTED. In this case, the device shall not receive a valid logical address and shall not be permitted to transmit on the network. If the attempt to rejoin was successful, the NWK rejoin response command received by the NWK layer shall contain a 16-bit logical address unique to that network, which the child can use in future transmissions. Note that this address may be identical to the current 16-bit network address of the device stored in the *nwkNetworkAddress* attribute of the NIB. The NWK layer shall then set the relationship field in the corresponding neighbor table entry to indicate that the neighbor is its parent. By this time, the parent shall have added the new device to its neighbor table. Furthermore, the NWK layer shall update the values of *nwkNetworkAddress*, *nwkUpdateId*, and *nwkPANId* in the NIB if necessary.

Figure 3.38 Child Rejoin Procedure



3.6.1.4.2.2 Parent Procedure

The procedure for a ZigBee coordinator or router to rejoin a device to its network using the NWK rejoin procedure is initiated by the arrival of a NWK layer rejoin command frame via the MAC data service. Only those devices that are either ZigBee coordinators or ZigBee routers shall initiate this procedure. If this procedure is initiated on any other device, the NLME shall terminate the procedure. When this procedure is initiated, the NLME of a potential parent shall first determine whether it already has knowledge of the requesting device. To do this, the NLME shall search its neighbor table in order to determine whether a matching 64-bit, extended address can be found. If an extended address match is found, the NLME shall check that the supplied DeviceCapabilities match the device type on record in the neighbor table. If the device type matches, the NLME shall consider the join attempt successful and use the 16-bit network address found in its neighbor table as the network address of the joining device. If a device type match is not found, the NLME shall remove all records of the device in its neighbor table and restart processing of the NWK layer rejoin command.

If the potential parent does not have the capacity to accept the joining device, the NLME shall terminate the procedure and indicate this fact in the subsequent rejoin response command. The Status parameter of this command shall indicate that the PAN is at capacity.

If the request to rejoin is granted, the NLME of the parent shall create a new entry for the child in its neighbor table, or modify the existing entry if one such already exists, using the supplied device information, and indicate a successful rejoin by replying to the requesting device with a NWK rejoin response command. If the *nwkAddrAlloc* attribute of the NIB has a value of 0x00, indicating tree addressing, the NLME shall allocate new a 16-bit network address for the joining device. See section 3.6.1.6 and section 3.6.1.7 for an explanation of the address assignment mechanisms.

If the *nwkAddrAlloc* attribute of the NIB does not have a value of 0x00, the allocate address sub-field of the capabilities information field of the rejoin request command frame payload may have a value of 0 indicating a self-assigned or pre-existing network address. In this case, as is the case with all NWK command frames, the 16-bit network address in the source address field of the NWK header, in combination with the 64-bit IEEE address from the source IEEE address field of the network header should be checked for address conflicts as described in section 3.6.1.9. If an address conflict is discovered, a new, and non-conflicting, address shall be chosen for the joining device and shall be placed in the network address field of command frame payload of the outgoing rejoin response command frame. Otherwise, the contents of the source address field of the incoming rejoin request command frame shall be placed in the network address field of the command frame payload of the outgoing rejoin response command frame.

The NLME shall then notify the next higher layer that a child has just rejoined the network by issuing the NLME-JOIN.indication primitive. The procedure for successfully rejoining a device to the network is illustrated in the MSC shown in Figure 3.39.

Figure 3.39 Parent Rejoin Procedure

3.6.1.4.3 Joining a Network Directly

This section specifies how a device can be directly added to a network by a previously designated parent device (ZigBee coordinator or router). In this case, the parent device is preconfigured with the 64-bit address of the child device. The following text describes how this prior address knowledge may be used to establish the parent-child relationship.

The procedure for a ZigBee coordinator or router to directly join a device to its network is initiated by issuing the NLME-DIRECT-JOIN.request primitive with the DeviceAddress parameter set to the address of the device to be joined to the network. Only those devices that are either a ZigBee coordinator or a ZigBee router may initiate this procedure. If this procedure is initiated on any other device, the NLME may terminate the procedure and notify the next higher layer of the illegal request by issuing the NLME-DIRECT-JOIN.confirm primitive with the Status parameter set to INVALID_REQUEST.

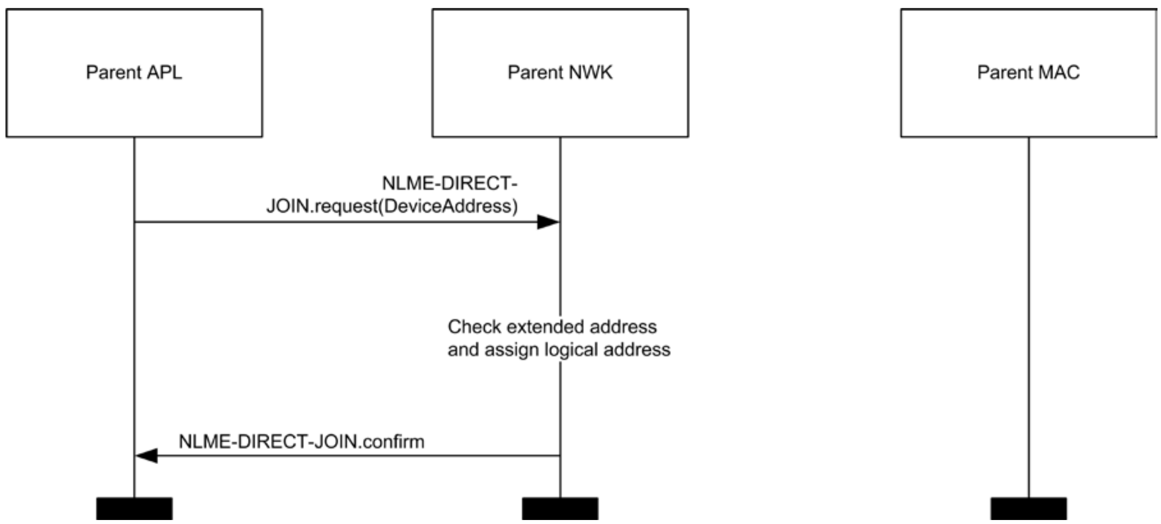
When this procedure is initiated, the NLME of the parent shall first determine whether the specified device already exists on its network. To do this, the NLME shall search its neighbor table in order to determine whether a matching 64-bit, extended address can be found. If a match is found, the NLME shall terminate the procedure and notify the next higher layer that the device is already present in the device listby issuing the NLME-DIRECT-JOIN.confirm primitive with the Status parameter set to ALREADY_PRESENT.

If a match is not found, the NLME shall, if possible, allocate a 16-bit network address for the new device as well as a new neighbor table entry. See section 3.6.1.6 and section 3.6.1.7 for an explanation of the address assignment mechanisms. If the parent device has no more room in its neighbor table, the NLME shall terminate the procedure and notify the next higher layer of the unavailable capacity by issuing the NLME-DIRECT-JOIN.confirm primitive with the Status parameter set to NEIGHBOR_TABLE_FULL. If capacity is available, the NLME shall inform the next higher layer that the device has joined the network by issuing the NLME-DIRECT-JOIN.confirm primitive with the Status parameter set to SUCCESS.

Once the parent has added the child to its network, it is still necessary for the child to make contact with the parent to complete the establishment of the parent-child relationship. The child shall fulfill this requirement by initiating the orphaning procedure, which is described in section 3.6.1.4.3.1.

A parent that supports direct joining shall follow the procedure illustrated in Figure 3.40 to successfully join a device to the network directly. This procedure does not require any over-the-air transmissions.

Figure 3.40 Joining a Device to a Network Directly



3.6.1.4.3.1 Joining or Re-joining a Network Through Orphaning

This section specifies how the orphaning procedure can be initiated by a device that has been directly joined to a network (joining through orphaning) or by a device that was previously joined to a network but has lost contact with its parent (re-joining through orphaning).

A device that has been added to a network directly shall initiate the orphan procedure in order to complete the establishment of its relationship with its parent. The application on the device will determine whether to initiate this procedure and, if so, will notify the network layer upon power up.

A device that was previously joined to a network has the option of initiating the orphan procedure if its NLME repeatedly receives communication failure notifications from its MAC sub-layer.

3.6.1.4.3.2 Child Procedure

The optional joining through orphaning procedure is initiated by a device using the NLME-JOIN.request primitive with the RejoinNetwork parameter set to 0x01.

When this procedure is initiated, the NLME shall first request that the MAC sub-layer perform an orphan scan over the over the set of channels given by the ScanChannels parameter. An orphan scan is initiated by issuing the MLME-SCAN.request primitive to the MAC sub-layer, and the result is communicated back to the NLME via the MLME-SCAN.confirm primitive.

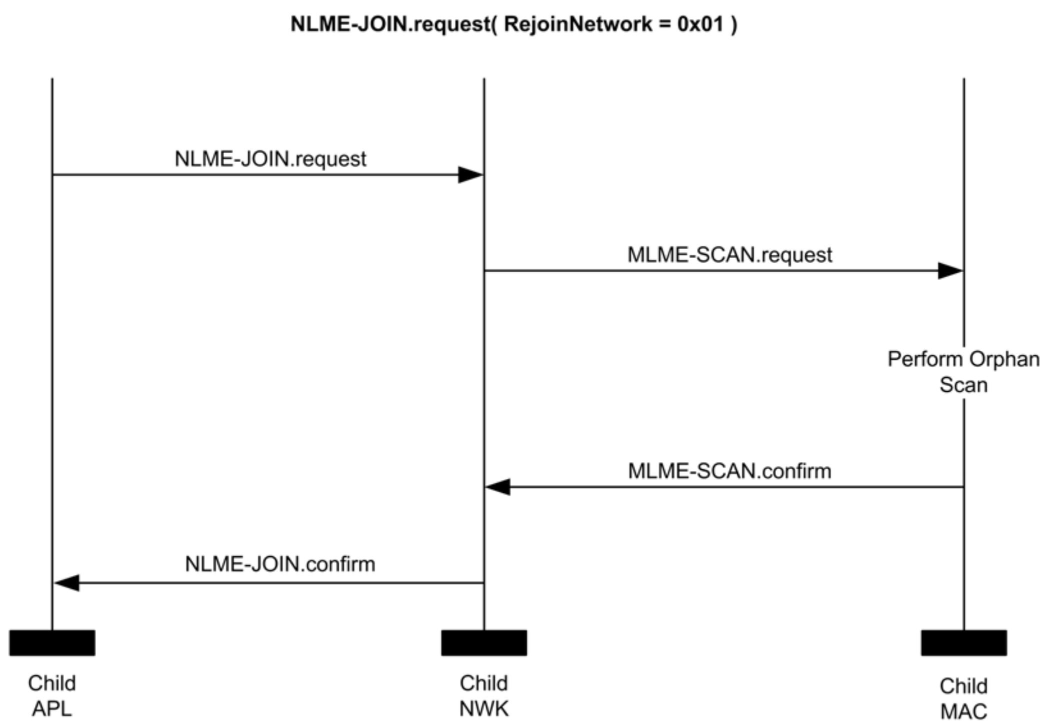
If the child has found its parent, the orphan scan was successful and the NLME shall inform the next higher layer of the success of its request to join or re-join the network by issuing the NLME-JOIN.confirm primitive with the Status parameter set to SUCCESS.

Note that if the child device is joining for the first time or if the child device has previously been joined to the network, but has failed to retain tree depth information as prescribed in section 3.6.1.8, it may not be able to operate correctly on the network without taking measures, outside the scope of this specification, for the recovery of this information.

If the orphan scan was unsuccessful (the parent has not been found), the NLME shall terminate the procedure and notify the next higher layer that no networks were found. This is achieved by issuing the NLME-JOIN.confirm primitive with the Status parameter set to NO_NETWORKS.

The procedure for a child to successfully join or re-join a network through orphaning is illustrated in the MSC shown in Figure 3.41.

Figure 3.41 Child Procedure for Joining or Re-Joining a Network through Orphaning



3.6.1.4.3.3 Parent Procedure

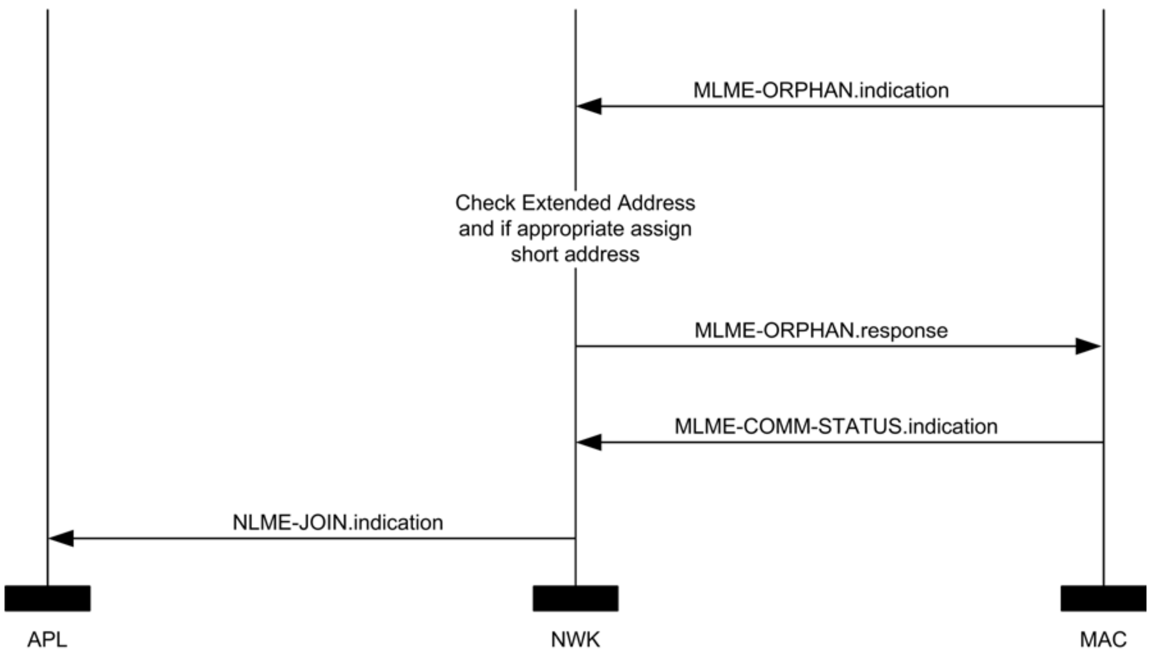
A device is notified of the presence of an orphaned device when it receives the MLME-ORPHAN.indication primitive from the MAC sub-layer. Only devices that are either ZigBee coordinators or ZigBee routers (that is, devices with parental capabilities) shall initiate this procedure. If this procedure is initiated by any other device, the NLME shall terminate the procedure.

When this procedure is initiated, the NLME shall first determine whether the orphaned device is its child. This is accomplished by comparing the extended address of the orphaned device with the addresses of its children, as recorded in its neighbor table. If a match is found (the orphaned device is its child), the NLME shall obtain the corresponding 16-bit network address and include it in its subsequent orphan response to the MAC sub-layer. The orphan response to the MAC sub-layer is initiated by issuing the MLME-ORPHAN.response primitive, and the status of the transmission is communicated back to the NLME via the MLME-COMM-STATUS.indication primitive.

If an address match is not found (the orphaned device is not its child), the procedure shall be terminated without indication to the higher layer.

The procedure for a parent to join or re-join its orphaned child to the network is illustrated in the MSC shown in Figure 3.42.

Figure 3.42 Parent Procedure for Joining or Re-Joining a Device to Its Network through Orphaning



3.6.1.5 Neighbor Tables

The neighbor table of a device shall contain information on every device within transmission range, up to some implementation-dependent limit.

The neighbor table is useful in two contexts. First of all, it is used during network discovery or rejoining to store information about routers within RF reception range that may be candidate parents. Second, after the device has joined a network, it is used to store relationship and link-state information about neighboring devices in that network. A table entry shall be updated every time a device receives any frame from the corresponding neighbor.

The outgoing cost field contains the cost of the link as measured by the neighbor. The value is obtained from the most recent link status command frame received from the neighbor. A value of 0 indicates that no link status command listing this device has been received.

The age field indicates the number of *nwkLinkStatusPeriod* intervals that have passed since the last link status command frame was received, up to a maximum value of *nwkRouterAgeLimit*.

Mandatory and optional data that are used in normal network operation are listed in Table 3.53.

8218 **Table 3.53 Neighbor Table Entry Format**

Field Name	**Field Type**	**Valid Range**	**Description**
Extended address	Integer	An extended 64-bit, IEEE address	64-bit IEEE address that is unique to every device.
Network address	Network address	0x0000 – 0xfff7	The 16-bit network address of the neighboring device. This field shall be present in every neighbor table entry.
Device type	Integer	0x00 – 0x02	The type of neighbor device: 0x00 = ZigBee coordinator 0x01 = ZigBee router 0x02 = ZigBee end device This field shall be present in every neighbor table entry.
RxOnWhenIdle	Boolean	TRUE or FALSE	Indicates if neighbor's receiver is enabled during idle periods: TRUE = Receiver is on FALSE = Receiver is off This field should be present for entries that record the parent or children of a ZigBee router or ZigBee coordinator.
End Device Configuration	Bitmask	0x0000 – 0xFFFF	The end device's configuration. See **Error! Reference source not found.** section 3.4.11.3.2. The default value shall be 0.
Timeout Counter	Integer	0x00000000 – 0x00F00000	This field indicates the current time remaining, in seconds, for the end device.

Field Name	Field Type	Valid Range	Description
Device Timeout	Integer	0x00000000 – 0x0001FA40	This field indicates the timeout, in seconds, for the end device child. The default value for end device entries is calculated by using the *nwkEndDeviceTimeoutDefault* value and indexing into Table 3.44, then converting the value to seconds. End Devices may negotiate a longer or shorter time using the NWK Command End Device Timeout Request.
Relationship	Integer	0x00 – 0x05	The relationship between the neighbor and the current device: 0x00=neighbor is the parent 0x01=neighbor is a child 0x02=neighbor is a sibling 0x03=none of the above 0x04=previous child 0x05=unauthenticated child This field shall be present in every neighbor table entry.
Transmit Failure	Integer	0x00 – 0xff	A value indicating if previous transmissions to the device were successful or not. Higher values indicate more failures. This field shall be present in every neighbor table entry.
LQI	Integer	0x00 – 0xff	The estimated link quality for RF transmissions from this device. See section 3.6.3.1 for a discussion of how this is calculated. This field shall be present in every neighbor table entry.
Outgoing Cost	Integer	0x00 - 0xff	The cost of an outgoing link as measured by the neighbor. A value of 0 indicates no outgoing cost is available. This field is mandatory if *nwkSymLink* = TRUE.

Field Name	Field Type	Valid Range	Description
Age	Integer	0x00 - 0xff	The number of nwkLinkStatusPeriod intervals since a link status command was received. This field is mandatory if nwkSymLink = TRUE.
Incoming beacon timestamp	Integer	0x000000-0xffffff	The time, in symbols, at which the last beacon frame was received from the neighbor. This value is equal to the timestamp taken when the beacon frame was received, as described in IEEE 802.15.4-2003 [B1]. This field is optional.
Beacon transmission time offset	Integer	0x000000-0xffffff	The transmission time difference, in symbols, between the neighbor's beacon and its parent's beacon. This difference may be subtracted from the corresponding incoming beacon timestamp to calculate the beacon transmission time of the neighbor's parent. This field is optional.
Keepalive Received	Boolean	TRUE or FALSE	This value indicates at least one keepalive has been received from the end device since the router has rebooted.

Information that may be used during network discovery and rejoining, as described above, is shown in Table 3.54. All of the fields shown are optional and should not be retained after the NLME has chosen a network to join. Neighbor table entries corresponding to devices that are not members of the chosen network should similarly be discarded.

Table 3.54 Additional Neighbor Table Fields

Field Name	Field Type	Valid Range	Description
Extended PAN ID	Integer	0x0000000000000001 - 0xfffffffffffffffe	The 64-bit unique identifier of the network to which the device belongs.
Logical channel	Integer	Selected from the available logical channels supported by the PHY.	The logical channel on which the network is operating.
Depth	Integer	0x00 – 0x0f	The tree depth of the neighbor device.
Beacon order	Integer	0x00 – 0x0f	The IEEE 802.15.4 beacon order for the device.
Permit joining	Boolean	TRUE or FALSE	An indication of whether the device is accepting joining requests. TRUE = device is accepting join requests. FALSE =device is not accepting join requests.
Potential parent	Integer	0x00 – 0x01	An indication of whether the device has been ruled out as a potential parent. 0x00 indicates that the device is not a potential parent. 0x01 indicates that the device is a potential parent.

3.6.1.6 Distributed Address Assignment Mechanism

The default value of the NIB attribute *nwkAddrAlloc* is 0x00, where network addresses are assigned using a distributed addressing scheme that is designed to provide every potential parent with a finite sub-block of network addresses. These addresses are unique within a particular network and are given by a parent to its children. The ZigBee coordinator determines the maximum number of children any device, within its network, is allowed. Of these children, a maximum of *nwkMaxRouters* can be router-capable devices. The remaining devices shall be reserved for end devices. Every device has an associated depth that indicates the minimum number of hops a transmitted frame must travel, using only parent-child links, to reach the ZigBee coordinator. The ZigBee coordinator itself has a depth of 0, while its children have a depth of 1. Multi-hop networks have a maximum depth that is greater than 1. The ZigBee coordinator also determines the maximum depth of the network.

Given values for the maximum number of children a parent may have, *nwkMaxChildren* (*Cm*), the maximum depth in the network, *nwkMaxDepth* (*Lm*), and the maximum number of routers a parent may have as children, *nwkMaxRouters* (*Rm*), we may compute the function, *Cskip(d)*, essentially the size of the address sub-block being distributed by each parent at that depth to its router-capable child devices for a given network depth, *d*, as follows:

$$Cskip(d) = \begin{cases} 1 + Cm \cdot (Lm - d - 1), \text{if } \mathrm{Rm} = 1 \\ \dfrac{1 + Cm - Rm - Cm * Rm^{Lm-d-1}}{1 - Rm}, \text{otherwise} \end{cases}$$

If a device has a *Cskip(d)* value of 0, then it shall not be capable of accepting children and shall be treated as a ZigBee end device for purposes of this discussion. The NLME of the device shall set the End device Capacity and Router Capacity sub fields of the MAC sub-layer beacon payload to 0.

A parent device that has a *Cskip(d)* value greater than 0 shall accept child devices and shall assign addresses to them differently depending on whether or not the child device is router-capable.

Network addresses shall be assigned to router-capable child devices using the value of *Cskip(d)* as an offset. A parent assigns an address that is 1 greater than its own to its first router-capable child device. Subsequently assigned addresses to router-capable child devices are separated from each other by *Cskip(d)*. A maximum of *nwkMaxRouters* of such addresses shall be assigned.

Network addresses shall be assigned to end devices in a sequential manner with the n^{th} address, , given by the following equation:

$$A_n = A_{parent} + Cskip(d) * Rm + n$$

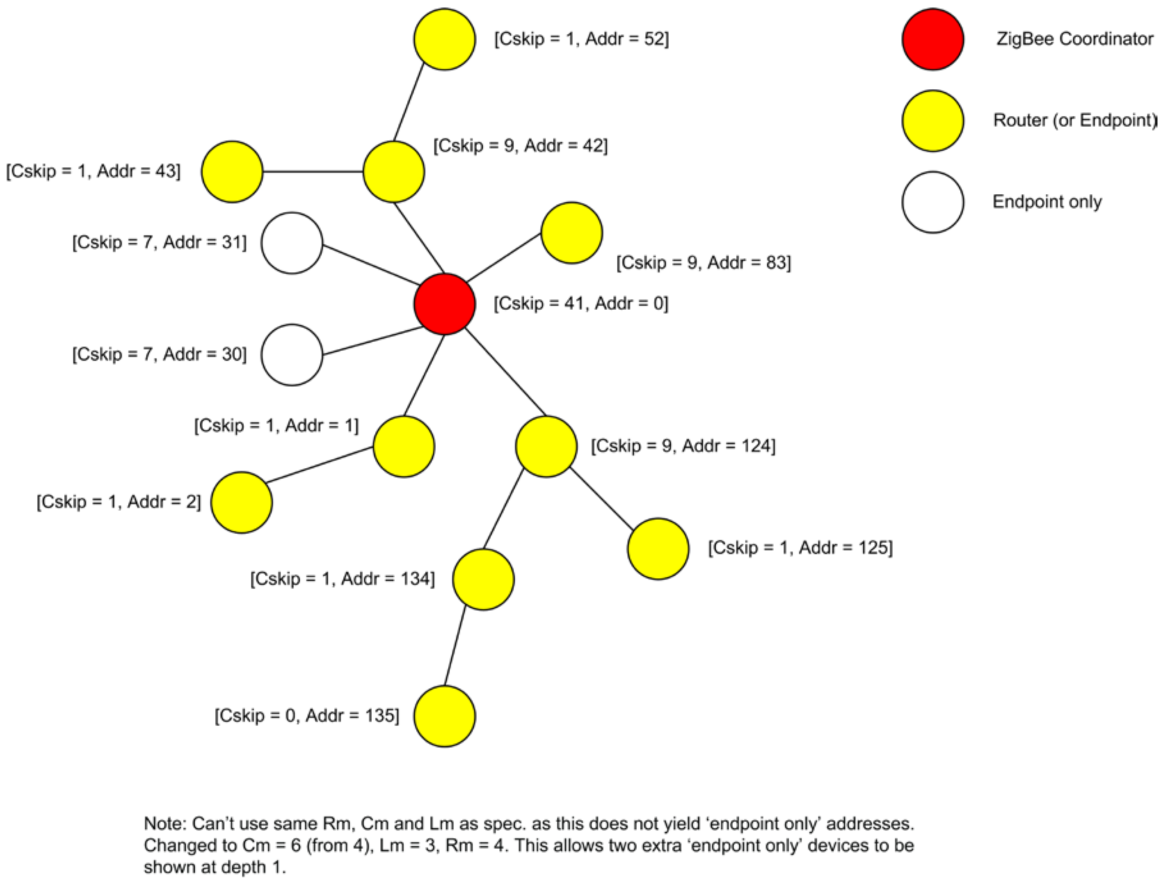
Where *d(1<n< (Cm - Rm))* and A_{parent} represents the address of the parent.

The *Cskip(d)* values for an example network having *nwkMaxChildren*=6, *nwkMaxRouters*=4 and *nwkMaxDepth*=3 are calculated and listed in Table 3.55. Figure 3.43 illustrates the example network.

Table 3.55 Example Addressing Offset Values for Each Given Depth within the Network

Depth in the Network, *d*	**Offset Value, *Cskip*(*d*)**
d of 0	Change Cskip(d) to 31
d of 1	Change Cskip(d) to 9
d of 2	Leave Cskip(d) as 1
d of 3	Leave Cskip(d) as 0

Figure 3.43 Address Assignment in an Example Network



Because an address sub-block cannot be shared between devices, it is possible that one parent exhausts its list of addresses while a second parent has addresses that go unused. A parent having no available addresses shall not permit a new device to join the network by setting the End Device Capacity and Router Capacity sub fields of the MAC sub-layer beacon payload to 0.

In this situation, the new device shall find another parent. If no other parent is available within transmission range of the new device, the device shall be unable to join the network unless it is physically moved or there is some other change.

3.6.1.7 Stochastic Address Assignment Mechanism

When the NIB attribute *nwkAddrAlloc* has value 0x02, addresses shall be chosen at random. The value of *nwkMaxRouter* is not relevant in this case. The random address assigned shall conform to the NIST testing regimen described in reference [B12]. When a device joins the network using MAC association, its parent shall choose a random address that does not already appear in any entry in the parent's NIB. Under stochastic addressing, once a device has been assigned an address, it has no reason to relinquish that address and should retain it unless it receives an indication that its address is in conflict with that of another device on the network. Furthermore, devices may self-assign random addresses under stochastic addressing and retain them, as in the case of joining a network using the rejoin command frame (see section 3.6.1.4.2). The ZigBee coordinator, which has no parent, shall always have the address 0x0000.

3.6.1.8 Installation and Addressing

It should be clear that *nwkMaxDepth* roughly determines the number of hops in network terms from the root of the tree to the farthest end device. In principle, *nwkMaxDepth* also determines the overall network diameter. In particular, for an ideal network layout in which the ZigBee coordinator is located in the center of the network, as illustrated in Figure 3.43, the network diameter should be 2* *nwkMaxDepth*. In practice, application-driven placement decisions and order of deployment may lead to a smaller diameter. In this case, *nwkMaxDepth* provides a lower bound on the network diameter while the 2* *nwkMaxDepth* provides the upper bound.

Finally, due to the fact that the tree is not dynamically balanced, when *nwkAddrAlloc* has a value of 0x00, the possibility exists that certain installation scenarios, such as long lines of devices, may exhaust the address capacity of the network long before the real capacity is reached.

Under stochastic address assignment, *nwkMaxDepth* is related to the number of hops across the network. This is not a controlled value in networks using stochastic address assignment.

3.6.1.9 Address Conflicts

An address conflict occurs when two devices in the same network have identical values for *nwkNetworkAddress*. Preventing all such conflicts, for example by using tree address assignment and prohibiting the reuse of assigned addresses, is not always practical. This section describes how address conflicts that do occur can be detected and corrected. Address conflict detection shall be enabled if the NIB attribute *nwkUniqueAddr* is FALSE.

Note that the network addresses used in routing messages are verified during the route discovery process. The device_annc now is also used to verify addresses. The verification applies only to devices, links, and information present at the time of the discovery or device_annc. Verification can be achieved at other times, such as before sending a unicast directly to a neighbor, by sending a network status command with a status code value of 0x0e, indicating address verification.

If a device receives a broadcast data frame and discovers an address conflict as a result of the receipt, as discussed below in section 3.6.1.9.2, it should not retransmit the frame as usual but shall discard it before taking the resolution actions described below in section 3.6.1.9.3.

3.6.1.9.1 Obtaining Address Information

The NWK layer obtains address information from incoming messages, including both NWK commands and data messages. Address information from data messages is passed to the NWK layer by being added to the network address map table in the NIB.

The ability to detect address conflicts is enhanced by adding one or both of the Destination IEEE Address and Source IEEE Address fields to a message’s NWK frame. When *nwkUniqueAddr* is FALSE, all NWK command messages shall contain the source IEEE address and also the destination IEEE address if it is known by the source device.

When *nwkUniqueAddr* is FALSE, route request commands shall include the sender's IEEE address in the Sender IEEE address field. This ensures that devices are aware of their neighbors' IEEE addresses.

3.6.1.9.2 Detecting Address Conflicts

After joining a network or changing address due to a conflict, a device shall send either a device_annc or initiate a route discovery prior to sending messages.

Upon receipt of a frame containing a 64-bit IEEE address in the NWK header, the contents of the *nwkAddressMap* attribute of the NIB and neighbor table should be checked for consistency.

If the destination address field of the NWK Header of the incoming frame is equal to the *nwkNetworkAddress* attribute of the NIB then the NWK layer shall check the destination IEEE address field, if present and even if it is the 0xffffffffffffffff address, against the value of *aExtendedAddress*. If the IEEE addresses are not identical then a local address conflict has been detected on *nwkNetworkAddress*.

If a neighbor table or address map entry is located in which the 64-bit address is the null IEEE address (0x00....00), the 64-bit address in the table can be updated. However, if the 64-bit address is not the null IEEE address and does not correspond to the received 64-bit address, the device has detected a conflict elsewhere in the network.

When a broadcast frame is received that creates a new BTR, if the Source Address field in the NWK Header is equal to the *nwkNetworkAddress* attribute of the NIB then a local address conflict has been detected on *nwkNetworkAddress*.

Address conflicts are resolved as described in section 3.6.1.9.3.

3.6.1.9.3 Resolving Address Conflicts

If a ZigBee coordinator or Router determines that there are multiple users of an address that is not its own, it shall inform the network by broadcasting a network status command with a status code of 0x0d indicating address conflict, and with the offending address in the destination address field. The network status command shall be broadcast to 0xFFFD, i.e. all devices with *macRxOnWhenIdle* = TRUE. The device shall delay initiation of this broadcast by a random jitter amount bounded by *nwkcMaxBroadcastJitter*. If during this delay a network status is received with the identical payload, the device shall cancel its own broadcast.

If the device has learned of the conflict other than receiving a network status command with a status of 0x0d, then it shall inform the network by broadcasting a network status command with a status code of 0x0d indicating address conflict, and with its previous address in the destination address field. The network status command shall be broadcast to 0xFFFD, i.e. all devices with macRxOnWhenIdle= TRUE. The device shall delay initiation of this broadcast by a random jitter amount bounded by *nwkcMaxBroadcastJitter*. If during this delay a network status is received with the identical payload, the device shall cancel its own broadcast. Regardless of how it learned of the conflict, it shall implement the procedure on Detecting Address Conflicts detailed in section 3.6.1.9.2.

If the conflict is detected on a ZigBee end device or *nwkAddrAlloc* is not equal to stochastic address assignment then the device shall perform a rejoin to obtain a new address. Otherwise, the device that requires a new address shall pick a new address randomly, avoiding all addresses that appear in NIB entries.

If a parent device detects or is informed of a conflict with the address of an end device child, the parent shall pick a new address for the end device child and shall send an unsolicited rejoin response command frame to inform the end device child of the new address. To notify the next higher layer of an address change the end device shall issue an NLME-NWK-STATUS.indication with status 'Network Address Update' and the new network address as the value of the ShortAddr parameter.

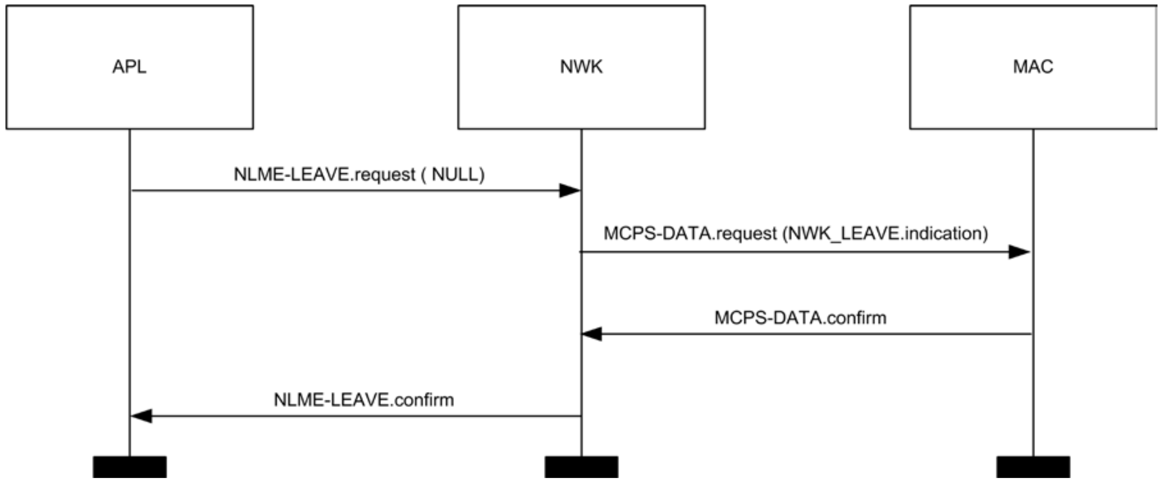
3.6.1.10 Leaving a Network

This section specifies methods for a device to remove itself from the network and for the parent of a device to request its removal. In both cases, the children of the removed device, if any, may also be removed.

3.6.1.10.1 Method for a Device to Initiate Its Own Removal from the Network

This section describes how a device can initiate its own removal from the network in response to the receipt of an NLME-LEAVE.request primitive from the next higher layer as shown in Figure 3.44.

Figure 3.44 Initiation of the Leave Procedure



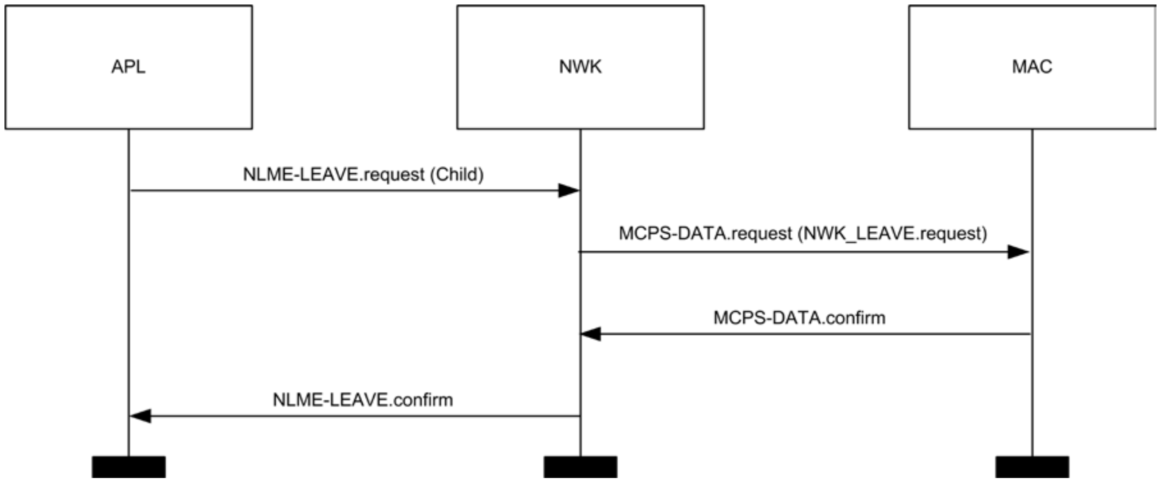
When the NWK layer of a ZigBee router or ZigBee coordinator, receives the NLME-LEAVE.request primitive with the DeviceAddress parameter equal to NULL or equal to the local device's IEEE address (indicating that the device is to remove itself) the device shall send a leave command frame using the MCPS-DATA.request primitive with the DstAddr parameter set to 0xffff indicating a MAC broadcast. The request sub-field of the command options field of the leave command frame shall be set to 0. The value of the remove children sub-field of the command options field of the leave command shall reflect the value of the RemoveChildren parameter of the NLME-LEAVE.request primitive, and the value of the Rejoin sub-field of the leave command shall reflect the value of the Rejoin parameter of the NLME-LEAVE.request primitive. After transmission of the leave command frame, it shall issue a NLME-LEAVE.confirm primitive to the higher layer with the DeviceAddress parameter equal to NULL. The Status parameter shall be SUCCESS if the leave command frame was transmitted successfully. Otherwise, the Status parameter of the NLME-LEAVE.confirm shall have the same value as the Status parameter returned by the MCPS-DATA.confirm primitive. Regardless of the Status parameter to the NLME-LEAVE.confirm, the device shall leave the network employing the procedure in 3.6.1.10.4.

If the device receiving the NLME-LEAVE.request primitive is a ZigBee end device, then the device shall send a leave command frame using the MCPS-DATA.request primitive with the DstAddr parameter set to the 16-bit network address of its parent device, indicating a MAC unicast. The request and remove children sub-fields of the command options field of the leave command frame shall be set to 0, and the rejoin flag in the command options shall be copied from the rejoin parameter of the NLME-LEAVE.request primitive. After transmission of the leave command frame, it shall set the *nwkExtendedPANId* attribute of the NIB to 0x0000000000000000 and issue a NLME-LEAVE.confirm primitive to the higher layer with the DeviceAddress parameter equal to NULL. The Status parameter shall be SUCCESS if the leave command frame was transmitted successfully. Otherwise, the Status parameter of the NLME-LEAVE.confirm shall have the same value as the Status parameter returned by the MCPS-DATA.confirm primitive. Regardless of the Status parameter to the NLME-LEAVE.confirm, the device shall leave the network employing the procedure in 3.6.1.10.4.

3.6.1.10.2 Method for a Device to Remove Its Child from the Network

This section describes how a device can initiate the removal from the network of one of its child devices in response to the receipt of an NLME-LEAVE.request primitive from the next higher layer as shown in Figure 3.45.

Figure 3.45 Procedure for a Device to Remove Its Child



When the NWK layer of a ZigBee coordinator or ZigBee router, receives the NLME-LEAVE.request primitive with the DeviceAddress parameter equal to the 64-bit IEEE address of a child device, if the relationship field of the neighbor table entry corresponding to that child device does not have a value of 0x05 indicating that the child has not yet authenticated, the device shall send a network leave command frame using the MCPS-DATA.request primitive with the DstAddr parameter set to the 16-bit network address of that child device. The request sub-field of the command options field of the leave command frame shall have a value of 1, indicating a request to leave the network. The value of the remove children sub-field of the command options field of the leave command shall reflect the value of the RemoveChildren parameter of the NLME-LEAVE.request primitive, and the value of the Rejoin sub-field of the leave command shall reflect the value of the Rejoin parameter of the NLME-LEAVE.request primitive.

If the relationship field of the neighbor table entry corresponding to the device being removed has a value of 0x05, indicating that it is an unauthenticated child, the device shall not send a network leave command frame.

Next, the NWK layer shall issue the NLME-LEAVE.confirm primitive with the DeviceAddress parameter set to the 64-bit IEEE address of the child device being removed. The Status parameter of the NLME-LEAVE.confirm primitive shall have a value of SUCCESS if the leave command frame was not transmitted, *i.e.* in the case of an unauthenticated child. Otherwise, the Status parameter of the NLME-LEAVE.confirm shall have the same value as the Status parameter returned by the MCPS-DATA.confirm primitive.

After the child device has been removed, the NWK layer of the parent should modify its neighbor table, and any other internal data structures that refer to the child device, to indicate that the device is no longer on the network. It is an error for the next higher layer to address and transmit frames to a child device after that device has been removed.

If an unauthenticated child device is removed from the network before it is authenticated, then the address formerly in use by the device being asked to leave may be assigned to another device that joins subsequently.

ZigBee end devices have no child devices to remove and should not receive NLME-LEAVE.request primitives with non-NULL DeviceAddress parameters.

3.6.1.10.3 Upon Receipt of the Leave Command Frame

Upon receipt of the leave command frame by the NWK layer via the MCPS-DATA.indication primitive, as shown in Figure 3.46, the device shall check the value of the request sub-field of the command options field of the command frame. If the request sub-field has a value of 0, then the NWK layer shall issue the NLME-LEAVE.indication primitive to the next higher layer with the device address parameter equal to the value in the source IEEE Address sub-field of the leave command frame. The device should also modify its neighbor table, and any other internal data structures that refer to the leaving device, to indicate that the leaving device is no longer on the network. It is an error for the next higher layer to address and transmit frames to a device after that device has left the network.

Figure 3.46 On Receipt of a Leave Command

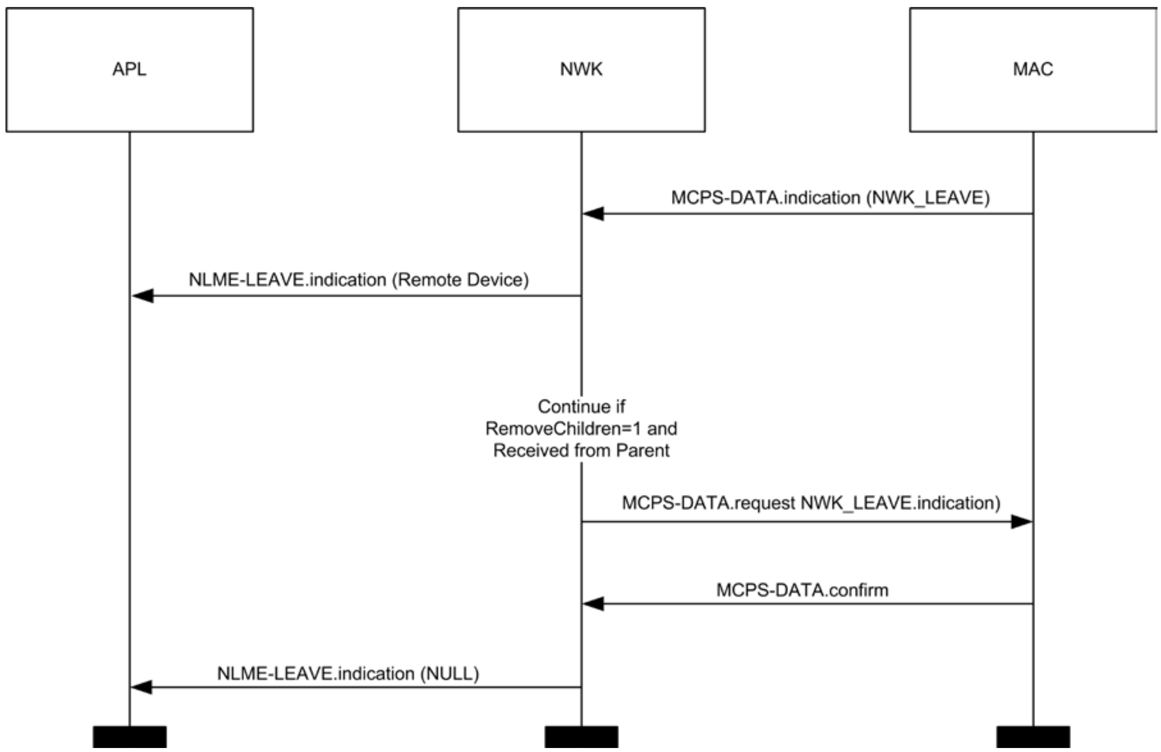
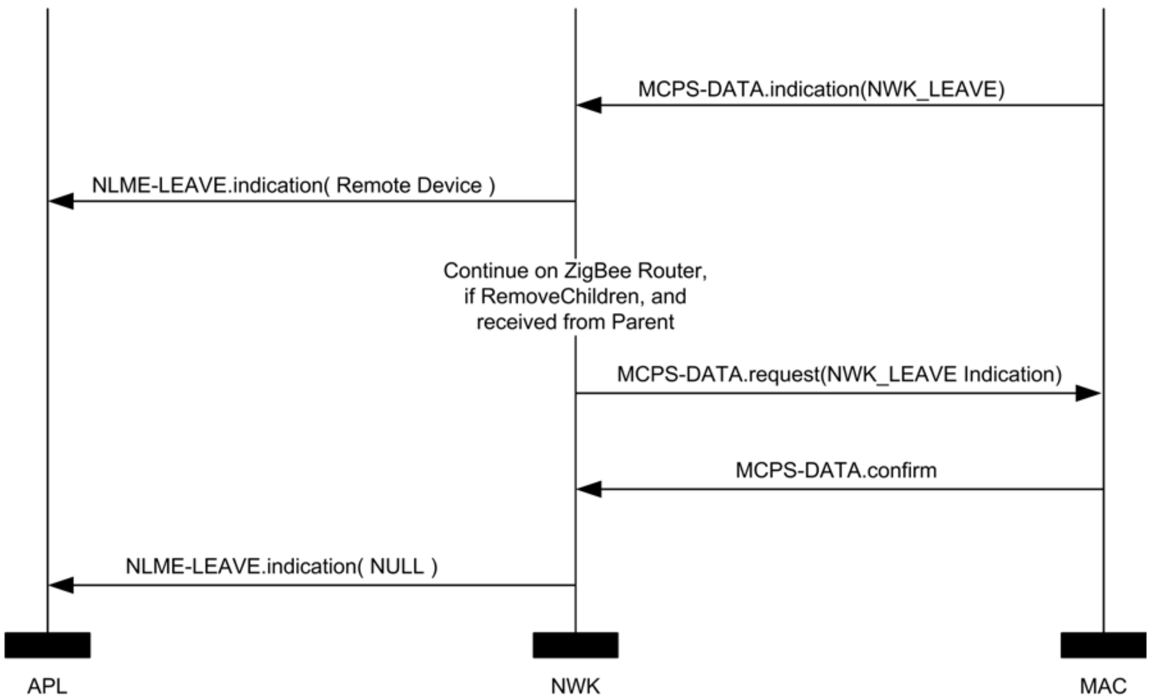


Figure 3.47 On Receipt of a Leave Command by a ZED



If, on receipt by the NWK layer of a ZigBee router of a leave command frame as described above, the SrcAddr parameter of the MCPS-DATA.indication that delivered the command frame is the 16-bit network address of the parent of the recipient, and the value of the remove children sub-field of the command options field is found to have a value of 1, then the recipient shall send a leave command frame using the MCPS-DATA.request primitive with the DstAddr parameter set to 0xffff indicating a MAC broadcast. The request sub-field of the command options field of the leave command frame shall be set to 0.

The value of the remove children sub-field and the rejoin sub-field of the command options field of the outgoing leave command shall reflect the value of the same field for the incoming leave command frame. After transmission of the leave command frame, it shall set the *nwkExtendedPANId* attribute of the NIB to 0x0000000000000000 and it shall issue a NLME-LEAVE.indication primitive to the higher layer with DeviceAddress parameter equal to NULL.

If the request sub-field has a value of 1 then the procedure in section 3.6.1.10.3.1shall be executed.:[1]

3.6.1.10.3.1 Validation of the leave request

The following procedure applies to processing of the NWK Leave (request) command frame and the ZDO Mgmt_leave_req.

1. If the device is a ZigBee Coordinator, the message shall be dropped and no further processing shall be performed.
2. If the device is ZigBee Router, the following shall be performed:
    a. The device shall not consider the Relationship field within the nwkNeighborTable entry corresponding to the sending device.
    b. If the nwkLeaveRequestAllowed in the NIB is TRUE, the device shall perform the procedure described in 3.6.1.10.1. No further processing is performed.

[1] CCB 1548

c. Otherwise if nwkLeaveRequestAllowed in the NIB is FALSE, no further processing is performed.

3. If the device is a ZigBee End Device, the following shall be performed:

   a. Examine the nwkNeighborTable for an entry where the Network Address is the same as the SrcAddr parameter of the MCPS-DATA.indication primitive that delivered the NWK command.

      i. If no entry is found, then no further processing shall be done.

   b. If the corresponding entry in the nwkNeighborTable has a Relationship value that is not 0x00 (neighbor is the parent), then no further processing shall be done.

   c. The sending device is the parent of the receiving device, the receiving device shall perform the procedure described in 3.6.1.10.1. No further processing is performed.

4. No further processing is performed.

If a ZigBee end device receives a leave command frame as described above and the SrcAddr parameter of the MCPS-DATA.indication that delivered the command frame is the 16-bit network address of the parent of the recipient, it shall set the *nwkExtendedPANId* attribute of the NIB to 0x0000000000000000 and shall issue a NLME-LEAVE.indication primitive to the higher layer with DeviceAddress parameter equal to NULL.

The NWK layer may employ retry techniques, as described in section 3.6.5 to enhance the reliability of the leave procedure but, beyond this note, these mechanisms are outside the scope of this specification.

3.6.1.10.4 Local Process for Leaving the network

Upon receipt of a NLME-LEAVE.request primitive or the NWK layer leave command, the following shall be employed.

1. If the Rejoin value is set to 1 in either the NLME-LEAVE.request primitive or the NWK Leave command, it shall do the following.

   a. The device may execute the rejoin procedure by issuing an NLME-JOIN.request with the RejoinNetwork set to 1.

   b. No further processing shall take place.

2. If the Rejoin value is set to 0, it shall clear the following values in the NIB:

   a. nwkNeighborTable
   b. nwkRouteTable
   c. nwkManagerAddr
   d. nwkUpdateId
   e. nwkNetworkAddress
   f. nwkGroupIDTable
   g. nwkExtendedPANID
   h. nwkRouteRecordTable
   i. nwkIsConcentrator
   j. nwkConcentratorRadius
   k. nwkSecurityMaterialSet
   l. nwkActiveKeySeqNumber

m. nwkAddressMap
n. nwkPANID
o. nwkTxTotal
p. nwkParentInformation

3. The device is no longer operating on the network.

3.6.1.11 Changing the ZigBee Coordinator Configuration

If the next higher layer of a ZigBee coordinator device wishes to change the configuration of the network, it shall request that the MAC sub-layer instigate the changes in its PIB. The ZigBee coordinator configuration is composed of the following items:

- Whether or not the device wishes to be a ZigBee parent
- The beacon order of the MAC superframe
- The superframe order of the MAC superframe
- Whether or not battery life extension mode is to be used

A change to the ZigBee coordinator configuration is initiated by issuing the NLME-NETWORK-FORMATION.request primitive to the NLME. The status of the attempt is communicated back via the NLME-NETWORK-FORMATION.confirm primitive.

For more details on the impact of such changes imposed on the MAC sub-layer see IEEE 802.15.4-2003 [B1].

3.6.1.12 Resetting a Device

The NWK layer of a device shall be reset immediately following initial power-up, before a join attempt to a new network and after a leave attempt where the device is not intending to rejoin the network. This process should not be initiated at any other time. A reset is initiated by issuing the NLME-RESET.request primitive to the NLME and the status of the attempt is communicated back via the NLME-RESET.confirm primitive. The reset process shall clear the routing table entries of the device.

Some devices may store NWK layer quantities in non-volatile memory and restore them after a reset. The WarmStart parameter of the NLME-RESET.request may also be used for this purpose. When *nwkAddrAlloc* is equal to 0x00, a device always gets a network address from its parent upon joining or rejoining. The new network address may be different from its old network address. In such a case, any device that is communicating with the device that has been reset must rediscover the device using higher-layer protocols. When *nwkAddrAlloc* is equal to 0x02, a device may use the same address on rejoining a network and therefore should not discard its address on reset unless it does not intend to rejoin the same network.

3.6.1.13 Managing a PANId Conflict

Since the 16-bit PANID is not a unique number there is a possibility of a PANId conflict. The next section explains how — through the use of the Network Report and Network Update command frames — the PANId of a network can be updated.

3.6.1.13.1 Detecting a PANId Conflict

Any device that is operational on a network and receives an MLME-BEACON-NOTIFY.indication in which the PAN identifier of the beacon frame matches its own PAN identifier but the EPID value contained in the beacon payload is either not present or not equal to *nwkExtendedPANID*, shall be considered to have detected a PAN Identifier conflict.

A node that has detected a PAN identifier conflict shall construct a Network Report Command frame of type PAN Identifier Conflict which shall be sent to the device identified by the address given in the *nwk-ManagerAddr* attribute of the NIB. The Report Information field will contain a list of all the 16-bit PAN identifiers that are being used in the local neighborhood. How this list is created is outside the scope of the specification, however it is recommended that it be constructed from the results of an MLME-SCAN.request of type ACTIVE.

3.6.1.13.2 Upon Receipt of a Network Report Command Frame

The device identified by the 16-bit network address contained within the *nwkManagerAddr* attribute of the NIB shall be the recipient of network report command frames of type PAN identifier conflict.

On receipt of the network report command frame, the designated network layer function manager shall select a new 16-bit PAN identifier for the network. The new PAN identifier is chosen at random, but a check is performed to ensure that the chosen PAN identifier is not already in use in the local neighborhood and also not contained within the Report Information field of the network report command frame.

Once a new PAN identifier has been selected, the designated network layer function manager shall first increment the NIB attribute *nwkUpdateId* (wrapping around to 0 if necessary) and then shall construct a network update command frame of type PAN identifier update. The update information field shall be set to the value of the new PAN identifier. The network update command frame shall be sent to the ZigBee coordinator.

After it sends out this command frame, the designated network layer function manager shall start a timer with a value equal to *nwkNetworkBroadcastDeliveryTime* OctetDurations. When the timer expires, the ZigBee coordinator shall change its current PAN ID to the newly selected one by reissuing the MLME-START.request with the new PANID.

Upon transmission of the Network Update command frame the designated network layer function manager shall create a NLME-NWK-STATUS.indication primitive with the NetworkAddr parameter set to 0 and the Status parameter set to PAN Identifier Update.

3.6.1.13.3 Upon Receipt of a Network Update Command Frame

On receipt of a network update command frame of type PAN identifier update, a device shall start a timer with a value equal to *nwkNetworkBroadcastDeliveryTime* OctetDurations. When the timer expires, the device shall change its current PAN Identifier to the value contained within the Update Information field.

Upon transmission of the network update command frame the device shall create a NLME-NWK-STATUS.indication primitive with the NetworkAddr parameter set to 0 and the Status parameter set to PAN Identifier Update.

Upon receipt of the Network Update command from the device identified by the *nwkManagerAddr* attribute of the NIB, the value contained in the update id field shall be stored in *nwkUpdateId* attribute in the NIB. The beacon payload shall also be updated.

3.6.2 Transmission and Reception

3.6.2.1 Transmission

Only those devices that are currently associated shall send data frames from the NWK layer. If a device that is not associated receives a request to transmit a frame, it shall discard the frame and notify the higher layer of the error by issuing an NLDE-DATA.confirm primitive with a status of INVALID_REQUEST.

All frames handled by or generated within the NWK layer shall be constructed according to the general frame format specified in Figure 3.5 and transmitted using the MAC sub-layer data service.

For data frames originating at a higher layer, the value of the source address field MAY be supplied using the Source address parameter of the NLDE-DATA.request primitive. If a value is not supplied or when the NWK layer needs to construct a new NWK layer command frame, then the source address field SHALL be set to the value of the *macShortAddress* attribute in the MAC PIB. Support of this parameter in the NLDE-DATA.request primitive is required if GP feature is to be supported by the implementation.

In addition to source address and destination address fields, all NWK layer transmissions shall include a radius field and a sequence number field. For data frames originating at a higher layer, the value of the radius field may be supplied using the Radius parameter of the NLDE-DATA.request primitive. If a value is not supplied, then the radius field of the NWK header shall be set to twice the value of the *nwkMaxDepth* attribute of the NIB (see Constants and NIB Attributes). For data frames originating at a higher layer, the value of the sequence number field MAY be supplied using the Sequence number parameter of the NLDE-DATA.request primitive. If a value is not supplied or when the NWK layer needs to construct a new NWK layer command frame, then the NWK layer SHALL supply the value. Support of this parameter in the NLDE-DATA.request primitive is required if GP feature is to be supported by the implementation. The NWK layer on every device shall maintain a sequence number that is initialized with a random value. The sequence number shall be incremented by 1, each time the NWK layer supplies a new sequence number value for a NWK frame. The value of the sequence number shall be inserted into the sequence number field of the frame's NWK header.

Once an NPDU is complete, if security is required for the frame, it shall be passed to the security service provider for subsequent processing according to the specified security suite (see section 4.2.2). Security processing is not required if the SecurityEnable parameter of the NLDE-DATA.request is equal to FALSE. If the NWK security level as specified in *nwkSecurityLevel* is equal to 0, then the security sub-field of the frame control field shall always be set to 0.

On successful completion of the secure processing, the security suite returns the frame to the NWK layer for transmission. The processed frame will have the correct auxiliary header attached. If security processing of the frame fails and the frame was a data frame, the frame will inform the higher layer of the NLDE-DATA.confirm primitive's status. If security processing of the frame fails and the frame is a network command frame, it is discarded and no further processing shall take place.

When the frame is constructed and ready for transmission, it shall be passed to the MAC data service. An NPDU transmission is initiated by issuing the MCPS-DATA.request primitive to the MAC sub-layer. The MCPS-DATA.confirm primitive then returns the results of the transmission.

3.6.2.2 Reception and Rejection

In order to receive data, a device must enable its receiver. The next higher layer may initiate reception using the NLME-SYNC.request primitive. On a beacon-enabled network, receipt of this primitive by the NWK layer shall cause a device to synchronize with its parent's next beacon and, optionally, to track future beacons. The NWK layer shall accomplish this by issuing an MLME-SYNC.request to the MAC sub-layer. On a non-beacon-enabled network, the NLME-SYNC.request shall cause the NWK layer of a device with *macRxOnWhenIdle* set to FALSE to poll the device's parent using the MLME-POLL.request primitive.

On a non-beacon-enabled network, the NWK layer on a ZigBee coordinator or ZigBee router shall ensure, to the maximum extent feasible, that the receiver is enabled whenever the device is not transmitting. On a beacon-enabled network, the NWK layer should ensure that the receiver is enabled when the device is not transmitting during the active period of its own superframe and of its parent's superframe. The NWK layer may use the *macRxOnWhenIdle* attribute of the MAC PIB for this purpose.

Once the receiver is enabled, the NWK layer will begin to receive frames via the MAC data service. On receipt of each frame, the radius field of the NWK header shall be decremented by 1. If, as a result of being decremented, this value falls to 0, the frame shall not, under any circumstances, be retransmitted. It may, however, be passed to the next higher layer or otherwise processed by the NWK layer as outlined elsewhere in this specification.

The NWK layer SHALL accept non-incremental NWK-level values in the Sequence number field of the ZigBee Network header for consecutive packets with the same value of the Source address field of the ZigBee Network header.

On receipt of a frame with the End Device Initiator sub-field of the frame control set to 1, the following processing shall take place.

1. If the receiving device is an end device the message shall be dropped and no further processing shall take place.
2. The receiving device shall search the neighbor table for an entry where the value of the Network Address matches the value of the Source Address field of the message, and the device type is 0x02 (end device). If no entry is found then the message shall be dropped and no further processing shall take place.

The following data frames shall be passed to the next higher layer using the NLDE-DATA.indication primitive:

- Frames with a broadcast address that matches a broadcast group of which the device is a member.
- Unicast data frames and source-addressed data frames for which the destination address matches the device's network address.
- Multicast data frames whose group identifier is listed in the *nwkGroupIDTable.*

If the receiving device is a ZigBee coordinator or an operating ZigBee router, that is, a router that has already invoked the NLME-START-ROUTER.request primitive, it shall process data frames as follows:

- Messages shall be verified to determine if an end device has switched router parents. This is outlined in section 3.6.2.3
- Broadcast and multicast data frames shall be relayed according to the procedures outlined in sections 3.6.5 and 3.6.6.
- Unicast data frames with a destination address that does not match the device's network address shall be relayed according to the procedures outlined in section 3.6.3.3. (Under all other circumstances, unicast data frames shall be discarded immediately.)
- Source-routed data frames with a destination address that does not match the device’s network address shall be relayed according to the procedures outlined in section 3.6.3.3.2.
- The procedure for handling route request command frames is outlined in section 3.6.3.5.2.
- The procedure for handling route reply command frames for which the destination address matches the device's network address is outlined in section 3.6.3.5.3.
- Route reply command frames for which the destination address does not match the device's network address shall be discarded immediately. Network status command frames shall be handled in the same manner as data frames.

The NWK layer shall indicate the receipt of a data frame to the next higher layer using the NLDE-DATA.indication primitive.

On receipt of a frame, the NLDE shall check the value of the security sub-field of the frame control field. If this value is non-zero, the NLDE shall pass the frame to the security service provider (see section 4.2.2) for subsequent processing according to the specified security suite. If the security sub-field is set to 0, the *nwkSecurityLevel* attribute in the NIB is non-zero, the device is currently joined and authenticated, and the incoming frame is a NWK data frame, the NLDE shall discard the frame. If the security sub-field is set to 0, the *nwkSecurityLevel* attribute in the NIB is non-zero, and the incoming frame is a NWK command frame and the command ID is 0x06 (rejoin request), the NLDE shall only accept the frame if it is destined to itself, that is, if it does not need to be forwarded to another device. Otherwise the frame shall be dropped and no further processing done.

If the device is not joined and authenticated, or undergoing the Trust Center Rejoin process, it shall perform the following checks. If the frame is a NWK command where the security sub-field of the frame is set to zero then it shall only accept the frame if the command ID is 0x07 (rejoin response). If the frame is a NWK data frame where the security sub-field is set to 0, the device shall further examine the APDU and determine if it contains an APS command ID of 0x05 (Transport Key). If the message does not contain an APS Command of 0x05 (Transport Key), then the message shall be dropped and no further processing done. All other messages where the security sub-field is set to 0 shall be dropped and no further processing shall be done.[2]

3.6.2.3 Examination for End Devices that have changed Router Parents

A router upon receipt of a NWK command or data message must perform the following:

1. Search the neighbor table for an entry where the Network Address matches the value of the NWK Source field in the message. If no match is found then go to step 6.
2. Examine if the Device Type of the entry corresponds to a ZigBee End Device. If it does not, go to step 6.
3. Examine if the MAC source field of the message matches the NWK source field. If it does go to step 6.
4. If the message is a broadcast, examine if an entry exists in nwkBroadcastTransactionTable, if it does then go to step 6. If the message is a unicast, continue processing.
5. At this point the message indicates it has been relayed by another device on the network acting as the end device's router parent; delete the corresponding neighbor table entry.
6. Continue to process the message.

When an end device joins or rejoins it will broadcast a ZDO Device_annce, which in turn will be processed as follows:

1. Search the neighbor table for an entry where the IEEEAddr in the ZDO Device_annce command frame matches the Extended Address field of the neighbor table entry and the Device Type field in the neighbor table entry is equal to End Device (0x02).
2. If no such entry is found, skip to step 4.
3. If an entry is found and the Device_Annce was broadcast, examine the nwkBroadcastTransactionTable. If there is no entry in the nwkBroadcastTransactionTable for this message, this indicates the message was relayed by another device on the network acting as the end device's router parent.
    a. Delete the neighbor table entry with the corresponding Extended Address equal to the IEEEAddr in the Device_Annce command.
4. Continue processing the Device_Annce message.

3.6.3 Routing

ZigBee coordinators and routers shall provide the following functionality:

- Relay data frames on behalf of higher layers
- Relay data frames on behalf of other ZigBee routers
- Participate in route discovery in order to establish routes for subsequent data frames

[2] CCB 1941

- Participate in route discovery on behalf of end devices
- Participate in route repair
- Employ the ZigBee path cost metric as specified in route discovery

ZigBee coordinators or routers may provide the following functionality:

- Maintain routing tables in order to remember best available routes
- Initiate route discovery on behalf of higher layers
- Initiate route discovery on behalf of other ZigBee routers
- Initiate route repair
- Conduct neighbor routing

3.6.3.1 Routing Cost

The ZigBee routing algorithm uses a path cost metric for route comparison during route discovery and maintenance. In order to compute this metric, a cost, known as the link cost, is associated with each link in the path and link cost values are summed to produce the cost for the path as a whole.

More formally, if we define a path *P* of length *L* as an ordered set of devices and a link, as a sub-path of length 2, then the path cost

$$C\{P\} = \sum_{i=1}^{L-1} C\{[D_i, D_{i+1}]\}$$

where each of the values is referred to as a link cost. The link cost for a link *l* is a function with values in the interval defined as:

$$C\{l\} = \begin{cases} 7, \\ \min\left(7, \mathrm{round}\left(\frac{1}{p_l^4}\right)\right) \end{cases}$$

where *pl* is defined as the probability of packet delivery on the link *l*.

Thus, implementers may report a constant value of 7 for link cost or they may report a value that reflects the probability *pl* of reception — specifically, the reciprocal of that probability — which should, in turn, reflect the number of expected attempts required to get a packet through on that link each time it is used. A device that offers both options may be forced to report a constant link cost by setting the value of the NIB attribute *nwkReportConstantCost* to TRUE. If the *nwkSymLink* attribute of the NIB has a value of TRUE, then the *nwkReportConstantCost* attribute must have a value of FALSE, and the NWK layer must calculate routing cost in the manner described above.

The question that remains, however, is how *pl* is to be estimated or measured. This is primarily an implementation issue, and implementers are free to apply their ingenuity. *pl* may be estimated over time by actually counting received beacon and data frames and observing the appropriate sequence numbers to detect lost frames. This is generally regarded as the most accurate measure of reception probability. However, the most straightforward method, available to all, is to form estimates based on an average over the per-frame LQI value provided by the IEEE 802.15.4-2003 MAC and PHY. Even if some other method is used, the initial cost estimates shall be based on average LQI. A table-driven function may be used to map average LQI values onto *C{l}* values. It is strongly recommended that implementers check their tables against data derived from tests performed on production hardware, as inaccurate costs will hamper the operating ability of the ZigBee routing algorithm.

3.6.3.2 Routing Tables

A ZigBee router or ZigBee coordinator may maintain a routing table. The information that shall be stored in a ZigBee routing table entry is shown in Table 3.56. The aging and retirement of routing table entries in order to reclaim table space from entries that are no longer in use is a recommended practice; it is, however, out of scope of this specification.

Table 3.56 Routing Table Entry

Field Name	Size	Description
Destination address	2 octets	The 16-bit network address or Group ID of this route. If the destination device is a ZigBee router, ZigBee coordinator, or an end device, and *nwkAddrAlloc* has a value of 0x02, this field shall contain the actual 16-bit address of that device. If the destination device is an end device and *nwkAddrAlloc* has a value of 0x00, this field shall contain the 16-bit network address of the device's parent.
Status	3 bits	The status of the route. See Table 3.57 for values.
No route cache	1 bit	A flag indicating that the destination indicated by this address does not store source routes.
Many-to-one	1 bit	A flag indicating that the destination is a concentrator that issued a many-to-one route request.
Route record required	1 bit	A flag indicating that a route record command frame should be sent to the destination prior to the next data packet.
GroupID flag	1 bit	A flag indicating that the destination address is a Group ID.
Next-hop address	2 octets	The 16-bit network address of the next hop on the way to the destination.

Table 3.57 enumerates the values for the route status field.

Table 3.57 Route Status Values

Numeric Value	Status
0x0	ACTIVE
0x1	DISCOVERY_UNDERWAY
0x2	DISCOVERY_FAILED
0x3	INACTIVE
0x4	VALIDATION_UNDERWAY

0x5 – 0x7	Reserved

This section describes the routing algorithm. The term “routing table capacity” is used to describe a situation in which a device has the ability to use its routing table to establish a route to a particular destination device. A device is said to have routing table capacity if:

- It is a ZigBee coordinator or ZigBee router
- It maintains a routing table
- It has a free routing table entry or it already has a routing table entry corresponding to the destination

If a ZigBee router or ZigBee coordinator maintains a routing table, it shall also maintain a route discovery table containing the information shown in Table 3.58. Routing table entries are long-lived, while route discovery table entries last only as long as the duration of a single route discovery operation and may be reused.

Table 3.58 Route Discovery Table Entry

Field Name	**Size (octets)**	**Description**
Route request ID	1	A sequence number for a route request command frame that is incremented each time a device initiates a route request.
Source address	2	The 16-bit network address of the route request’s initiator.
Sender address	2	The 16-bit network address of the device that has sent the most recent lowest cost route request command frame corresponding to this entry’s route request identifier and source address. This field is used to determine the path that an eventual route reply command frame should follow.
Forward cost	1	The accumulated path cost from the source of the route request to the current device.
Residual cost	1	The accumulated path cost from the current device to the destination device.
Expiration time	2	A countdown timer indicating the number of milliseconds until route discovery expires. The initial value is *nwkcRouteDiscoveryTime.*

A device is said to have “route discovery table capacity” if:

- It maintains a route discovery table
- It has a free entry in its route discovery table

If a device has both routing table capacity and route discovery table capacity then it may be said to have “routing capacity.”

During route discovery, the information that a ZigBee router or ZigBee coordinator is required to maintain in order participate in the discovery of a particular route is distributed between a routing table entry and a route discovery table entry. Once discovery has been completed, only the routing table entry need be maintained in order for the NWK layer to perform routing along the discovered route. Throughout this section, references are made to this relationship between a routing table entry and its "corresponding" route discovery table entry and vice versa. The maintenance of this correspondence is up to the implementer since entries in the tables have no elements in common, but it is worth noting in this regard that the unique "keys" that define a route discovery are the source address of the route discovery command frame and the route request ID generated by that device and carried in the command frame payload.

If a device has the capability to initiate a many-to-one route request, it may also maintain a route record table (see Table 3.50).

3.6.3.3 Upon Receipt of a Unicast Frame

On receipt of a unicast frame from the MAC sub-layer, or an NLDE-DATA.request from the next higher layer, the NWK layer routes it according to the following procedure.

If the receiving device is a ZigBee router or ZigBee coordinator, and the destination of the frame is a ZigBee end device and also the child of the receiving device, the frame shall be routed directly to the destination using the MCPS-DATA.request primitive, as described in section 3.6.2.1. The frame shall also set the next hop destination address equal to the final destination address. Otherwise, for purposes of the ensuing discussion, define the *routing address* of a device to be its network address if it is a router or the coordinator or an end device and *nwkAddrAlloc* has a value of 0x02, or the network address of its parent if it is an end device and *nwkAddrAlloc* has a value of 0x00. Define the *routing destination* of a frame to be the routing address of the frame's NWK destination. Note that distributed address assignment makes it possible to derive the routing address of any device from its address. See section 3.6.1.6 for details.

A ZigBee router or ZigBee coordinator may check the neighbor table for an entry corresponding to the routing destination of the frame. If there is such an entry, the device may route the frame directly to the destination using the MCPS-DATA.request primitive as described in section 3.6.2.1.

A device that has routing capacity shall check its routing table for an entry corresponding to the routing destination of the frame. If there is such an entry, and if the value of the route status field for that entry is ACTIVE or VALIDATION_UNDERWAY, the device shall relay the frame using the MCPS-DATA.request primitive and set the route status field of that entry to ACTIVE if it does not already have that value. If the many-to-one field of the routing table entry is set to TRUE, the NWK shall follow the procedure outlined in section 3.6.3.5.4 to determine whether a route record command frame must be sent.

When relaying a unicast frame, the SrcAddrMode and DstAddrMode parameters of the MCPS-DATA.request primitive shall both have a value of 0x02, indicating 16-bit addressing. The SrcPANId and DstPANId parameters shall both have the value provided by the macPANId attribute of the MAC PIB for the relaying device. The SrcAddr parameter shall be set to the value of *macShortAddress* from the MAC PIB of the relaying device, and the DstAddr parameter shall be the value provided by the next-hop address field of the routing table entry corresponding to the routing destination. Bit *b0* of the TxOptions parameter should be set to 1, indicating acknowledged transmission.

The NWK Sequence Number of a replayed packet shall not be changed by a router device relaying the packet. The router device relaying a packet shall leave the NWK Sequence Number of the originating device in the NWK Sequence Number field.

If the device has a routing table entry corresponding to the routing destination of the frame but the value of the route status field for that entry is DISCOVERY_UNDERWAY, the device shall determine if it initiated the discovery by consulting its discovery table. If the device initiated the discovery, the frame shall be treated as though route discovery has been initiated for this frame, otherwise, the device shall initiate route discovery as described in section 3.6.3.5.1. The frame may optionally be buffered pending route discovery or routed along the tree using hierarchical routing, provided that the NIB attribute *nwkUseTreeRouting* has a value of TRUE. If the frame is routed along the tree, the discover route sub-field of the NWK header frame control field shall be set to 0x00.

If the device has a routing table entry corresponding to the routing destination of the frame but the route status field for that entry has a value of DISCOVERY_FAILED or INACTIVE, the device may route the frame along the tree using hierarchical routing, provided that the NIB attribute *nwkUseTreeRouting* has a value of TRUE. If the device does not have a routing table entry for the routing destination with a status value of ACTIVE or VALIDATION_UNDERWAY, and it received the frame from the next higher layer, it shall check its source route table for an entry corresponding to the routing destination. If such an entry is found and the length is less than *nwkMaxSourceRoute*, the device shall transmit the frame using source routing as described in section 3.6.3.3.1. If the device does not have a routing table entry for the routing destination and it is not originating the frame using source routing, it shall examine the discover route sub-field of the NWK header frame control field. If the discover route sub-field has a value of 0x01, the device shall initiate route discovery, as described in section 3.6.3.5.1. If the discover route sub-field has a value of 0 and the NIB attribute *nwkUseTreeRouting* has a value of TRUE then the device shall route along the tree using hierarchical routing. If the discover route sub-field has a value of 0, the NIB attribute *nwkUseTreeRouting* has a value of FALSE, and there is no routing table corresponding to the routing destination of the frame, the frame shall be discarded and the NLDE shall issue the NLDE-DATA.confirm primitive with a status value of ROUTE_ERROR.

A device without routing capacity shall route along the tree using hierarchical routing provided that the value of the NIB attribute *nwkUseTreeRouting* is TRUE. If the value of the NIB attribute *nwkUseTreeRouting* is FALSE, the frame shall be discarded. If the frame is the result of an NLDE-DATA.request from the NHL of the current device, the NLDE shall issue the NLDE-DATA.confirm primitive with a status value of ROUTE_ERROR. If the frame is being relayed on behalf of another device, the NLME shall issue a network status command frame destined for the device that is the source of the frame with a status of 0x04, indicating a lack of routing capacity. It shall also issue the NLME-NWK-STATUS.indication to the next higher layer with the NetworkAddr parameter equal to the 16-bit network address of the frame, and the Status parameter equal to 0x04, indicating a lack of routing capacity.

For hierarchical routing, if the destination is a descendant of the device, the device shall route the frame to the appropriate child. If the destination is a child, and it is also an end device, delivery of the frame can fail due to the *macRxOnWhenIdle* state of the child device. If the child has *macRxOnWhenIdle* set to FALSE, indirect transmission as described in IEEE 802.15.4-2003 [B1] may be used to deliver the frame. If the destination is not a descendant, the device shall route the frame to its parent.

Every other device in the network is a descendant of the ZigBee coordinator and no device in the network is the descendant of any ZigBee end device. For a ZigBee router with address A at depth d, if the following logical expression is true, then a destination device with address D is a descendant:

$$A < D < A + Cskip(d - 1)$$

For a definition of *Cskip(d)*, see section 3.6.1.6.

If it is determined that the destination is a descendant of the receiving device, the address N of the next hop device is given by:

$$N = D$$

for ZigBee end devices, where $D > A + Rm \times Cskip(d)$, and otherwise:

$$N = A + 1 + \left\lfloor \frac{D - (A + 1)}{Cskip(d)} \right\rfloor \times Cskip(d)$$

If the NWK layer on a ZigBee router or ZigBee coordinator fails to deliver a unicast or multicast frame for any reason, the router or coordinator shall make its best effort to report the failure. No failure should be reported as the result of a failure to deliver a NLME-NWK-STATUS. The failure reporting may take one of two forms. If the failed frame was being relayed as a result of a request from the next higher layer, then the NWK layer shall issue an NLDE-DATA.confirm with the error to the next higher layer. The value of the NetworkAddr parameter of the primitive shall be the intended destination of the frame. If the frame was being relayed on behalf of another device, then the relaying device shall send a network status command frame back to the source of the frame. The destination address field of the network status command frame shall be taken from the destination address field of the failed data frame.

In either case, the reasons for failure that may be reported appear in Table 3.42.

3.6.3.3.1 Originating a Source Routed Data Frame

If, on receipt of a data frame from the next higher layer, it is determined that the frame should be transmitted using source routing as described above, the source route shall be retrieved from the route record table.

If there are no intermediate relay nodes, the frame shall be transmitted directly to the routing destination without source routing by using the MCPS-DATA.request primitive, with the DstAddr parameter value indicating the routing destination.

If there is at least one relay node, the source route flag of the NWK header frame control field shall be set, and the NWK header source route subframe shall be present. The relay count sub-field of the source route subframe shall have a value equal to the number of relays in the relay list. The relay index sub-field shall have a value equal to 1 less than the number of relays. The relay list sub-field shall contain the list of relay addresses, least significant octet first. The relay closest to the destination shall be listed first. The relay closest to the originator shall be listed last.

The device shall relay the frame using the MCPS-DATA.request primitive. The DstAddr parameter shall have the value of the final relay address in the relay list.

3.6.3.3.2 Relaying a Source Routed Data Frame

Upon receipt of a source routed data frame from the MAC sub-layer as described in section 3.6.3.3, if the relay index sub-field of the source route sub-frame has a value of 0, the device shall check the destination address field of the NWK header of the frame. If the destination address field of the NWK header of the frame is equal in value to the *nwkNetworkAddress* attribute of the NIB, then the frame shall be passed to the next higher layer using the NLDE-DATA.indication primitive. If the destination address field is not equal to the *nwkNetworkAddress* attribute of the NIB, and the receiving device is a ZigBee router or ZigBee coordinator, the device shall relay the frame directly to the NWK header destination using the MCPS-DATA.request primitive, otherwise the frame shall be discarded silently.

If the relay index sub-field has a value other than 0, the device shall compare its network address with the address found at the relay index in the relay list. If the addresses do not match, the frame shall be discarded and no further action shall be taken. Otherwise, as long as the destination address is not the address of an end device where the relaying device is the parent, the device shall decrement the relay index sub-field by 1, and relay the frame to the address immediately prior to its own address in the relay list sub-field. If the destination address of the frame is an end device child of the relaying device, the frame shall be unicast using the MCPS-DATA.request primitive.

When relaying a source routed data frame, the NWK layer of a device shall also examine the routing table entry corresponding to the source address of the frame. If the no route cache field of the routing table entry has a value of FALSE, then the route record required field of the routing table entry shall be set to FALSE.

3.6.3.4 Link Status Messages

Wireless links may be asymmetric, that is, they may work well in one direction but not the other. This can cause route replies to fail, since they travel backwards along the links discovered by the route request.

For many-to-one routing and two-way route discovery (*nwkSymLink* = TRUE), it is a requirement to discover routes that are reliable in both directions. To accomplish this, routers exchange link cost measurements with their neighbors by periodically transmitting link status frames as a one-hop broadcast. The reverse link cost information is then used during route discovery to ensure that discovered routes use high-quality links in both directions.

3.6.3.4.1 Initiation of a Link Status Command Frame

When joined to a network, a ZigBee router or coordinator shall periodically send a link status command every *nwkLinkStatusPeriod* seconds, as a one-hop broadcast without retries. It may be sent more frequently if desired. Random jitter should be added to avoid synchronization with other nodes. See section 3.4.8 for the link status command frame format.

End devices do not send link status command frames.

3.6.3.4.2 Upon Receipt of a Link Status Command Frame

Upon receipt of a link status command frame by a ZigBee router or coordinator, the age field of the neighbor table entry corresponding to the transmitting device is reset to 0. The list of addresses covered by a frame is determined from the first and last addresses in the link status list, and the first frame and last frame bits of the command options field. If the receiver's network address is outside the range covered by the frame, the frame is discarded and processing is terminated. If the receiver's network address falls within the range covered by the frame, then the link status list is searched. If the receiver's address is found, the outgoing cost field of the neighbor table entry corresponding to the sender is set to the incoming cost value of the link status entry. If the receiver's address is not found, the outgoing cost field is set to 0.

End devices do not process link status command frames.

3.6.3.4.3 Aging the Neighbor Table

For devices using link status messages, the age fields for routers in the neighbor table are incremented every *nwkLinkStatusPeriod*. If the value exceeds *nwkRouterAgeLimit*, the outgoing cost field of the neighbor table entry is set to 0. In other words, if a device fails to receive *nwkRouterAgeLimit* link status messages from a router neighbor in a row, the old outgoing cost information is discarded. In this case, the neighbor entry is considered stale and may be reused if necessary to make room for a new neighbor. End devices do not issue link status messages and therefore should never be considered stale.

If *nwkAddrAlloc* has a value of 0x00, neighbor table entries for relatives should not be considered stale and reused.

3.6.3.5 Route Discovery

Route discovery is the procedure whereby network devices cooperate to find and establish routes through the NWK. *Unicast route discovery* is always performed with regard to a particular source device and a particular destination device. *Multicast route discovery* is performed with respect to a particular source device and a multicast group. *Many-to-one route discovery* is performed by a source device to establish routes to itself from all ZigBee routers and ZigBee coordinator, within a given radius. A source device that initiates a many-to-one route discovery is designated as a concentrator and referred to as such in this document. Throughout section 3.6.3.5 *a destination address* may be a 16-bit broadcast address, the 16-bit network address of a particular device, or a 16-bit multicast address, also known as a multicast group ID. A route request command whose destination address is the routing address of a particular device and whose route request option field does not have the multicast bit set, is a *unicast route request*. A route request command whose route request option field has the multicast bit set is a *multicast route request*. The destination address field of a multicast route request shall be a multicast group ID. A route request command payload whose destination address sub-field is a broadcast address (see Table 3.59) is a *many-to-one route request*. The multicast bit in the route request option field of a many-to-one route request shall be set to 0.

Note that on RREP new frames shall be created at every hop. In all other cases the packets shall not be not considered a “new” frame. A new frame shall be one with a new route request identifier. For RREP the sequence number is regenerated every hop. For RREC the sequence number does not change with every hop.

3.6.3.5.1 Initiation of Route Discovery

The unicast route discovery procedure for a device shall be initiated on a ZigBee router or ZigBee coordinator by the NWK layer up on receipt of an NLME-ROUTE-DISCOVERY.request primitive from the next higher layer where the DstAddrMode parameter has a value of 0x02. Or, up on receipt of an NLDE-DATA.request primitive from a higher layer with the DstAddrMode set to 0x02 and the discover route sub-field set to 0x01, for which there is no routing table entry corresponding to the routing address of the destination device (the 16-bit network address indicated by the DstAddr parameter). Or, on receipt of a frame from the MAC sub-layer for which the value of the destination address field in the NWK header is not the address of the current device, the address of an end device child of the current device, or a broadcast address and:

- The discover route sub-field of the frame control field has a value of 0x01, and
- there is no routing table entry corresponding to the routing destination of the frame, and
- either the value of the source address field of the NWK header of the received frame is the same as the 16-bit network address of one of the end device children of the current device, or
- the *nwkUseTreeRouting* attribute of the NIB has a value of TRUE.

The route discovery procedure for a multicast address shall be initiated by the NWK layer either in response to the receipt of an NLME-ROUTE-DISCOVERY.request primitive from the next higher layer where the DstAddrMode parameter has a value of 0x01, or as specified in section 3.6.6.2.2.

If the device initiating route discovery has no routing table entry corresponding to the routing address of the destination device, it shall establish a routing table entry with status equal to DISCOVERY_UNDERWAY. If the device has an existing routing table entry corresponding to the routing address of the destination with status equal to ACTIVE or VALIDATION _UNDERWAY, that entry shall be used and the status field of that entry shall retain its current value. If it has an existing routing table entry with a status value other than ACTIVE or VALIDATION_UNDERWAY, that entry shall be used and the status of that entry shall be set to DISCOVERY_UNDERWAY. The device shall also establish the corresponding route discovery table entry if one with the same initiator and route request ID does not already exist.

Each device issuing a route request command frame shall maintain a counter used to generate route request identifiers. When a new route request command frame is created, the route request counter is incremented and the value is stored in the device's route discovery table in the Route request identifier field. Other fields in the routing table and route discovery table are set as described in section 3.6.3.2.

The NWK layer may choose to buffer the received frame pending route discovery or, if the frame is a unicast frame and the NIB attribute *nwkUseTreeRouting* has a value of TRUE, set the discover route sub-field of the frame control field in the NWK header to 0 and forward the data frame along the tree.

Once the device creates the route discovery table and routing table entries, the route request command frame shall be created with the payload depicted in Figure 3.12. The individual fields are populated as follows:

- The command frame identifier field shall be set to indicate the command frame is a route request, see Table 3.40.
- The Route request identifier field shall be set to the value stored in the route discovery table entry.
- The multicast flag and destination address fields shall be set in accordance with the destination address for which the route is to be discovered.
- The path cost field shall be set to 0.

Once created, the route request command frame is ready for broadcast and is passed to the MAC sub-layer using the MCPS-DATA.request primitive.

When broadcasting a route request command frame at the initiation of route discovery, the NWK layer shall retry the broadcast *nwkcInitialRREQRetries* times after the initial broadcast, resulting in a maximum of *nwkcInitialRREQRetries* + 1 transmissions. The retries will be separated by a time interval of *nwkcRREQRetryInterval* OctetDurations.

The many-to-one route discovery procedure shall be initiated by the NWK layer of a ZigBee router or coordinator on receipt of an NLME-ROUTE-DISCOVERY.request primitive from the next higher layer where the DstAddrMode parameter has a value of 0x00. A many-to-one route request command frame is not retried; however, a discovery table entry is still created to provide loop detection during the *nwkcRouteDiscoveryTime* period. If the NoRouteCache parameter of the NLME-ROUTE-DISCOVERY.request primitive is TRUE, the many-to-one sub-field of the command options field of the command frame payload shall be set to 2. Otherwise, the many-to-one sub-field shall be set to 1. Note that in this case, the NWK layer should maintain a route record table. The destination address field of the NWK header shall be set to 0xfffc, the all-router broadcast address. The broadcast radius shall be set to the value in *nwkConcentratorRadius*. A source device that initiates a many-to-one route discovery is designated as a concentrator and referred to as such in this document and the NIB attribute *nwkIsConcentrator* should be set to TRUE. If a device has *nwkIsConcentrator* equal to TRUE and there is a non-zero value in *nwkConcentratorDiscoveryTime,* the network layer should issue a route request command frame each *nwkConcentratorDiscoveryTime*.

3.6.3.5.2 Upon Receipt of a Route Request Command Frame

Upon receipt of a route request command frame, if the device is an end device, it shall drop the frame. Otherwise, it shall determine if it has routing capacity.

If the device does not have routing capacity and the route request is a multicast route request or a many-to-one-route request, the route request shall be discarded and route request processing shall be terminated.

If *nwkAddrAlloc* is 0x00 and the device does not have routing capacity and the route request is a unicast route request, the device shall check if the frame was received along a valid path. A path is valid if the frame was received from one of the device's children and the source device is a descendant of that child device, or if the frame was received from the device's parent device and the source device is not a descendant of the device. If the route request command frame was not received along a valid path, it shall be discarded. Otherwise, the device shall check if it is the intended destination. It shall also check if the destination of the command frame is one of its end device children by comparing the destination address field of the route request command frame payload with the address of each of its end device children, if any. If either the device or one of its end device children is the destination of the route request command frame, it shall reply with a route reply command frame. When replying to a route request with a route reply command frame, the device shall construct a frame with the frame type field set to 0x01. The route reply's source address shall be set to the 16-bit network address of the device creating the route reply and the destination address shall be set to the calculated next hop address, considering the originator of the route request as the destination. The link cost from the next hop device to the current device shall be computed as described in section 3.6.3.1 and inserted into the path cost field of the route reply command frame. The route reply command frame shall be unicast to the next hop device by issuing an MCPS-DATA.request primitive.

If the device is not the destination of the route request command frame, the device shall compute the link cost from the previous device that transmitted the frame, as described in section 3.6.3.1. This value shall be added to the path cost value stored in the route request command frame. The route request command frame shall then be unicast towards the destination using the MCPS-DATA.request service primitive. The next hop for this unicast transmission is determined in the same manner as if the frame were a data frame addressed to the device identified by the destination address field in the payload.

If the device does have routing capacity and the received request is a unicast route request, the device shall check if it is the destination of the command frame by comparing the destination address field of the route request command frame payload with its own address. It shall also check if the destination of the command frame is one of its end device children by comparing the destination address field of the route request command frame payload with the address of each of its end device children, if any. If neither the device nor one of its end device children is the destination of the route request command frame, the device shall determine if a route discovery table (see Table 3.58) entry exists with the same route request identifier and source address field. If no such entry exists, one shall be created.

If the device does have routing capacity and the multicast sub-field of the route request command options field of the received route request frame indicates a multicast route request, the device shall determine whether an entry already exists in the *nwkGroupIDTable* for which the group identifier field matches the destination address field of the frame. If a matching entry is found, the device shall determine if a route discovery table (see Table 3.58) entry exists with the same route request identifier and source address field. If no such entry exists, one shall be created.

For many-to-one route requests, and for regular route requests if the *nwkSymLink* attribute is TRUE, upon receipt of a route request command frame, the neighbor table is searched for an entry corresponding to the transmitting device. If no such entry is found, or if the outgoing cost field of the entry has a value of 0, the frame is discarded and route request processing is terminated. The maximum of the incoming and outgoing costs for the neighbor is used for the purposes of the path cost calculation, instead of the incoming cost. This includes the value used to increment the path cost field of the route request frame prior to retransmission.

When creating the route discovery table entry, the fields are set to the corresponding values in the route request command frame. The only exception is the forward cost field, which is determined by using the previous sender of the command frame to compute the link cost, as described in section 3.6.3.1, and adding it to the path cost contained the route request command frame. The result of the above calculation is stored in the forward cost field of the newly created route discovery table entry. If the *nwkSymLink* attribute is set to TRUE, the device shall also create a routing table entry with the destination address field set to the source address of the route request command frame and the next hop field set to the address of the previous device that transmitted the command frame. The status field shall be set to ACTIVE. The device shall then issue a route reply command frame to the source of the route request command frame. In the case that the device already has a route discovery table entry for the source address and route request identifier pair, the device shall determine if the path cost in the route request command frame is less than the forward cost stored in the route discovery table entry. The comparison is made by first computing the link cost from the previous device that sent this frame, as described in section 3.6.3.1, then adding it to the path cost value in the route request command frame. If this value is greater than the value in the route discovery table entry, the frame shall be dropped and no further processing is required. Otherwise, the forward cost and sender address fields in the route discovery table are updated with the new cost and the previous device address from the route request command frame.

If the *nwkSymLink* attribute is set to TRUE and the received route request command frame is a unicast route request, the device shall also create a routing table entry with the destination address field set to the source address of the route request command frame and the next hop field set to the address of the previous device that transmitted the command frame. The status field shall be set to ACTIVE. The device shall then respond with a route reply command frame. In either of these cases, if the device is responding on behalf of one of its end device children, the responder address in the route reply command frame payload shall be set equal to the address of the end device child and not of the responding device.

When a device with routing capacity is not the destination of the received route request command frame, it shall determine if a route discovery table entry (see Table 3.58) exists with the same route request identifier and source address field. If no such entry exists, one shall be created. The route request timer shall be set to expire in *nwkcRouteDiscoveryTime* OctetDurations. If a routing table entry corresponding to the routing address of the destination exists and its status is not ACTIVE or VALIDATION_UNDERWAY, the status shall be set to DISCOVERY_UNDERWAY. If no such entry exists and the frame is a unicast route request, an entry shall be created and its status set to DISCOVERY_UNDERWAY. If the frame is a many-to-one route request, the device shall also create a routing table entry with the destination address field equal to the source address of the route request command frame by setting the next hop field to the address of the previous device that transmitted the command frame. If the frame is a many-to-one route request (*i.e.* the many-to-one sub-field of the command options field of the command frame payload has a non-zero value), the many-to-one field in the routing table entry shall be set to TRUE, the route record required field shall be set to TRUE[3], and the no route cache flag shall be set to TRUE if the many-to-one sub-field of the command options field of the command frame payload has a value of 2 or to FALSE if it has a value of 1. If the routing table entry is new, or if the no route cache flag is set to TRUE, or if the next hop field changed, the route record required field shall be set to TRUE, otherwise it remains unchanged. The status field shall be set to ACTIVE. When the route request timer expires, the device deletes the route request entry from the route discovery table. When this happens, the routing table entry corresponding to the routing address of the destination shall also be deleted, if its status field has a value of DISCOVERY_UNDERWAY and there are no other entries in the route discovery table created as a result of a route discovery for that destination address.

If an entry in the route discovery table already exists, the path cost in the route request command frame shall be compared to the forward cost value in the route discovery table entry. The comparison is made by computing the link cost from the previous device, as described in section 3.6.3.1, and adding it to the path cost value in the route request command frame. If this path cost is greater, the route request command frame is dropped and no further processing is required. Otherwise, the forward cost and sender address fields in the route discovery table are updated with the new cost and the previous device address from the route request command frame. Additionally, the path cost field in the route request command frame shall be updated with the cost computed for comparison purposes. If the *nwkSymLink* attribute is set to TRUE and the received route request command frame is a unicast route request, the device shall also update any routing table entry with the destination address field set to the source address of the route request command frame, and the next hop field set to the address of the previous device that transmitted the command frame. The status field shall be set to ACTIVE. The device shall then broadcast the route request command frame using the MCPS-DATA.request primitive.

When broadcasting a route request command frame, the NWK layer shall delay retransmission by a random jitter amount calculated using the formula:

2 x R[nwkcMinRREQJitter, nwkcMaxRREQJitter]

where is a random function on the interval. The units of this jitter amount are milliseconds. Implementers may adjust the jitter amount so that route request command frames arriving with large path cost are delayed more than frames arriving with lower path cost. The NWK layer shall retry the broadcast *nwkcRREQRetries* times after the original relay resulting in a maximum of *nwkcRREQRetries* + 1 relays per relay attempt. Implementers may choose to discard route request command frames awaiting retransmission in the case that a frame with the same source and route request identifier arrives with a lower path cost than the one awaiting retransmission.

The device shall also set the status field of the routing table entry corresponding to the routing address of the destination field in the payload to DISCOVERY_UNDERWAY. If no such entry exists, it shall be created.

[3] CCB 1487

When replying to a route request with a route reply command frame, a device that has a route discovery table entry corresponding to the source address and route request identifier of the route request shall construct a command frame with the frame type field set to 0x01. The source address field of the NWK header shall be set to the 16-bit network address of the current device and the destination address field shall be set to the value of the sender address field from the corresponding route discovery table entry. The device constructing the route reply shall populate the payload fields in the following manner.

- The NWK command identifier shall be set to route reply.
- The route request identifier field shall be set to the same value found in the route request identifier field of the route request command frame.
- The originator address field shall be set to the source address in the NWK header of the route request command frame.
- Using the sender address field from the route discovery table entry corresponding to the source address in the NWK header of the route request command frame, the device shall compute the link cost as described in section 3.6.3.1. This link cost shall be entered in the path cost field.

The route reply command frame is then unicast to the destination by using the MCPS-DATA.request primitive and the sender address obtained from the route discovery table as the next hop.

3.6.3.5.3 Upon Receipt of a Route Reply Command Frame

On receipt of a route reply command frame, a device shall perform the following procedure.

If the receiving device has no routing capacity and its NIB attribute *nwkUseTreeRouting* has a value of TRUE, it shall send the route reply as though it were a data frame being forwarded using tree routing. If the receiving device has no routing capacity and its NIB attribute *nwkUseTreeRouting* has a value of FALSE, it shall discard the command frame. Before forwarding the route reply command frame the device shall update the path cost field in the payload by computing the link cost from the next hop device to itself as described in section 3.6.3.1 and adding this to the value in the route reply path cost field.

To support legacy devices, a route reply received with a radius of 1 shall NOT be dropped. It shall continue to be processed as follows.

If the receiving device has routing capacity, it shall check whether it is the destination of the route reply command frame by comparing the contents of the originator address field of the command frame payload with its own address. If it is, it shall search its route discovery table for an entry corresponding to the route request identifier in the route reply command frame payload. If there is no such entry, the route reply command frame shall be discarded and route reply processing shall be terminated. If a route discovery table entry exists, the device shall search its routing table for an entry with a destination address field equal to the routing address corresponding to the responder address in the route reply command frame. If there is no such routing table entry, the route reply command frame shall be discarded and, if a route discovery table entry corresponding to the route request identifier in the route reply command frame exists, it shall also be removed and route reply processing shall be terminated. If a routing table entry and a route discovery table entry exist and if the status field of the routing table entry is set to DISCOVERY_UNDERWAY, it shall be changed to VALIDATION_UNDERWAY if the routing table entry's GroupId flag is TRUE or to ACTIVE otherwise; the next hop field in the routing table shall be set to the previous device that forwarded the route reply command frame. The residual cost field in the route discovery table entry shall be set to the path cost field in the route reply payload.

If the status field was already set to ACTIVE or VALIDATION_UNDERWAY, the device shall compare the path cost in the route reply command frame to the residual cost recorded in the route discovery table entry, and update the residual cost field and next hop field in the routing table entry if the cost in the route reply command frame is smaller. If the path cost in the route reply is not smaller, the route reply shall be discarded and no further processing shall take place. Note that NLDE data requests may be processed as soon as the first valid route is determined.

If the device receiving the route reply is not the destination, the device shall find the route discovery table entry corresponding to the originator address and route request identifier in the route reply command frame payload. If no such route discovery table entry exists, the route reply command frame shall be discarded. If a route discovery table entry exists, the path cost value in the route reply command frame and the residual cost field in the route discovery table entry shall be compared. If the route discovery table entry value is less than the route reply value, the route reply command frame shall be discarded.

Otherwise, the device shall find the routing table entry with a destination address field equal to the routing address corresponding to the responder address in the route reply command frame. In this case, it is an error if the route discovery table entry exists and there is no corresponding routing table entry, and the route reply command frame should be discarded. The routing table entry shall be updated by replacing the next hop field with the address of the previous device that forwarded the route reply command frame. The route discovery table entry shall also be updated by replacing the residual cost field with the value in the route reply command frame.

Whenever the receipt of a route reply causes the next hop field of the corresponding routing table entry to be modified, and the routing table entry's GroupId flag is TRUE, the device shall set the expiration time field of the corresponding route discovery table entry to expire in *nwkcWaitBeforeValidation* OctetDurations if the device is the destination of the route reply and *nwkcRouteDiscoveryTime* OctetDurations if it is not.

After updating its own route entry, the device shall forward the route reply to the destination. Before forwarding the route reply, the path cost value shall be updated. The sender shall find the next hop to the route reply’s destination by searching its route discovery table for the entry matching the route request identifier and the source address and extracting the sender address. It shall use this next hop address to compute the link cost as described in section 3.6.3.1. This cost shall be added to the path cost field in the route reply. The destination address in the command frame NWK header shall be set to the next hop address and the frame shall be unicast to the next hop device using the MCPS-DATA.request primitive. The DstAddr parameter of the MCPS-DATA.request primitive shall be set to the next-hop address from the route discovery table.

If the value of the *nwkSymLink* attribute of the NIB has a value of TRUE, the NWK layer shall, upon relaying the route reply command frame, also create a reverse routing table entry if such an entry does not yet exist. The value of the destination address field of the routing table entry shall correspond to the value of the originator address field of the route reply command frame. The status field shall have a value of ACTIVE. The next-hop address field shall have a value corresponding to the next hop address in the route reply command being relayed, as determined in the previous paragraph. If the reverse routing table entry already exists the next-hop address field shall be updated, if necessary.

3.6.3.5.4 Initiation and Processing of a Route Record Command Frame

If the NWK layer of a ZigBee router or ZigBee coordinator is initiating a unicast data frame as a result of an NLDE-DATA.request from the next higher layer and the many-to-one field of the routing table entry corresponding to the destination address of the frame has a value of TRUE, then the NWK layer shall examine the route record required field of that same routing table entry. If the route record required field also has a value of TRUE, the NWK shall unicast a route record command to the destination before transmitting the data frame.

If the NWK layer of a ZigBee router or ZigBee coordinator is forwarding a unicast data frame on behalf of one of its end device children and the many-to-one field of the destination’s routing table entry has a value of TRUE, then the device shall unicast a route record command to the destination before relaying the data frame.

An optional optimization is possible in which the router or coordinator may keep track of which of its end device children have received source routed data frames from a particular concentrator device and can thereby reduce the number of route record commands it transmits to that concentrator on behalf of its end device children.

Each relay node that receives the route record command shall append its network address to the command payload, increment the relay count, and forward the message. If no next hop is available, or if delivery to the next hop fails, or if there is insufficient space in the payload for the network address, the command frame shall be discarded and no error command shall be generated.

Upon receipt of the route record command by the destination, the route shall be stored in the source route table. Any existing source routes to the message source or intermediary nodes shall be replaced by the new route information.

3.6.3.6 Upon Expiration of a Route Discovery Table Entry

When a route discovery table entry is created, the expiration timer shall be set to expire in *nwkcRouteDiscoveryTime* OctetDurations. For entries whose GroupId flag in the corresponding entry in the routing table is TRUE, when a route reply is received that causes the next hop to change, the expiration time field of the corresponding route discovery table entry is set to expire in *nwkcWaitBeforeValidation* OctetDurations if the device is the destination of the route reply and *nwkcRouteDiscoveryTime* OctetDurations if it is not. When the timer expires, the device shall delete the entry from the route discovery table. If the device is the originator of the route request and the routing table entry corresponding to the destination address has a Status field value of VALIDATION_UNDERWAY, then the device shall transmit a message to validate the route: either the message-buffered pending route discovery or a network status command with a status code of 0x0a (validate route). If the routing table entry corresponding to the destination address has any Status field value other than ACTIVE or VALIDATION_UNDERWAY and there are no other entries in the route discovery table corresponding to that routing table entry, the routing table entry shall also be deleted.

3.6.3.7 Route Maintenance

A device NWK layer shall maintain a failure counter for each neighbor to which it has an outgoing link, *i.e.*, to which it has been required to send data frames. If the outgoing link is classified as a failed link, then the device shall respond as described in the following paragraphs. Implementers may choose a simple failure-counting scheme to generate this failure counter value or they may use a more accurate time-windowed scheme. Note that it is important not to initiate repair too frequently since repair operations may flood the network and cause other traffic disruptions. The procedure for retiring links and ceasing to keep track of their failure counter is out of the scope of this specification.

3.6.3.7.1 In Case of Link Failure

If a failed link is encountered while a device is forwarding a unicast data frame using a routing table entry with the many-to-one field set to TRUE, a network status command frame with status code of 0x0c indicating many-to-one route failure shall be generated. The destination address field in the NWK header of the network status command frame shall be equal to the destination address field in the NWK header of the frame causing the error. The destination address field of the network status command payload shall be equal to the source address field in the NWK header of the frame causing the error. The network status command frame shall be unicast to a random router neighbor using the MCPS-DATA.request primitive. Because it is a many-to-one route, all neighbors are expected to have a routing table entry to the destination. Upon receipt of the network status command frame, if no routing table entry for the destination is present, or if delivery of the network status command frame to the next hop in the routing table entry fails, the network status command frame shall again be unicast to a random router neighbor using the MCPS-DATA.request primitive. The radius counter in the NWK header will limit the maximum number of times the network status command frame is relayed. Upon receipt of the network status command frame by its destination it shall be passed up to the next higher layer using the NLME-NWK-STATUS.indication primitive. Many-to-one routes are not automatically rediscovered by the NWK layer due to route errors.

If a failed link is encountered while the device is forwarding a unicast frame using normal unicast routing, the device shall issue a network status command frame back to the source device of the frame with a status code indicating the reason for the failure (see Table 3.42), and issue an NLME-NWK-STATUS.indication to the next higher layer with a status code indicating the reason for the failure.

On receipt of a network status command frame by a router that is the intended destination of the command where the status code field of the command frame payload has a value of 0x01 or 0x02 indicating a link failure, the NWK layer will remove the routing table entry corresponding to the value of the destination address field of the command frame payload, if one exists, and inform the next higher layer of the failure using the NLME-NWK-STATUS.indication using the same status code.

On receipt of a network status command frame by a router that is the parent of an end device that is the intended destination, where the status code field of the command frame payload has a value of 0x01 or 0x02 indicating a link failure, the NWK layer will remove the routing table entry corresponding to the value of the destination address field of the command frame payload, if one exists. It will then relay the frame as usual to the end device.

On receipt of a network status command frame by an end device, the NWK layer shall inform the next higher layer of the failure using the NLME-NWK-STATUS.indication.

If an end device encounters a failed link to its parent, the end device shall inform the next higher layer using the NLME-NWK-STATUS.indication primitive with a Status parameter value of 0x09 indicating parent link failure (see Table 3.42). Similarly if a ZigBee router without routing capacity for which *nwkUseTreeRouting* has a value of TRUE encounters a failed link to its parent, it shall inform the next higher layer using the NLME-NWK-STATUS.indication primitive with a Status parameter value of 0x09 indicating parent link failure.

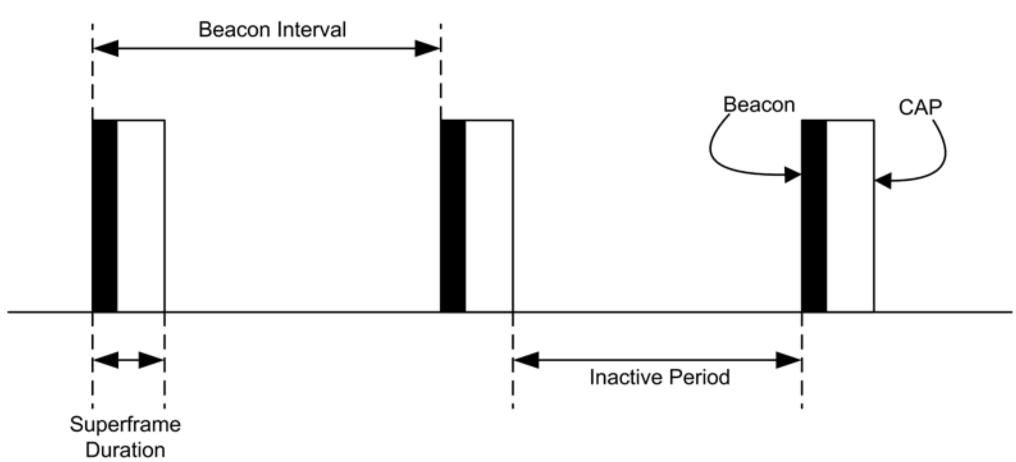
3.6.4 Scheduling Beacon Transmissions

Beacon scheduling is necessary in a multi-hop topology to prevent the beacon frames of one device from colliding with either the beacon frames or the data transmissions of its neighboring devices. Beacon scheduling is necessary when implementing a tree topology but not a mesh topology, as beaconing is not permitted in ZigBee mesh networks.

3.6.4.1 Scheduling Method

The ZigBee coordinator shall determine the beacon order and superframe order for every device in the network (see [B1] for more information on these attributes). Because one purpose of multi-hop beaconing networks is to allow routing nodes the opportunity to sleep in order to conserve power, the beacon order shall be set much larger than the superframe order. Setting the attributes in this manner makes it possible to schedule the active portion of the superframes of every device in any neighborhood such that they are non-overlapping in time. In other words, time is divided into approximately (*macBeaconInterval*/*macSuperframeDuration*) non-overlapping time slots, and the active portion of the superframe of every device in the network shall occupy one of these non-overlapping time slots. An example of the resulting frame structure for a single beaconing device is shown in Figure 3.48.

Figure 3.48 Typical Frame Structure for a Beaconing Device



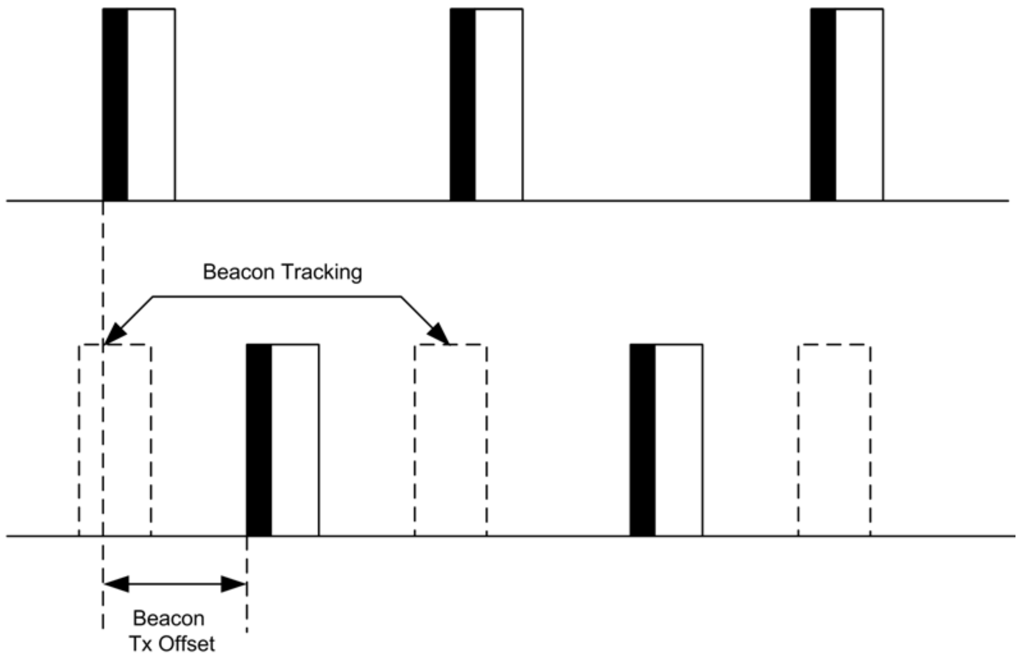
The beacon frame of a device shall be transmitted at the start of its non-overlapping time slot, and the transmit time shall be measured relative to the beacon transmit time of the parent device. This time offset shall be included in the beacon payload of every device in a multi-hop beaconing network (see section 3.6.7 for a complete list of beacon payload parameters). Therefore a device receiving a beacon frame shall know the beacon transmission time of both the neighboring device and the parent of the neighboring device, since the transmission time of the parent may be calculated by subtracting the time offset from the timestamp of the beacon frame. The receiving device shall store both the local timestamp of the beacon frame and the offset included in the beacon payload in its neighbor table. The purpose of having a device know when the parent of its neighbor is active is to maintain the integrity of the parent-child communication link by alleviating the hidden node problem. In other words, a device will never transmit at the same time as the parent of its neighbor.

Communication in a tree network shall be accomplished using the parent-child links to route along the tree. Since every child tracks the beacon of its parent, transmissions from a parent to its child shall be completed using the indirect transmission technique. Transmissions from a child to its parent shall be completed during the CAP of the parent. Details for the communication procedures can be found in IEEE 802.15.4-2003 [B1].

A new device wishing to join the network shall follow the procedure outlined in section 3.6.1.4. In the process of joining the network, the new device shall build its neighbor table based on the information collected during the MAC scan procedure. Using this information, the new device shall choose an appropriate time for its beacon transmission and CAP (the active portion of its superframe structure) such that the active portion of its superframe structure does not overlap with that of any neighbor or of the parent of any neighbor. If there is no available non-overlapping time slot in the neighborhood, the device shall not transmit beacons and shall operate on the network as an end device. If a non-overlapping time slot is available, the time offset between the beacon frames of the parent and the new device shall be chosen and included in the beacon payload of the new device. Any algorithm for selecting the beacon transmission time that avoids beacon transmission during the active portion of the superframes of its neighbors and their parents may be employed, as interoperability will be ensured.

To counteract drift, the new device shall track the beacon of its parent and adjust its own beacon transmission time such that the time offset between the two remains constant. Therefore, the beacon frames of every device in the network are essentially synchronized with those of the ZigBee coordinator. Figure 3.49 illustrates the relationship between the active superframe portions of a parent and its child.

Figure 3.49 Parent-Child Superframe Positioning Relationship



The density of devices that can be supported in the network is inversely proportional to the ratio of the superframe order to the beacon order. The smaller the ratio, the longer the inactive period of each device and the more devices that can transmit beacon frames in the same neighborhood. It is recommended that a tree network utilize a superframe order of 0, which, when operating in the 2.4 GHz band, gives a superframe duration of 15.36 ms and a beacon order of between 6 and 10, which, in the 2.4 GHz band, gives a beacon interval between 0.98304s and 15.72864s. Using these superframe and beacon order values, a typical duty cycle for devices in the network will be between ~2% and ~0.1% regardless of the frequency band.

3.6.5 Broadcast Communication

This section specifies how a broadcast transmission is accomplished within a ZigBee network. Any device within a network may initiate a broadcast transmission intended for a number of other devices that are part of the same network. A broadcast transmission is initiated by the local APS sub-layer entity through the use of the NLDE-DATA.request primitive by setting the DstAddr parameter to a broadcast address as shown in Table 3.59, or by the NWK layer through the use of these same broadcast addresses in the construction of an outgoing NWK header. (Note that broadcast transmission for link status and route request command frames is handled differently as described in section 3.6.3.4 and section 3.6.3.5.2 respectively.)

Table 3.59 Broadcast Addresses

Broadcast Address	Destination Group
0xffff	All devices in PAN
0xfffe	Reserved
0xfffd	*macRxOnWhenIdle* = TRUE
0xfffc	All routers and coordinator

Broadcast Address	Destination Group
0xfffb	Low power routers only
0xfff8 - 0xfffa	Reserved

To transmit a broadcast MSDU, the NWK layer of a ZigBee router or ZigBee coordinator issues an MCPS-DATA.request primitive to the MAC sub-layer with the DstAddrMode parameter set to 0x02 (16-bit network address) and the DstAddr parameter set to 0xffff. For a ZigBee end device, the MAC destination address of the broadcast frame shall be set equal to the 16-bit network address of the parent of the end device. The PANId parameter shall be set to the PANId of the ZigBee network. This specification does not support broadcasting across multiple networks. Broadcast transmissions shall not use the MAC sub-layer acknowledgement; instead, a passive acknowledgement mechanism may be used. Passive acknowledgement means that every ZigBee router and ZigBee coordinator keeps track of which of its neighboring devices have successfully relayed the broadcast transmission. The MAC sub-layer acknowledgement is disabled by setting the acknowledged transmission flag of the TxOptions parameter to FALSE. All other flags of the TxOptions parameter shall be set based on the network configuration.

The ZigBee coordinator, each ZigBee router and those ZigBee end devices with *macRxOnWhenIdle* equal to TRUE, shall keep a record of any new broadcast transaction that is either initiated locally or received from a neighboring device. This record is called the broadcast transaction record (BTR) and shall contain at least the sequence number and the source address of the broadcast frame. The broadcast transaction records are stored in the *nwkBroadcastTransactionTable* (BTT) as shown in Table 3.60.

Table 3.60 Broadcast Transaction Record

Field Name	Size	Description
Source Address	2 bytes	The 16-bit network address of the broadcast initiator.
Sequence Number	1 byte	The NWK layer sequence number of the initiator's broadcast.
Expiration Time	1 byte	A countdown timer indicating the number of seconds until this entry expires; the initial value is *nwkNetworkBroadcastDeliveryTime*.

When a device receives a broadcast frame from a neighboring device, it shall compare the destination address of the frame with its device type. If the destination address does not correspond to the device type of the receiver as outlined in Table 3.59, the frame shall be discarded. If the destination address corresponds to the device type of the receiver, the device shall compare the sequence number and the source address of the broadcast frame with the records in its BTT.

If the device has a BTR of this particular broadcast frame in its BTT, it may update the BTR to mark the neighboring device as having relayed the broadcast frame. It shall then drop the frame. If no record is found, it shall create a new BTR in its BTT and may mark the neighboring device as having relayed the broadcast. The NWK layer shall then indicate to the higher layer that a new broadcast frame has been received using the NLDE-DATA.indication. If the device is a ZigBee router (ZR) or a ZigBee Coordinator (ZC) and the radius field is greater than zero; then the frame shall be retransmitted. Otherwise it shall be dropped. Before the retransmission, it shall wait for a random time period called broadcast jitter. This time period shall be bounded by the value of the *nwkcMaxBroadcastJitter* attribute. ZigBee end devices with *macRxOnWhenIdle* equal to FALSE shall not participate in the relaying of broadcast frames and need not maintain a BTT for broadcast frames that they originate.

If, on receipt of a broadcast frame, the NWK layer finds that the BTT is full and contains no expired entries, then the frame should be dropped. In this situation the frame should not be retransmitted, nor should it be passed up to the next higher layer.

A ZigBee coordinator or ZigBee router operating in a non-beacon-enabled ZigBee network shall retransmit a previously broadcast frame at most *nwkMaxBroadcastRetries* times. If the device does not support passive acknowledgement, then it shall retransmit the frame exactly *nwkMaxBroadcastRetries* times. If the device supports passive acknowledgement and any of its neighboring devices have not relayed the broadcast frame within *nwkPassiveAckTimeout* OctetDurations then it shall continue to retransmit the frame up to a maximum of *nwkMaxBroadcastRetries* times.

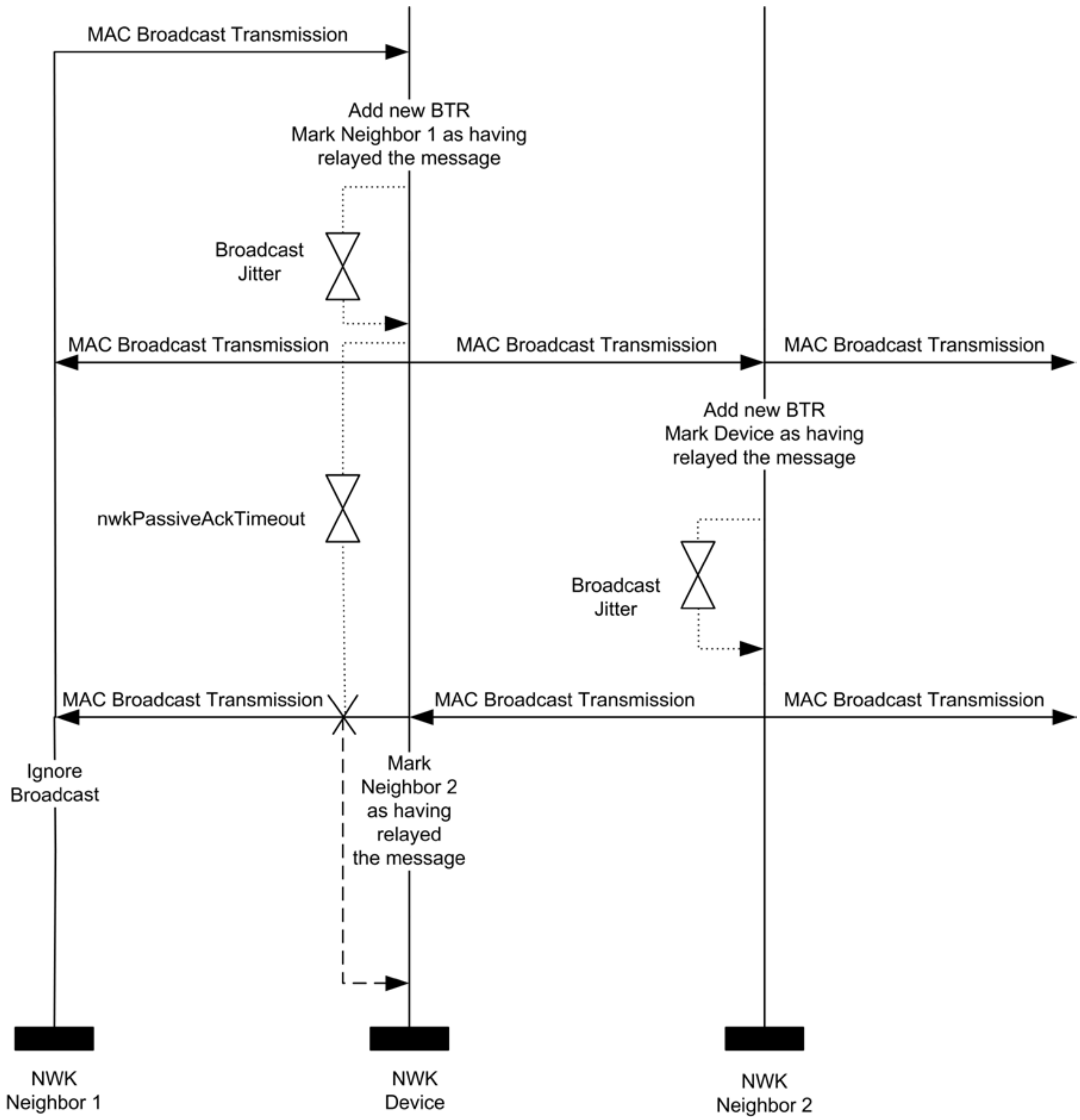
A device should change the status of a BTT entry after *nwkNetworkBroadcastDeliveryTime* OctetDurations have elapsed since its creation. The entry status should change to expired and thus the entry can be overwritten if required when a new broadcast is received.

When a ZigBee router that has the *macRxOnWhenIdle* MAC PIB attribute set to FALSE receives a broadcast transmission, it shall use a different procedure for retransmission than the one outlined above. It shall retransmit the frame without delay to each of its neighbors individually, using a MAC layer unicast, that is, with the DstAddr parameter of the MCPS-DATA.request primitive set to the address of each neighbor device and not to the broadcast address. Similarly, a router or coordinator with the *macRxOnWhenIdle* MAC PIB attribute set to TRUE, which has one or more neighbors with the *macRxOnWhenIdle* MAC PIB attribute set to FALSE, shall, in the case where the destination address is 0xffff denoting broadcast to all devices, retransmit the broadcast frame to each of these neighbors in turn as a MAC layer unicast in addition to performing the more general broadcast procedure spelled out in the previous paragraphs. Indirect transmission, as described in IEEE 802.15.4-2003 [B1], may be employed to ensure that these unicasts reach their destination.

Every ZigBee router shall have the ability to buffer at least 1 frame at the NWK layer in order to facilitate retransmission of broadcasts.

Figure 3.50 shows a broadcast transaction between a device and two neighboring devices**.**

Figure 3.50 Broadcast Transaction Message Sequence Chart



3.6.6 Multicast Communication

This section specifies how multicast transmission is accomplished within a ZigBee network. Multicast addressing is accomplished using 16-bit multicast group IDs. A multicast group is a collection of nodes, all registered under the same multicast group ID, that are physically separated by a hop distance of no more than a given radius, known as the MaxNonMemberRadius. A multicast message is sent to a particular destination group and is received by all members of that group. Only data frames are multicast — no NWK command frames are multicast.

Multicast frames are propagated through the network by both members and non-members of the destination multicast group. A packet may be sent in one of two modes as indicated by a mode flag in the packet which determines the method of relay to the next hop. If the original message was created by a member of the group, it is considered to be in 'Member Mode' and is relayed by means of broadcasts. If the original message was created by a non-member of the group, it is considered to be in 'Non-Member Mode' and is relayed by means of unicasts towards a group member. Once a non-member message reaches any member of the destination group, it is instantly transformed into a Member Mode type relay for the duration of the life of the packet regardless of who relays it next.

Multicast messages may be originated by end devices but are not sent to devices where *macRxOnWhenIdle* is equal to FALSE.

3.6.6.1 The Group ID Table

The NWK layer of a device may maintain a group ID table, *nwkGroupIDTable*, accessible as an attribute of the NIB as shown in Table 3.49. If the *nwkGroupIDTable* NIB attribute is present then it shall contain a set of 16-bit group identifiers for groups of which the device is a member.

Note that the optional *nwkGroupIDTable* NIB attribute has a functional overlap with the mandatory APS group table (see Table 2-18). If a device maintains both tables, and thereby expects to use NWK-layer multicast as a method for receiving group-addressed frames, it must assure that each 16-bit group identifiers that appears in the APS group table also appears in the NWK group table.

Note also that from an implementation perspective, it would be wasteful to duplicate the list of group identifiers across layers and it is assumed that implementers will find a way to combine the APS and NWK group tables to avoid waste.

3.6.6.2 Upon Receipt of a Multicast Frame from the Next Higher Layer

If an NLDE-DATA.request is received by the NWK layer from its next higher layer and the multicast control field is 0x01, the NWK layer shall determine whether an entry exists in the *nwkGroupIDTable* having a group identifier field matching the destination address of the frame. If a matching entry is found, the NWK layer shall multicast the frame according to the procedure outlined in section 3.6.6.2.1. If a matching entry is not found, the frame shall be initiated as a non-member mode multicast using the procedure outlined in section 3.6.6.2.2.

3.6.6.2.1 Initiating a Member Mode Multicast

The NWK layer shall set the multicast mode sub-field of the multicast control field to 0x01 (member mode). If the BTT table is full and contains no expired entries, the message shall not be sent and the NLDE shall issue the NLDE-DATA.confirm primitive with a status value of BT_TABLE_FULL. If the BTT is not full or contains an expired BTR, a new BTR shall be created with the local node as the source and the multicast frame's sequence number. The message shall then be transmitted according to the procedure described in the final paragraph of section 3.6.6.3.

3.6.6.2.2 Initiating a Non-Member Mode Multicast

The NWK layer shall set the multicast mode sub-field of the multicast control field to 0x00 (non-member mode). Then, the NWK layer shall check its routing table for an entry corresponding to the GroupID destination of the frame. If there is such an entry, the NWK layer shall examine the entry's status field. If the status is ACTIVE, then the device shall (re)transmit the frame. If the status is VALIDATION_UNDERWAY, then the status shall be changed to ACTIVE, the device shall transmit the frame according to the procedure described in the final paragraph of section 3.6.6.4, and the NLDE shall issue the NLDE-DATA.confirm primitive with the status value received from the MCPS-DATA.confirm primitive. If there is no routing table entry corresponding to the GroupID destination of the frame and the value of the DiscoverRoute parameter is 0x00 (suppress route discovery), the frame shall be discarded and the NLDE shall issue the NLDE-DATA.confirm primitive with a status value of ROUTE_DISCOVERY_FAILED. If the DiscoverRoute parameter has a value of 0x01 (enable route discovery) and there is no routing table entry corresponding to the GroupID destination of the frame, then the device shall initiate route discovery immediately as described in section 3.6.3.5.1. The frame may optionally be buffered pending route discovery. If it is not buffered, the frame shall be discarded and the NLDE shall issue the NLDE-DATA.confirm primitive with a status value of FRAME_NOT_BUFFERED.

3.6.6.3 Upon Receipt of a Member Mode Multicast Frame

When a device receives a member mode multicast frame from a neighboring device, it shall compare the sequence number and the source address of the multicast frame with the records in its BTT. If the device has a BTR of this particular multicast frame in its BTT it shall discard the frame. If no record is found and the BTT is full and contains no expired entries, it shall discard the frame. If no record is found and the BTT is not full or contains an expired BTR, it shall create a new BTR and continue processing the message as outlined in the following paragraph.

When a member mode multicast frame has been received from a neighbor and added to the BTT, the NWK layer shall then determine whether an entry exists in the *nwkGroupIDTable* whose group identifier field matches the destination group ID of the frame. If a matching entry is found, the message shall be passed to the next higher layer, the multicast mode sub-field of the multicast control field shall be set to 0x01 (member mode), the value of the NonmemberRadius sub-field shall be set to the value of the MaxNonmemberRadius sub-field in the multicast control field, and the message shall be transmitted as outlined in the following paragraph.

If a matching entry is not found, the NWK layer shall examine the frame's multicast NonmemberRadius field. If the value of the NonmemberRadius sub-field of the multicast field is 0 the message shall be discarded, along with the newly added BTR. Otherwise, the NonmemberRadius sub-field shall be decremented if it is less than 0x07 and the frame shall be transmitted as outlined in following paragraphs. If, as a result of being decremented, this value falls to 0, the frame shall not, under any circumstances, be retransmitted.

Each member mode multicast message shall be transmitted *nwkMaxBroadcastRetries* times. For member mode multicast frames that did not originate on the local device, the initial transmission shall be delayed by a random time bounded by the value of the *nwkcMaxBroadcastJitter* attribute. A device shall delay a period of *nwkPassiveAckTimeout* OctetDurations between retransmissions of a particular member mode multicast message. Unlike broadcasts, there is no passive acknowledgement for multicasts. ZigBee end devices shall not participate in the relaying of multicast frames.

To transmit a member mode multicast MSDU, the NWK layer issues an MCPS-DATA.request primitive to the MAC sub-layer with the DstAddrMode parameter set to 0x02 (16-bit network address) and the DstAddr parameter set to 0xffff, which is the broadcast network address. The PANId parameter shall be set to the PANId of the ZigBee network. Member mode multicast transmissions shall not use the MAC sub-layer acknowledgement or the passive acknowledgement used for broadcasts. The MAC sub-layer acknowledgement is disabled by setting the acknowledged transmission flag of the TxOptions parameter to FALSE. All other flags of the TxOptions parameter shall be set based on the network configuration.

3.6.6.4 Upon Receipt of a Non-Member Mode Multicast Frame

When a device receives a non-member mode multicast frame from a neighboring device, the NWK layer shall determine whether an entry exists in the *nwkGroupIDTable* having a group identifier field that matches the destination address of the frame. If a matching entry is found, the multicast control field shall be set to 0x01 (member mode) and the message shall be processed as if it had been received as a member mode multicast. If no matching *nwkGroupIDTable* entry is found, the device shall check its routing table for an entry corresponding to the GroupID destination of the frame. If there is no such routing table entry, the message shall be discarded. If there is such an entry, the NWK layer shall examine the entry's status field. If the status is ACTIVE, the device shall (re)transmit the frame. If the status is VALIDATION_UNDERWAY, the status shall be changed to ACTIVE and the device shall (re)transmit the frame. To transmit a non-member mode multicast MSDU, the NWK layer issues an MCPS-DATA.request primitive to the MAC sublayer with the DstAddrMode parameter set to 0x02 (16-bit network address) and the DstAddr parameter set to the next hop as determined from the matching routing table entry. The PANId parameter shall be set to the PANId of the ZigBee network. The MAC sub-layer acknowledgement shall be enabled by setting the acknowledged transmission flag of the TxOptions parameter to TRUE. All other flags of the TxOptions parameter shall be set based on the network configuration.

3.6.7 NWK Information in the MAC Beacons

This section specifies how the NWK layer uses the beacon payload of a MAC sub-layer beacon frame to convey NWK layer-specific information to neighboring devices.

The beacon payload shall contain the information shown in Table 3.61. This enables the NWK layer to provide additional information to new devices that are performing network discovery and allows these new devices to more efficiently select a network and a particular neighbor to join. Refer to section 3.6.1.4.1.1 for a detailed description of the network discovery procedure.

Table 3.61 NWK Layer Information Fields

Name	Type	Valid Range	Description
Protocol ID	Integer	0x00 – 0xff	This field identifies the network layer protocols in use and, for purposes of this specification, shall always be set to 0, indicating the ZigBee protocols. The value 0xff shall also be reserved for future use by the ZigBee Alliance.
Stack profile	Integer	0x00 – 0x0f	A ZigBee stack profile identifier.
nwkcProtocolVersion	Integer	0x00 – 0x0f	The version of the ZigBee protocol.
Router capacity	Boolean	TRUE or FALSE	This value is set to TRUE if this device is capable of accepting join requests from router-capable devices and is set to FALSE otherwise.

Name	**Type**	**Valid Range**	**Description**
Device depth	Integer	0x00 – 0x0f	The network depth of this device. A value of 0x00 indicates that this device is the ZigBee coordinator for the network.
End device capacity	Boolean	TRUE or FALSE	This value is set to TRUE if the device is capable of accepting join requests from end devices seeking to join the network and is set to FALSE otherwise.
nwkExtendedPANId	64-bit extended address	0x0000000000000001 – 0xfffffffffffffffe	The globally unique ID for the PAN of which the beaconing device is a member. By default, this is the 64-bit IEEE address of the ZigBee coordinator that formed the network, but other values are possible and there is no required structure to the address.
TxOffset	Integer	0x000000 – 0xffffff	This value indicates the difference in time, measured in symbols, between the beacon transmission time of the device and the beacon transmission time of its parent; This offset may be subtracted from the beacon transmission time of the device to calculate the beacon transmission time of the parent. This parameter is set to the default value of 0xFFFFFF in beaconless networks.
nwkUpdateId	Integer	0x00 - 0xFF	This field reflects the value of *nwkUpdateId* from the NIB.

The NWK layer of the ZigBee coordinator shall update the beacon payload immediately following network formation. All other ZigBee devices shall update it immediately after the join is completed and any time the network configuration (any of the parameters specified in Table 3.10) changes. The beacon payload is written into the MAC sub-layer PIB using the MLME-SET.request primitive. The *macBeaconPayloadLength* attribute is set to the length of the beacon payload, and the octet sequence representing the beacon payload is written into the *macBeaconPayload* attribute. The formatting of the bit sequence representing the beacon payload is shown in Figure 3.50.

Figure 3.51 Format of the MAC Sub-Layer Beacon Payload

Bits: 0–7	**8–11**	**12–15**	**16–17**	**18**	**19–22**	**23**	**24–87**	**88–111**	**112–119**
Protocol ID	Stack profile	*nwk cProtocol Version*	Re-served	Router capacity	Device depth	End de-vice ca-pacity	*nwk Extended PANId*	Tx Offset	*Nwk UpdateId*

3.6.8 Persistent Data

Devices operating in the field may be restarted either manually or programmatically by maintenance personnel, or may be restarted accidentally for any number of reasons, including localized or network-wide power failures, battery replacement during the course of normal maintenance, impact, and so on. The following information should be preserved across resets in order to maintain an operating network:

- The device's PAN Id and Extended PAN Id.
- The device's 16-bit network address.
- If *nwkAddrAlloc* is equal to 0, a device shall save the following information for each associated router child in the neighbor table:
  - The 64-bit IEEE address
  - 16-bit network address
- For each device in the nwkNeighborTable of the NIB with a device type set to 0x02 (ZigBee End Device), the following shall be saved:
  - The 64-bit IEEE address
  - 16-bit network address
  - The End Device Configuration value
  - Device Timeout value
- If the device is an end device, the *nwkParentInformation* value in the NIB.
- For end devices, the 16-bit network address of the parent device.
- The stack profile in use.
- The device depth.

The method by which these data are made to persist is beyond the scope of this specification.

3.6.9 Low Power Routers (LPR)

Low power routers are defined as routers operating on batteries for multiple years by regularly powering off their radios. LPRs shall be recognized by high power routers (HPR) looking at the following capability information bit-fields (see Table 3.52) during the joining phase:

- Device type set to 1
- Receive on when idle set to FALSE

LPR devices should be able to receive network command frames that are broadcast in the network. This can be achieved by setting the destination address in the NWK header to the broadcast address for all routers and coordinators (see Table 3.59).

3.6.10 End Device Aging and Management

The end device and router relationship is established via MAC association or NWK rejoin, and can be dissolved via a leave command. However there are a number of ways in which the relationship can get broken, where router parent and end device do not agree. For example the router parent may think it is still the router parent for an end device when in fact the end device has switched to a new parent, or the router parent may age out the child since it has had no communication with it for an extended period of time.

Router parents have a finite amount of local resources to store end device information. As such it is desirable to clean out old entries to allow for new end devices to join. End devices shall be aged out by the router according to the rules defined below.

3.6.10.1 End Device Aging Mechanism

A router parent must age neighbor table entries for end devices. It is important to note that prior versions of this specification did not have this requirement and thus legacy devices exist that do not have this child aging mechanism.

A router parent shall keep track of the amount of real time that has passed and decrement the Timeout counter value for each end device entry in its neighbor table until the value reaches 0. When a neighbor table entry's Timeout counter value reaches 0, the router parent shall delete the entry from the neighbor table.

End Devices may periodically send a keepalive message to reset the Timeout counter value. See section 3.6.10.3 for details.

3.6.10.2 Establishing the Timeout

A router shall initially set the timeout for all end devices according to the default value of *nwkEndDeviceTimeoutDefault* in Table 3.49. The following describes how an end device may update this value from the default.

After joining or rejoining the network the end device shall send an End Device Timeout Request command to its parent. This shall be done even if the end device is joining or rejoining to the same parent. The message shall include their timeout period and configuration.

Routers shall process the End Device Timeout Request command as follows.

1. If the Requested Timeout Enumeration value in the frame is not within the valid range, it shall generate an End Device Timeout Response command with a status of INCORRECT_VALUE and no further processing of the message shall take place.
2. The parent shall find the neighbor table entry for the sending device and verify that the entry corresponds to an end device. If no entry is found or the entry is not an end device, then the message shall be dropped and no further processing should take place.
3. The received value shall be converted into an actual timeout amount. This shall be done by obtaining the actual timeout value for the corresponding Requested Timeout Enumeration in Table 3.44. The value shall be converted from minutes into seconds if it is not already a value in seconds. The parent shall set the Timeout Counter and Device Timeout values of the neighbor table entry to the converted value.
4. The parent shall set the End Device Configuration information in the neighbor table for the corresponding end device's entry to the value of the End Device Configuration field in the received message.
5. The parent shall generate an End Device Timeout Response command with a status of SUCCESS. It shall fill in the value of the *Parent Information Bitmask* field according to the keepalive methods it supports.

An End Device that receives an End Device Timeout Response Command shall process it as follows.

1. If the status is SUCCESS it shall set the *nwkParentInformation* value in the NIB to value of the Parent Information field of the received command. No further processing shall take place.

2. If the End Device receives the command with a status value other than SUCCESS, it shall assume its timeout value has not been configured on the parent.

End Devices may receive no End Device Timeout Response command at all if they are communicating with a legacy device that does not have support for this command. They shall treat this the same as receiving an End Device Timeout Response with a non-SUCCESS status code.

3.6.10.3 End Device Keepalive

All end devices (including RxOnWhenIdle=TRUE) that have received an End Device Timeout Response Command with a status of SUCCESS may periodically send a keepalive to their router parent to insure they remain in the router's neighbor table.

The keepalive message will refresh the timeout on the parent device so that the parent does not delete the child from its neighbor table. The period for sending the keepalive to the router parent shall be determined by the manufacturer of the device and is not specified by this standard. It is recommended that the period allows the end device to send 3 keepalive messages during the Device Timeout period. This will help insure that a single missed keepalive message will not age out the end device on the router parent.

There are two keepalive mechanisms described below. The method the end device uses depends on the support of the router parent. The router parent will indicate its support in the End Device Timeout Response command frame and this information will be stored in the NIB.

When an End Device needs to send a keepalive message, it shall examine the *nwkParentInformation* value in the NIB. If bit 0 has a value of 1 (indicating support of the MAC data poll keepalive) then the device shall send a MAC data poll command unicast to its parent.

Otherwise if the value of bit 1 has a value of 1, then the device shall send an End Device Timeout Request command as a unicast to refresh the keepalive timer. If the transmission is successful, the device shall wait for macResponseWaitTime for an End Device Timeout Response from its parent. If the transmission was unsuccessful, or if no End Device Timeout Response command is received, or if the status field indicates a value other than SUCCESS, the end device shall generate a NLME-NWK-STATUS.indication with a code of 0x09 (Parent Link Failure).

3.6.10.4 MAC Data Poll Processing

A router whose *nwkParentInformation* in the NIB has bit 1 set to 0, shall support the MAC Data poll as an End Device keepalive. A router is not required to support this method. If it does not it must support the End Device Timeout Request method.

Upon receipt of an MLME-POLL.Indication the router parent shall examine its neighbor table and do **one** of the following:

1. If there is no entry in the neighbor table corresponding to the DeviceAddress of the MLME-Poll.Indication primitive, then the device shall construct a leave message. The destination NWK address shall be set to the value of the MAC source of the MAC data poll. See section 3.6.10.4.1 for more information on the leave message. The message shall be added to the indirect transaction queue of the MAC layer.

2. If there is an entry in the neighbor table for the sending device's MAC source, then the local device shall set the Timeout counter value to the value of the *End Device Keepalive Timeout* value, and it shall set the Keepalive Received value to TRUE.

When an End Device sends a MAC Data poll command it shall assume that the parent has knowledge of the end device and the Timeout Counter associated with the end device has been reset in the parent's neighbor table. The End Device will behave per reference [B1] with regard to the data pending bit in the MAC Ack, and will follow standard processing of any leave message that may be received after sending a data poll.

3.6.10.4.1 Sending a Leave Message

A router shall send a leave message when it wants to inform an end device it is no longer a parent to the end device. The leave message shall be one of the following messages:

1. NWK Leave Request
    a. A device that chooses to send a NWK leave request shall set fields of the NWK Command as follows.
        i. The destination IEEE address sub-field of the frame control shall be set to 0, indicating that no destination IEEE address is present.
        ii. The destination IEEE address field shall not be present in the message.
        iii. The request sub-field of the command options field shall be set to 1.
        iv. The rejoin request sub-field of the command shall be set to 1.
2. ZDO Mgmt_Leave_Req
    a. A device that chooses to send a ZDO Mgmt_Leave_Req shall set the fields of the of the ZDO Mgmt_leave_req command as follows:
        i. The Device Address field shall be set to NULL (0x0000000000000000)
        ii. The Remove Children Bit shall be set to 0.
        iii. The Rejoin bit shall be set to 1.
    b. The Acknowledgement request sub-field of the APS Frame control field shall be set to 0 (no acknowledgement requested).

3.6.10.5 Setting the End Device Timeout on the Router Parent

A router shall set the default values for Timeout Counter and End Device Keepalive Timeout to the time-span indicated by *nwkEndDeviceTimeoutDefault* as converted to seconds.

After successfully joining or rejoining the network and receiving the network key, an End Device shall send an End Device Timeout Request command to its router parent indicating its desired timeout. Upon receipt and successful processing of the End Device Timeout Request router parents shall update the timeout values accordingly. See section 3.6.10.2 for details.

Legacy devices will not send an End Device Timeout Request and thus will receive the default timeout.

3.6.10.6 Local End Device Timeout

An end device may keep track of its timeout using the following mechanism:

1. The end device shall find the corresponding neighbor table entry for its router parent.
2. It shall decrement the Timeout Counter value in the Neighbor Table entry based on the amount of real time that has passed, until that value reaches 0.
3. If the Timeout Counter reaches a value of 0, it shall assume that its parent has timed out the device.

If the end device has determined that it has been timed out, it can choose to perform a rejoin to get back on the network as described in section 3.6.1.4.2. Alternatively it is permissive for an end device to always perform a rejoin without keep tracking of its local end device timeout.

There is no requirement that the end device re-establish connectivity with the network if it has determined that it has reached the timeout value established with its router parent. An end device may choose to delay rejoining the network until it is appropriate, for example when the end device has data it needs to send.

3.6.10.7 Persistent Values on the Parent Router

The router parent is expected to persistently store the end device information in the neighbor table (see section 3.6.8).

3.6.10.8 Reboot and Child Aging

On reboot routers shall set the Timeout Counter value for each end device in its neighbor table to the entry's value of Device Timeout. In other words, end devices shall be given a full time period for aging out.

On reboot it is recommended end devices immediately initiate a keep-alive message to verify connectivity to their parent.

3.6.10.9 Diagrams Illustrating End Device Management

Figure 3.52 Initial Setup of the End Device Timeout

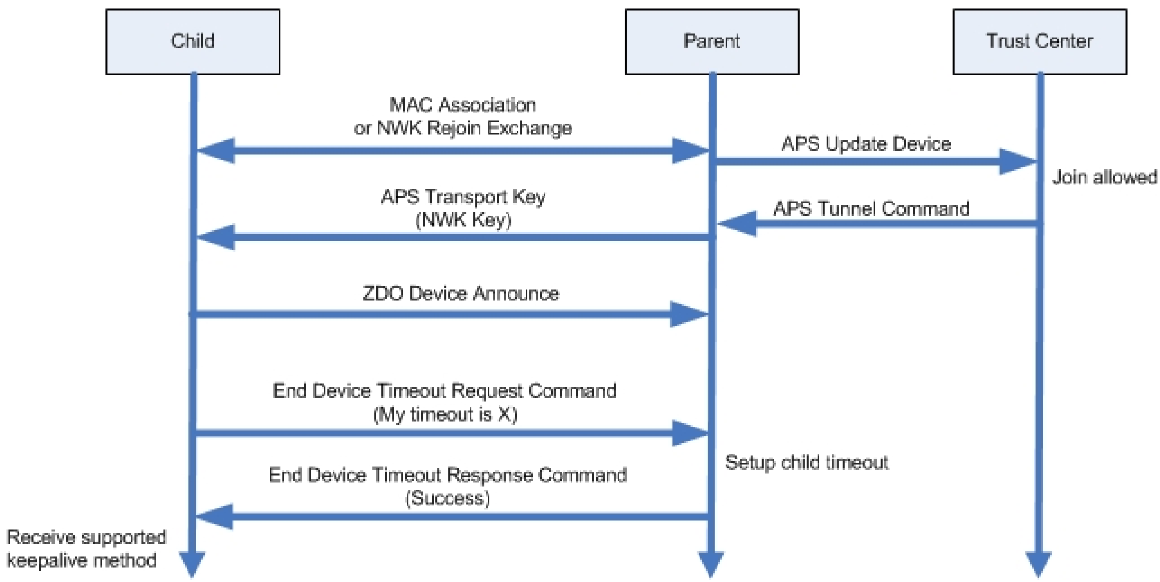


Figure 3.52 shows an end device joining into a network and the series of message exchanges. After the end device has joined and has a copy of the NWK key, it will send a NWK command of End Device Request to the parent and check for a response.

Figure 3.53 Child Keepalive: MAC Data Poll Method

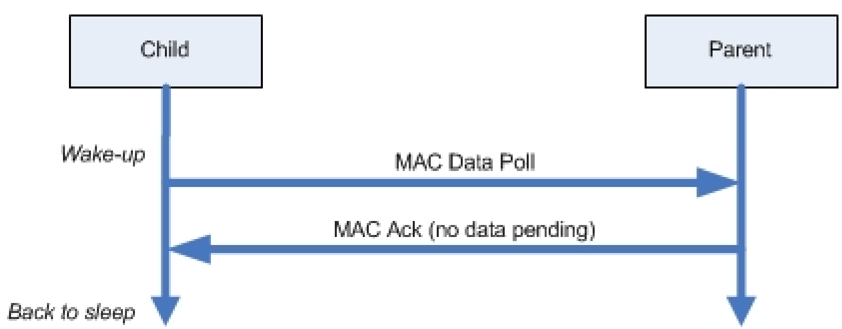


Figure 3.53 shows normal operation of a child talking to a parent that supports the MAC Data Poll Keepalive Method. When the data pending bit is unset in the MAC acknowledgement, the end device can assume that the parent still remembers the device.

Figure 3.54 Child Keepalive: End Device Timeout Request Method

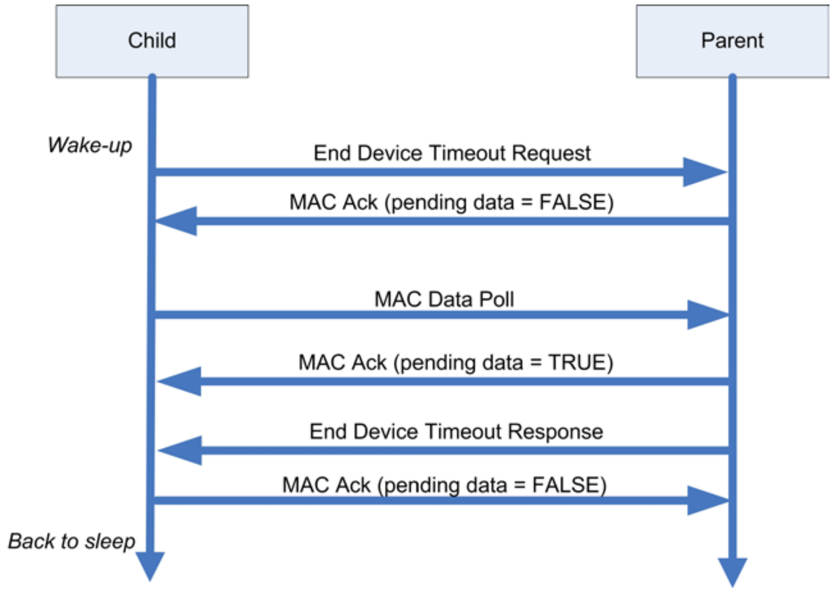
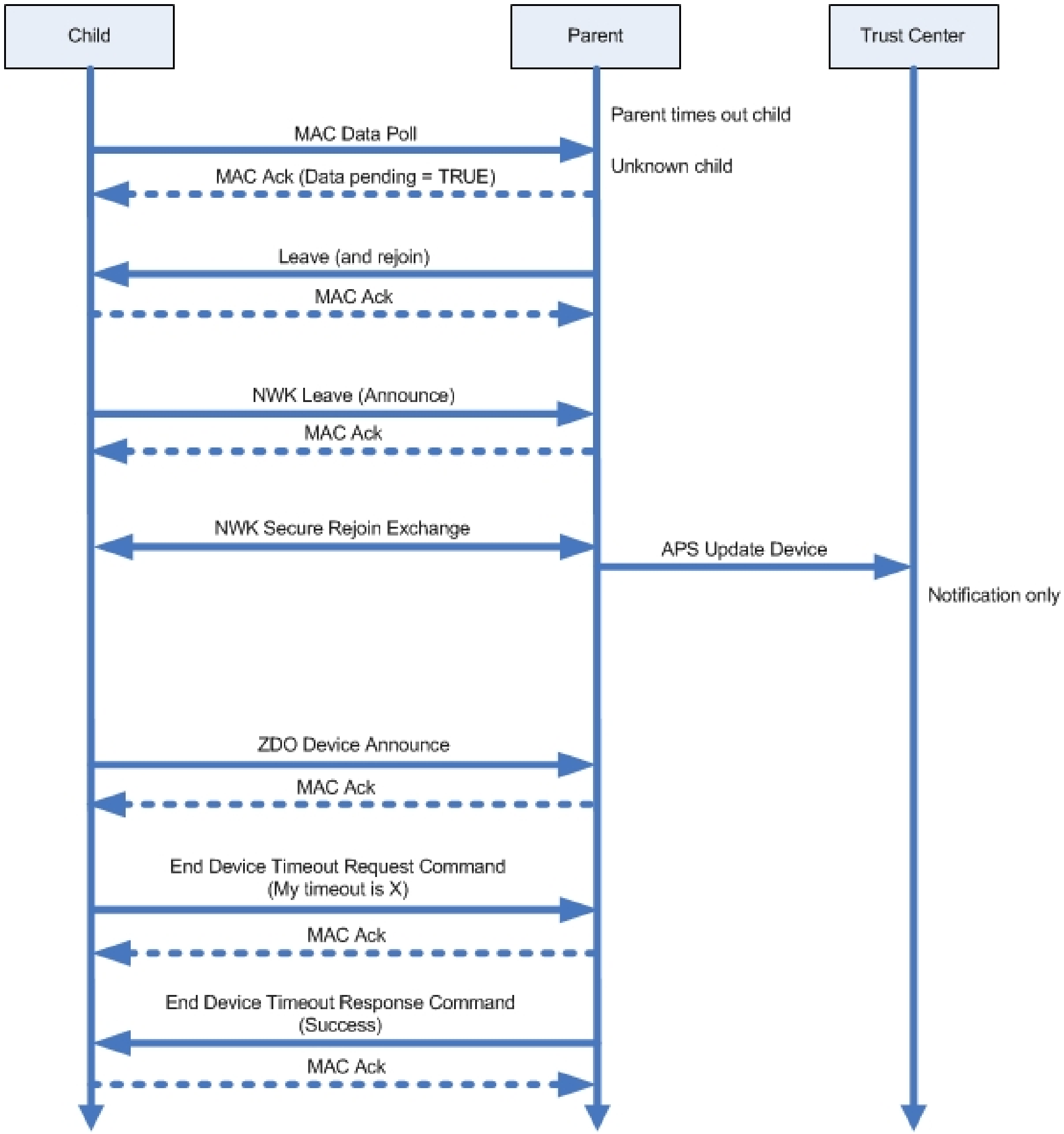


Figure 3.54 shows normal operation of a child talking to a parent that supports the End Device Timeout Request keepalive method. T.

9765 **Figure 3.55 Aging out Children: MAC Data Poll Method - Secure Rejoin**



9766

9767

Figure 3.56 Aging out Children: MAC Data Poll - Trust Center Rejoin

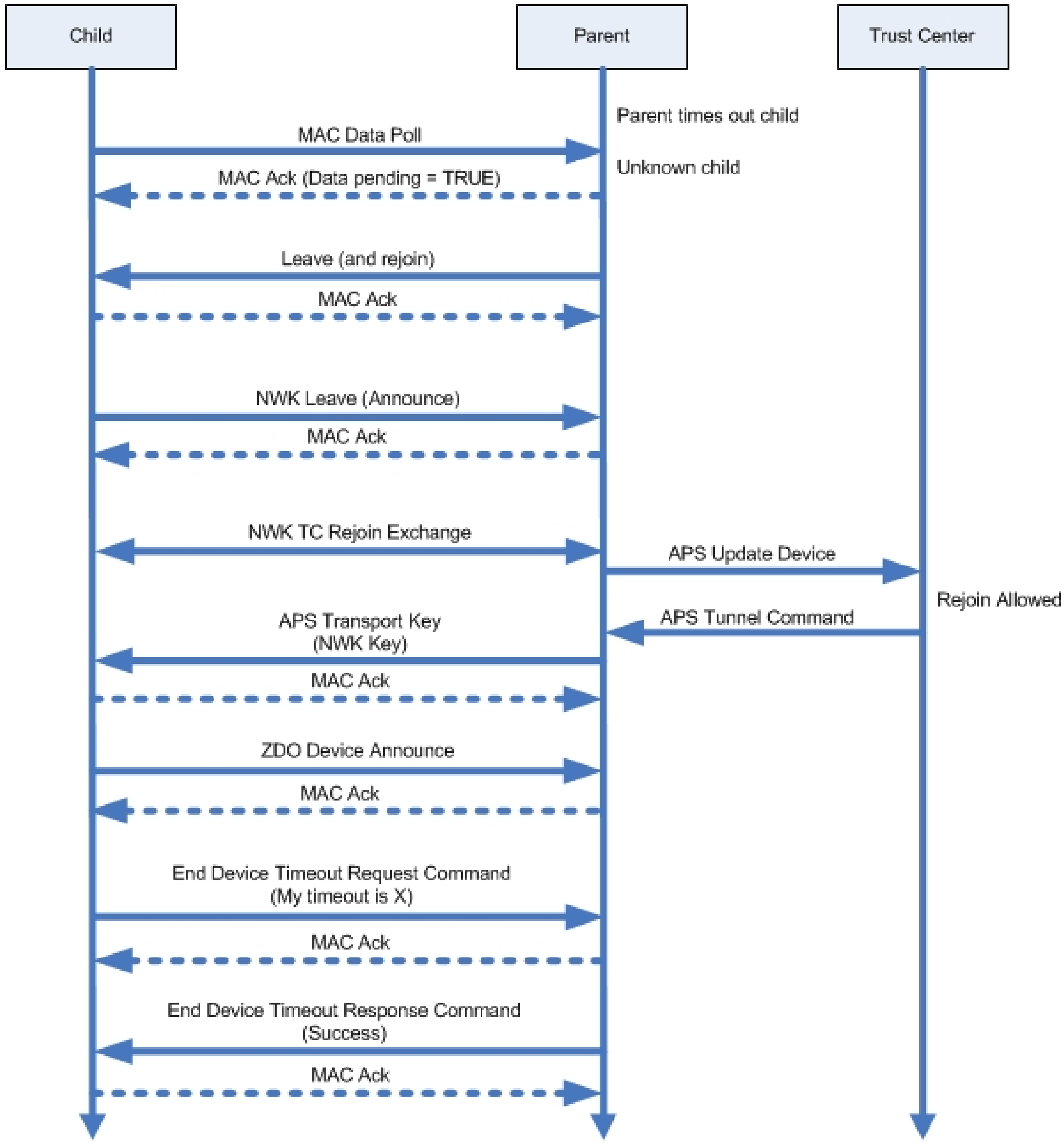


Figure 3.55 and Figure 3.56 show what happens when a parent that supports the MAC data poll keepalive method, ages out the child. The parent will indicate to the child that it has a pending message for the child by setting the data pending bit to TRUE in the MAC acknowledgement. The parent will then transmit a leave message to the device with the rejoin bit set to TRUE. The device will announce leaving the network and perform a rejoin. Figure 3.55 shows a secure rejoin while Figure 3.56 shows a Trust Center Rejoin. After the rejoin is successful the device will send the NWK Command End Device Timeout Request and receive a response.

Figure 3.57 Aging out Children: End Device Timeout Request Method - Secure Rejoin

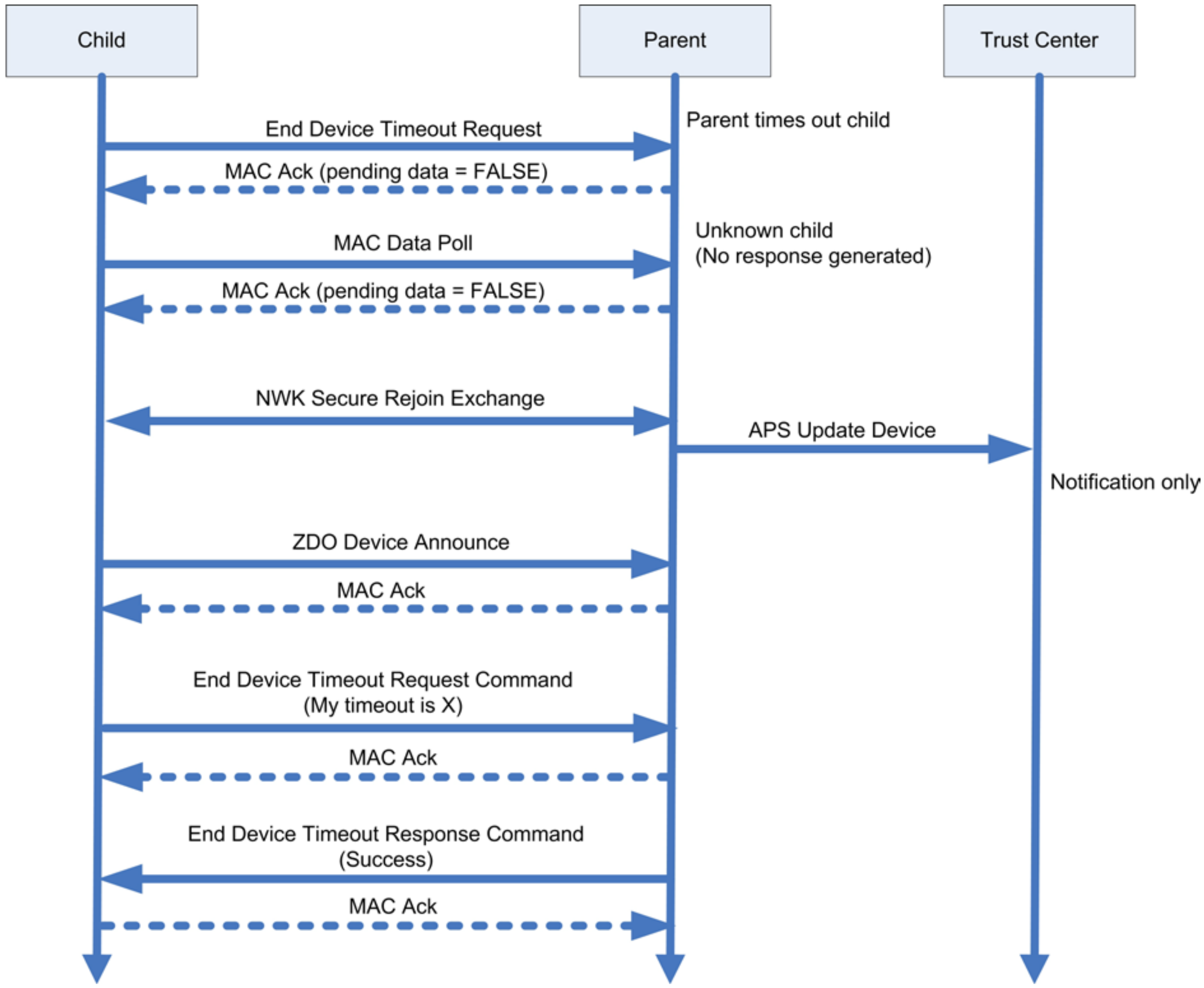


Figure 3.58 Aging out Children: End Device Timeout Request Method - Trust Center Rejoin

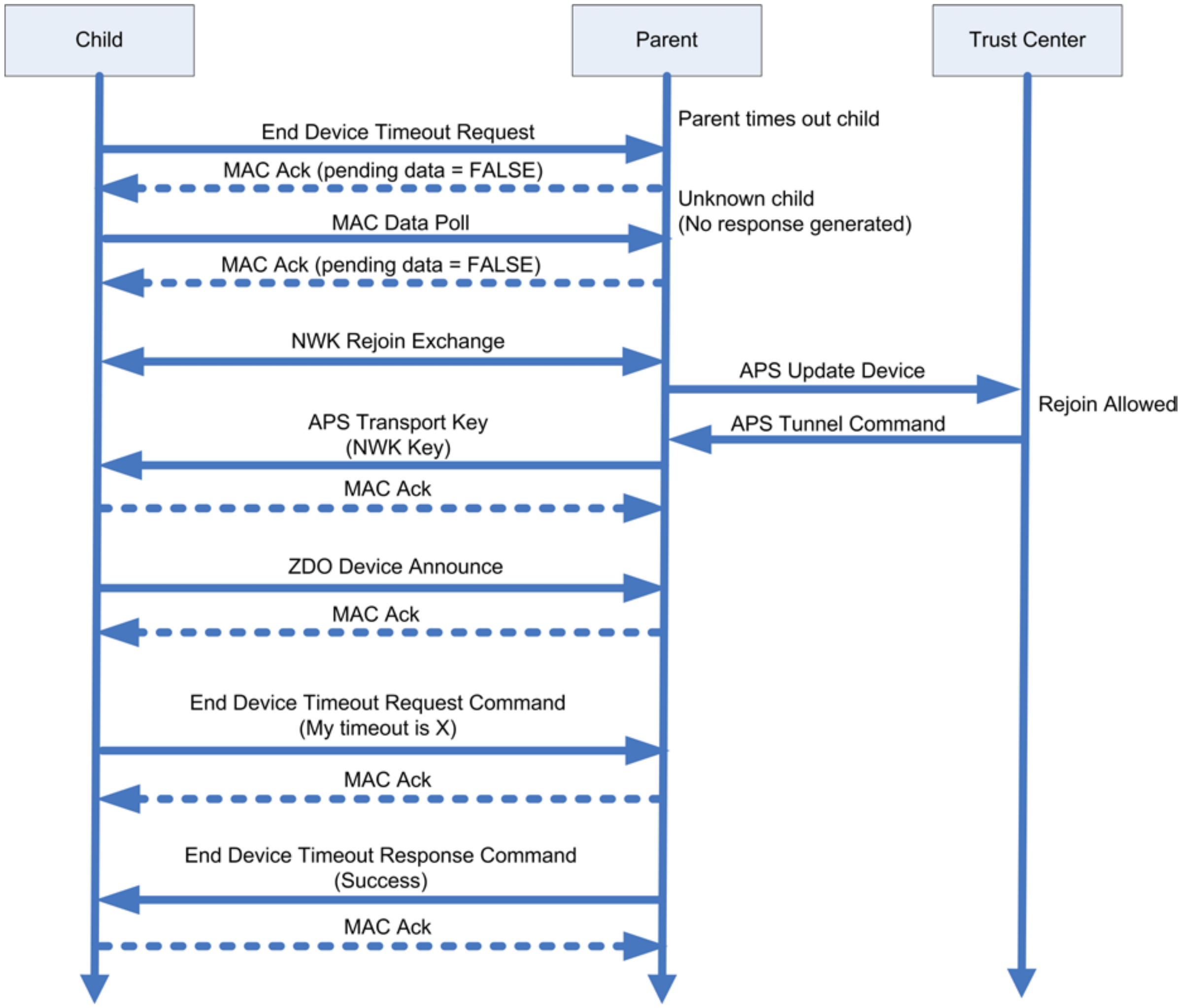


Figure 3.57 and Figure 3.58 shows what happens when an end device is aged out of the parent's table with a parent that supports the End Device Timeout Request method. An end device sends an End Device Timeout Request and receives no response. Afterwards it will perform a rejoin. Figure 3.57 shows a secure rejoin while Figure 3.58 shows a Trust Center rejoin. Once the device has completed the rejoin it will send a NWK command End Device timeout request and receive the response.

3.6.10.10 Trust Center Rejoin or Secure Rejoin

An end device that has detected it has been aged out of its parent's child table may choose to use either a Secure Rejoin or a Trust Center rejoin. The choice to use one or the other is up to the implementation but can be based on whether it may have missed a network key update. A device that has missed a network key update will have to use a Trust Center Rejoin. However in a case where that situation has not occurred, a Secure Rejoin will complete more quickly and can be used instead. It is possible that an end device may try both methods to insure it can get back on the network.

3.7 NWK Layer Status Values

Network (NWK) layer confirmation primitives often include a parameter that reports on the status of the request to which the confirmation applies. Values for NWK layer Status parameters appear in Table 3.62.

Table 3.62 NWK Layer Status Values

Name	Value	Description
SUCCESS	0x00	A request has been executed successfully.
INVALID_PARAMETER	0xc1	An invalid or out-of-range parameter has been passed to a primitive from the next higher layer.
INVALID_REQUEST	0xc2	The next higher layer has issued a request that is invalid or cannot be executed given the current state of the NWK layer.
NOT_PERMITTED	0xc3	An NLME-JOIN.request has been disallowed.
STARTUP_FAILURE	0xc4	An NLME-NETWORK-FORMATION.request has failed to start a network.
ALREADY_PRESENT	0xc5	A device with the address supplied to the NLME-DIRECT-JOIN.request is already present in the neighbor table of the device on which the NLME-DIRECT-JOIN.request was issued.
SYNC_FAILURE	0xc6	Used to indicate that an NLME-SYNC.request has failed at the MAC layer.
NEIGHBOR_TABLE_FULL	0xc7	An NLME-JOIN-DIRECTLY.request has failed because there is no more room in the neighbor table.
UNKNOWN_DEVICE	0xc8	An NLME-LEAVE.request has failed because the device addressed in the parameter list is not in the neighbor table of the issuing device.
UNSUPPORTED_ATTRIBUTE	0xc9	An NLME-GET.request or NLME-SET.request has been issued with an unknown attribute identifier.
NO_NETWORKS	0xca	An NLME-JOIN.request has been issued in an environment where no networks are detectable.
Reserved	0xcb	
MAX_FRM_COUNTER	0xcc	Security processing has been attempted on an outgoing frame, and has failed because the frame counter has reached its maximum value.
NO_KEY	0xcd	Security processing has been attempted on an outgoing frame, and has failed because no key was available with which to process it.

Name	**Value**	**Description**
BAD_CCM_OUTPUT	0xce	Security processing has been attempted on an outgoing frame, and has failed because the security engine produced erroneous output.
Reserved	0xcf	
ROUTE_DISCOVERY_FAILED	0xd0	An attempt to discover a route has failed due to a reason other than a lack of routing capacity.
ROUTE_ERROR	0xd1	An NLDE-DATA.request has failed due to a routing failure on the sending device or an NLME-ROUTE-DISCOVERY.request has failed due to the cause cited in the accompanying NetworkStatusCode.
BT_TABLE_FULL	0xd2	An attempt to send a broadcast frame or member mode multicast has failed due to the fact that there is no room in the BTT.
FRAME_NOT_BUFFERED	0xd3	An NLDE-DATA.request has failed due to insufficient buffering available. A non-member mode multicast frame was discarded pending route discovery.

This page intentionally left blank.

CHAPTER 4 SECURITY SERVICES SPECIFICATION

4.1 Document Organization

The remaining portions of this document specify in greater detail the various security services available within the ZigBee stack. Basic definitions and references are given in clause 4.2. A general description of the security services is given in section 4.2.1. In this clause, the overall security architecture is discussed; basic security services provided by each layer of this architecture are introduced. Sections 4.2.2 and 4.2.3 give the ZigBee Alliance's security specifications for the Network (NWK) layer and the Application Support Sublayer (APS) layer, respectively. These clauses introduce the security mechanisms, give the primitives, and define any frame formats used for security purposes. Section 4.5 describes security elements common to the NWK and APS layers. Section 4.6 provides a basic functional description of the available security features. Finally, annexes provide technical details and test vectors needed to implement and test the cryptographic mechanisms and protocols used by the NWK and APS layers.

4.2 General Description

Security services provided for ZigBee include methods for key establishment, key transport, frame protection, and device management. These services form the building blocks for implementing security policies within a ZigBee device. Specifications for the security services and a functional description of how these services shall be used are given in this document.

4.2.1 Security Architecture and Design

In this clause, the security architecture is described. Where applicable, this architecture complements the security services that are already present in the IEEE Std. 802.15.4 802 [B1] security specification.

4.2.1.1 Security Assumptions

The level of security provided by the ZigBee security architecture depends on the safekeeping of the symmetric keys, on the protection mechanisms employed, and on the proper implementation of the cryptographic mechanisms and associated security policies involved. Trust in the security architecture ultimately reduces to trust in the secure initialization and installation of keying material and to trust in the secure processing and storage of keying material.

Implementations of security protocols, such as key establishment, are assumed to properly execute the complete protocol and not to leave out any steps thereof. Random number generators are assumed to operate as expected. Furthermore, it is assumed that secret keys do not become available outside the device in an unsecured way. That is, a device will not intentionally or inadvertently transmit its keying material to other devices unless the keying material is protected, such as during key-transport. During initial key transport the keying material used for protection may be a well-known key, thus resulting in a brief moment of vulnerability where the key could be obtained by any device. Alternatively, the initial key transport may be done using a pre-shared secret key that is passed out-of-band from the ZigBee network. The following caveat in these assumptions applies: due to the low-cost nature of *ad hoc* network devices, one cannot generally assume the availability of tamper-resistant hardware. Hence, physical access to a device may yield access to secret keying material and other privileged information, as well as access to the security software and hardware.

Due to cost constraints, ZigBee has to assume that different applications using the same radio are not logically separated (for example, by using a firewall). In addition, from the perspective of a given device it is not even possible (barring certification) to verify whether cryptographic separation between different applications on another device — or even between different layers of the communication stack thereof — is indeed properly implemented. Hence, one must assume that separate applications using the same radio trust each other; that is, there is no cryptographic task separation. Additionally, lower layers (for example, APS, NWK, or MAC) are fully accessible by any of the applications. These assumptions lead to an open trust model for a device; different layers of the communication stack and all applications running on a single device trust each other.

In summary:

- The provided security services cryptographically protect the interfaces between different devices only.
- Separation of the interfaces between different stack layers on the same device is arranged non-cryptographically, via proper design of security service access points.

4.2.1.2 Security Design Choices

The open trust model (as described in section 4.2.1.1) on a device has far-reaching consequences. It allows re-use of the same keying material among different layers on the same device and it allows end-to-end security to be realized on a device-to-device basis rather than between pairs of particular layers (or even pairs of applications) on two communicating devices.

However, one must also take into consideration whether one is concerned with the ability of malevolent network devices to use the network to transport frames across the network without permission.

These observations lead to the following architectural design choices:

First, the principle that "*the layer that originates a frame is responsible for initially securing it*" is established. For example, if a NWK command frame needs protection, NWK layer security shall be used.

Second, if protection from theft of service is required (*i.e.*, from malevolent network devices), NWK layer security shall be used for all frames, except those passed between a router and a newly joined device (until the newly joined device receives the active network key). Thus, only a device that has joined the network and successfully received the active network key will be able to have its messages communicated more than one hop across the network.

Third, due to the open trust model, security can be based on the reuse of keys by each layer. For example, the active network key shall be used to secure APS layer broadcast frames or NWK layer frames. Reuse of keys helps reduce storage costs.

Fourth, end-to-end security is provided such that it is possible for only source and destination devices to access messages protected by a shared key. This ensures that routing of messages between the two devices with the shared key can be independent of trust considerations.

Fifth, to simplify interoperability of devices, the base security level used by all devices in a given network, and by all layers of a device, shall be the same. If an application needs more security for its payload than is provided by network level security, it can establish application level security with another device. There are several policy decisions which any real implementation must address correctly. Application profiles should include policies to:

Handle error conditions arising from securing and unsecuring packets. Some error conditions may indicate loss of synchronization of security material, or may indicate ongoing attacks.

Detect and handle loss of counter synchronization and counter overflow.

Detect and handle loss of key synchronization.

Expire and periodically update keys, if desired.

The other security design choice is done by the device that forms a network. This device sets the security policies and processes followed by the network and devices that join the network.

4.2.1.3 Security Keys

Security amongst a network of ZigBee devices is based on "link" keys and a "network" key. Unicast communication between APL peer entities is secured by means of a 128-bit link key shared by two devices, while broadcast communications and any network layer communications are secured by means of a 128-bit network key shared amongst all devices in the network. The intended recipient is always aware of the exact security arrangement; that is, the recipient knows whether a frame is protected with a link key or a network key.

A device shall acquire link keys either via key-transport, or pre-installation (for example, during factory installation). A device shall acquire a network key via key-transport. Some application profiles have also developed out of band mechanisms or key negotiation protocols used for generating link keys or network keys on devices. Ultimately, security between devices depends on secure initialization and installation of these keys.

There is one type of network key; however, it can be used in either distributed or centralized security models. The security model controls how a network key is distributed; and may control how network frame counters are initialized. The security model does not affect how messages are secured.

There are two different types of trust center link keys: global and unique. The type of trust center link key in use by the local device shall determine how the device handles various trust center messages (APS commands), including whether to apply APS encryption. A Trust Center link key may also be used to secure APS data messages between the Trust Center and the corresponding peer device. The choice of whether to use APS security on those APS data messages is up to the higher layer application.

A link key between two devices, neither of which is the trust center, is known as an application link key.

The default value for the centralized security global trust center link key shall have a value of 5A 69 67 42 65 65 41 6C 6C 69 61 6E 63 65 30 39 (ZigBeeAlliance09).

The different types of keys used are described in Table 4.1.

Table 4.1 Link Keys Used in ZigBee Networks

Key Name	Description
Centralized security global trust center link key	Link key used for joining centralized security networks
Distributed security global link key	Link key used for joining distributed security networks
Install code link key	Link key derived from install code from joining device to create unique trust center link key for joining

Application link key	Link key used between two devices for application layer encryption
Device Specific trust center link key	Link key used between the trust center and a device in the network. Used for trust center commands and application layer encryption.

In a secured network there are a variety of security services available. Prudence dictates that one would prefer to avoid re-use of keys across different security services, which otherwise could cause security leaks due to unwanted interactions. As such, these different services use a key derived from a one-way function using the link key (as specified in section 4.5.3). The use of uncorrelated keys ensures logical separation of the execution of different security protocols. The key-load key is used to protect transported link keys; the key-transport key is used to protect transported network keys. The active network key may be used by the NWK and APL layers of ZigBee. As such, the same network key and associated outgoing and incoming frame counters shall be available to all of these layers. The link keys may be used only by the APS sublayer. As such, the link key shall be available only to the APL layer.

An installation code is a short code that uses an algorithm to derive the 128-bit AES key. The mechanism for deriving a key from an installation code are out of scope of this specification.

4.2.1.4 ZigBee Security Architecture

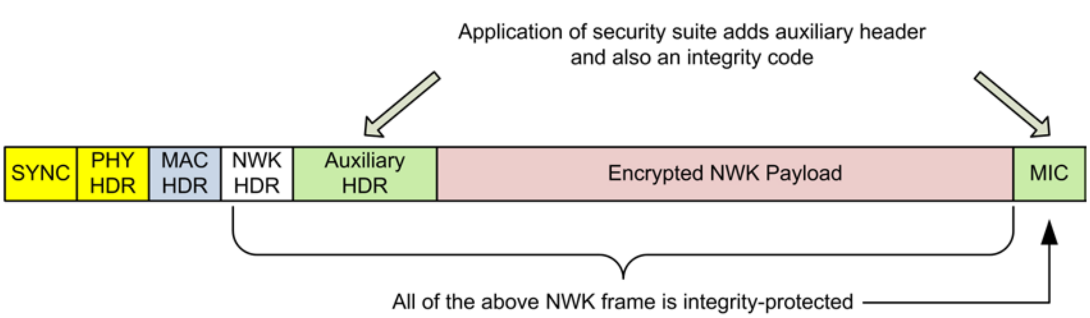
The ZigBee security architecture includes security mechanisms at two layers of the protocol stack. The NWK and APS layers are responsible for the secure transport of their respective frames. Furthermore, the APS sublayer provides services for the establishment and maintenance of security relationships. The ZigBee Device Object (ZDO) manages the security policies and the security configuration of a device. Figure 1.1 shows a complete view of the ZigBee protocol stack. The security mechanisms provided by the APS and NWK layers are described in this version of the specification.

4.2.2 NWK Layer Security

When a frame originating at the NWK layer needs to be secured ZigBee shall use the frame-protection mechanism given in section 4.3.1 of this specification, unless the SecurityEnable parameter of the NLDE-DATA.request primitive is FALSE, explicitly prohibiting security. For example, no NWK layer security is used during transport of the NWK Key over the last hop to a joining device since APS security will be used to protect the frame. The NWK layer's frame-protection mechanism shall make use of the Advanced Encryption Standard (AES) [B8] and use CCM* as specified in Annex A. The security level applied to a NWK frame shall be determined by the *nwkSecurityLevel* attribute in the NIB. Upper layers manage NWK layer security by setting up active and alternate network keys and by determining which security level to use.

Figure 4.1 shows an example of the security fields that may be included in a NWK frame.

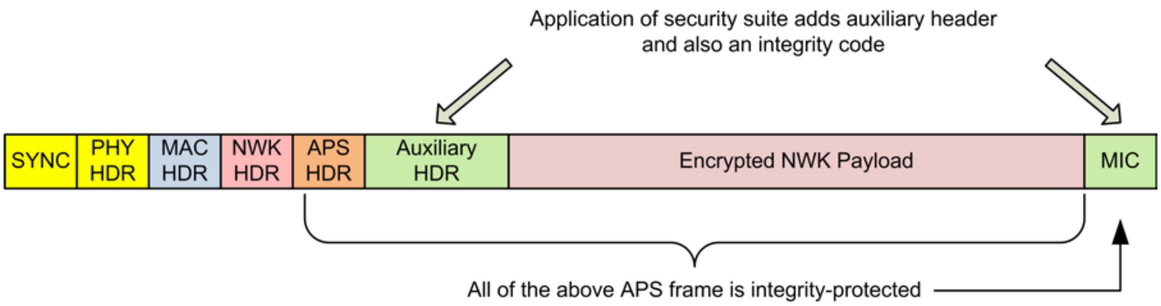
Figure 4.1 ZigBee Frame with Security on the NWK Level



4.2.3 APL Layer Security

When a frame originating at the APL layer needs to be secured, the APS sublayer shall handle security. The APS layer's frame-protection mechanism is given in section 4.4.1 of this specification. The APS layer allows frame security to be based on link keys or the network key. Figure 4.2 shows an example of the security fields that may be included in an APL frame. The APS layer is also responsible for providing applications and the ZDO with key establishment, key transport, and device management services.

Figure 4.2 ZigBee Frame with Security on the APS Level



4.2.3.1 Transport Key

The transport-key service provides secured means to transport a key to another device or other devices. The secured transport-key command provides a means to transport link, or network key from a key source (for example, the Trust Center) to other devices.

4.2.3.2 Update Device

The update device service provides a secure means for a router device to inform the Trust Center that a third device has had a change of status that must be updated (for example, the device joined or left the network). This is the mechanism by which the Trust Center maintains an accurate list of active network devices.

4.2.3.3 Remove Device

The remove device service provides a secure means by which a Trust Center informs a router device that one of the router's children or the router itself must be removed from the network. For example, the remove device service may be employed to remove from a network a device that has not satisfied the Trust Center's security requirements for network devices.

4.2.3.4 Request Key

The request-key service provides a secure means for a device to request an end-to-end application link key or trust center link key, from the Trust Center.

4.2.3.5 Switch Key

The switch-key service provides a secure means for a Trust Center to inform another device that it should switch to a different active network key.

4.2.3.6 Verify-Key

The verify-key service provides a secure means for a device to verify that the device and the Trust Center agree on the current value of the device's link key.

4.2.3.7 Confirm Key

The confirm-key service provides a secure means for a Trust Center to confirm a previous request to verify a link key.

4.2.4 Trust Center Role

For security purposes, ZigBee defines the role of "Trust Center". The Trust Center is the device trusted by devices within a network to distribute keys for the purpose of network and potentially end-to-end application configuration management. All members of the network shall recognize exactly one active Trust Center, and there shall be exactly one Trust Center in each centralized security network. The Trust Center is responsible for establishing, maintaining and updating security policies for the network.

In a distributed security network, all routers have the capability to act as the Trust Center and distribute keys for network security. This distributed trust center role is used for network key distribution but not trust center link key distribution since there is not a singular trust center in the network.

In some applications a device can be pre-loaded with the Trust Center address and initial Trust Center link key, or the joining device's Trust Center link key can be installed out of band.

In applications that can tolerate a moment of vulnerability, the network key can be sent via APS secured key transport using a well-known link key.

In a centralized security model, the Trust Center established policies for joining devices and network security. It may require devices to be known before providing the network key update for joining, or may require a preconfigured link key be installed out of band. These Trust Center policies are described in section 4.7.1.

In a centralized security network a device securely communicates with its Trust Center using the current Trust Center link key.

For purposes of trust management, a device only accepts a Trust Center link key or active network key originating from its Trust Center via key transport. For purposes of network management in a centralized security network, a device accepts an initial active network key and updated network keys only from its Trust Center when secured with its Trust Center Link key. For purposes of configuration, a device accepts link keys intended for establishing end-to-end security between two devices only from its Trust Center or through application level negotiation using a higher level protocol between the two devices. Aside from the initial Trust Center link key or network key, additional link, and network keys are only accepted if they originate from a device's Trust Center via secured key transport or negotiated using higher level application protocols.

4.3 NWK Layer Security

The NWK layer is responsible for the processing steps needed to securely transmit outgoing frames and securely receive incoming frames. Upper layers control the security processing operations by setting up the appropriate keys and frame counters and establishing which security level to use.

4.3.1 Frame Security

The detailed steps involved in security processing of outgoing and incoming NWK frames are described in sections 4.3.1.1 and 4.3.1.2, respectively.

4.3.1.1 Security Processing of Outgoing Frames

If the NWK layer has a frame, consisting of a header *NwkHeader* and payload *Payload*, which needs security protection and *nwkSecurityLevel* > 0, and in the case of a NWK data frame, the SecurityEnabled parameter in NLDEDATA.request had a value of TRUE, it shall apply security as follows:

1. Obtain the *nwkActiveKeySeqNumbe*r from the NIB and use it to retrieve the active network key *key*, outgoing frame counter *OutgoingFrameCounter*, and key sequence number *KeySeqNumber* from the *nwkSecurityMaterialSet* attribute in the NIB. Obtain the security level from the *nwkSecurityLevel* attribute from the NIB. If the outgoing frame counter is equal to 2^{32}-1, or if the key cannot be obtained, security processing shall fail and no further security processing shall be done on this frame.
2. Construct the auxiliary header *AuxiliaryHeader* (see section 4.5.1):
    a. Set the security control field as follows:
        i. The security level sub-field shall be the security level obtained from step 1.
        ii. The key identifier sub-field shall be set to '01' (that is, the active network key).
        iii. The extended nonce sub-field shall be set to 1.
    b. Set the source address field to the 64-bit extended address of the local device.
    c. Set the frame counter field to the outgoing frame counter from step 1.
    d. Set the key sequence number field to the sequence number from step 1.
3. Execute the CCM mode encryption and authentication operation, as specified in Annex A, with the following instantiations:
    a. Obtain the parameter *M* from Table 4.40 corresponding to the security level from step 1;
    b. The bit string *Key* shall be the key obtained from step 1;
    c. The nonce *N* shall be the 13-octet string constructed using the security control field from step a, the frame counter field from step d, and the source address field from step c (see section 4.5.2.2);
    d. If the security level requires encryption, the octet string *a* shall be the string *NwkHeader* || *AuxiliaryHeader* and the octet string *m* shall be the string *Payload*. Otherwise, the octet string *a* shall be the string *NwkHeader* || *AuxiliaryHeader* || *Payload* and the octet string *m* shall be a string of length zero.
4. If the CCM mode invoked in step 3 outputs 'invalid', security processing shall fail and no further security processing shall be done on this frame.
5. Let *c* be the output from step 3. If the security level requires encryption, the secured outgoing frame shall be *NwkHeader* || *AuxiliaryHeader* || *c*, otherwise the secured outgoing frame shall be *NwkHeader* || *AuxiliaryHeader* || *Payload* || *c.*
6. If the secured outgoing frame size is greater than *aMaxMacFrameSize* security processing shall fail and no further security processing shall be done on this frame.
7. The outgoing frame counter from step 1 shall be incremented by one and stored in the *OutgoingFrameCounter* element of the network security material descriptor referenced by the *nwkActiveKeySeqNum-ber* in the NIB; that is, the outgoing frame counter value associated with the key used to protect the frame is updated.
8. The security level sub-field of the security control field shall be over-written by the 3-bit all-zero string '000'.

4.3.1.2 Security Processing of Incoming Frames

If the NWK layer receives a secured frame (consisting of a header *NwkHeader*, auxiliary header *AuxiliaryHeader*, and payload *SecuredPayload*) as indicated by the security sub-field of the NWK header frame control field, it shall perform security processing as follows:

1. Determine the security level from the *nwkSecurityLevel* attribute of the NIB. Over-write the 3-bit security level sub-field of the security control field of the *AuxiliaryHeader* with this value. Determine the sequence number *SequenceNumber*, sender address *SenderAddress*, and received frame count *ReceivedFrameCount* from the auxiliary header *AuxiliaryHeader* (see section 4.5.1). If *ReceivedFrameCounter* is equal to 2^{32}-1, security processing shall indicate a failure to the next higher layer with a status of 'bad frame counter' and no further security processing shall be done on this frame.
2. Obtain the appropriate security material (consisting of the key and other attributes) by matching *SequenceNumber* to the sequence number of any key in the *nwkSecurityMaterialSet* attribute in the NIB. If the security material cannot be obtained, security processing shall indicate a failure to the next higher layer with a status of 'frame security failed' and no further security processing shall be done on this frame.
3. If there is an incoming frame count *FrameCount* corresponding to *SenderAddress* from the security material obtained in step 2 and if *ReceivedFrameCount* is less than *FrameCount*, security processing shall indicate a failure to the next higher layer with a status of 'bad frame counter' and no further security processing shall be done on this frame.
4. Execute the CCM mode decryption and authentication checking operation, as specified in section A.2, with the following instantiations:
   a. The parameter *M* shall be obtained from Table 4.40 corresponding to the security level from step 1.
   b. The bit string *Key* shall be the key obtained from step 2.
   c. The nonce *N* shall be the 13-octet string constructed using the security control, the frame counter, and the source address fields from *AuxiliaryHeader* (see section 4.5.2.2). Note that the security level subfield of the security control field has been overwritten in step 1 and now contains the value determined from the *nwkSecurityLevel* attribute from the NIB.
   d. The octet string *SecuredPayload* shall be parsed as *Payload1* || *Payload2*, where the rightmost string *Payload2* is an *M*-octet string. If this operation fails, security processing shall indicate a failure to the next higher layer with a status of 'frame security failed' and no further security processing shall be done on this frame.
   e. If the security level requires decryption, the octet string *a* shall be the string *NwkHeader* || *AuxiliaryHeader* and the octet string *c* shall be the string *SecuredPayload*. Otherwise, the octet string *a* shall be the string *NwkHeader* || *AuxiliaryHeader* || *Payload1* and the octet string *c* shall be the string *Payload2*.
5. Return the results of the CCM operation:
   a. If the CCM mode invoked in step 4 outputs ‘invalid’, security processing shall indicate a failure to the next higher layer with a status of 'frame security failed' and no further security processing shall be done on this frame.
   b. Let *m* be the output of step 4. If the security level requires encryption, set the octet string *UnsecuredNwkFrame* to the string *NwkHeader* || *m*. Otherwise, set the octet string *UnsecuredNwkFrame* to the string *NwkHeader* || *Payload1*.
6. Set *FrameCount* to (*ReceivedFrameCount* + 1) and store both *FrameCount* and *SenderAddress* in the NIB. If storing this frame count and address information will cause the memory allocation for this type of information to be exceeded, and the *nwkAllFresh* attribute in the NIB is TRUE, then security processing shall fail and no further security processing shall be done on this frame. *UnsecuredNwkFrame* now represents the unsecured received network frame and security processing shall succeed. So as to never cause the storage of the frame count and address information to exceed the available memory, the memory allocated for incoming frame counters needed for NWK layer security shall be bounded by *M*N*, where *M* and *N* represent the cardinality of *nwkSecurityMaterialSet* and *nwkNeighborTable* in the NIB, respectively.
7. If the sequence number of the received frame belongs to a newer entry in the *nwkSecurityMaterialSet*, set the *nwkActiveKeySeqNumber* to the received sequence number.

8. If there is an entry in nwkNeighborTable in the NIB whose extended address matches SenderAddress and whose relationship field has value 0x05 (unauthenticated child), then set relationship field in that entry to the value 0x01 (child).

4.3.2 Secured NPDU Frame

The NWK layer frame format (see section 3.3.1) consists of a NWK header and NWK payload field. The NWK header consists of frame control and routing fields. When security is applied to an NPDU frame, the security bit in the NWK frame control field shall be set to 1 to indicate the presence of the auxiliary frame header. The format for the auxiliary frame header is given in section 4.5.1. The format of a secured NWK layer frame is shown in Figure 4.3. The auxiliary frame header is situated between the NWK header and payload fields.

Figure 4.3 Secured NWK Layer Frame Format

<table>
<tr><th>Octets: Variable</th><th>14</th><th colspan="2">Variable</th></tr>
<tr><td rowspan="2">Original NWK header ([B3], Clause 7.1)</td><td rowspan="2">Auxiliary frame header</td><td>Encrypted payload</td><td>Encrypted message integrity code (MIC)</td></tr>
<tr><td colspan="2">Secure frame payload = output of CCM</td></tr>
<tr><td colspan="2">Full NWK header</td><td colspan="2">Secured NWK payload</td></tr>
</table>

4.3.3 Security-Related NIB Attributes

The NWK PIB contains attributes that are required to manage security for the NWK layer. Each of these attributes can be read and written using the NLMEGET.request and NLME-SET.request primitives, respectively. The security-related attributes contained in the NWK PIB are presented in Table 4.2, Table 4.3, and Table 4.4.

Table 4.2 NIB Security Attributes

Attribute	Identifier	Type	Range	Description	Default
nwkSecurityLevel	0xa0	Octet	0x00-07	The security level for outgoing and incoming NWK frames; the allowable security level identifiers are presented in Table 4.40.	0x05
nwkSecurityMaterialSet	0xa1	A set of 2 network security material descriptors (see Table 4.2)	Variable	Set of network security material descriptors capable of maintaining an active and alternate network key.	-

Attribute	Identifier	Type	Range	Description	Default
nwkActiveKey SeqNumber	0xa2	Octet	0x00-0xFF	The sequence number of the active network key in *nwkSecurityMaterialSet.*	0x00
nwkAllFresh	0xa3	Boolean	TRUE | FALSE	Indicates whether incoming NWK frames must be all checked for freshness when the memory for incoming frame counts is exceeded. See section 4.3.1.2.	TRUE

Table 4.3 Elements of the Network Security Material Descriptor

Name	Type	Range	Description	Default
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a network key by the Trust Center and used to distinguish network keys for purposes of key updates, and incoming frame security operations. This is only used when operating in a centralized security network.	00
OutgoingFrame Counter	Ordered set of 4 octets.	0x00000000-0xFFFFFFFF	Outgoing frame counter used for outgoing frames.	0x00000000
IncomingFrameCounterSet	Set of incoming frame counter descriptor values. See Table 4.3.	Variable	Set of incoming frame counter values and corresponding device addresses.	Null set
Key	Ordered set of 16 octets.	-	The actual value of the key.	-

Name	Type	Range	Description	Default
NetworkKeyType	Octet	0x01 - 0x01	The type of the key. 0x01 = standard All other values are reserved.	0x01

Table 4.4 Elements of the Incoming Frame Counter Descriptor

Name	Type	Range	Description	Default
SenderAddress	Device address	Any valid 64-bit address	Extended device address.	Device specific
IncomingFrame Counter	Ordered set of 4 octets	0x00000000-0xFFFFFFFF	Incoming frame counter used for incoming frames.	0x00000000

4.3.4 Network Frame Counter Requirements

Device shall maintain outgoing NWK frame counters across factory resets. The outgoing NWK frame counter must only be reset as detailed in this specification. A factory reset includes any over the air message, such as a NWK leave. It is permitted for manufacturers to provide a full factory reset that erases all persisted data as a separate user action.

A device can join a network, join other networks and then attempt to join the original network again. Neighbors on the original network will have a neighbor table entry for the device with the incoming frame counter set to the value that was heard when the device was previously on the network. If a fresh security material set with an outgoing NWK frame counter of zero is created when the original network is joined for a second time, devices in that network will reject frames sent with this frame counter. Devices must therefore have sufficient shadow copies of their security material set and associated EPID to store the outgoing frame counter and EPID for each network that they may join. As an implementation optimization, it is permissible to store a single instance of the outgoing NWK frame counter that is used across all security material sets. This outgoing NWK frame counter must be preserved across factory resets and when joining different networks. The only time the outgoing frame counter is reset to zero is when the device is already on a network, it receives an APSME-SWITCH-KEY and its outgoing frame counter is greater than 0x80000000.

4.3.4.1 Network Frame Counter Usage Calculations

One leap year is 366*24*60*60 = 31,622,400 seconds. The frame counter will wrap every 4,294,967,295 counts. Therefore a device would need to continuously send at a rate greater than 135 packets per second to cause the frame counter to wrap in less than a year.

Often devices do not store the exact frame counter in flash memory but use a store ahead method to prevent wearing out flash memory. This will cause the device to jump its frame counter ahead on reboot to the next higher increment. If a device increments its frame counter by 1024 on a reboot, it would have to reboot at a rate greater than once every 7 seconds to cause a wrap in a year.

A device must be able to store two network keys. If there are two network key updates whilst the device is asleep or turned off, it will no longer have a valid network key and will only be able to join the network via a Trust center rejoin. Limiting the network key updates to a maximum of once every 30 days mitigates this issue.

4.4 APS Layer Security

The APS layer is responsible for the processing steps needed to securely transmit outgoing frames, securely receive incoming frames, and securely establish and manage cryptographic keys. Upper layers control the management of cryptographic keys by issuing primitives to the APS layer.

Table 4.5 lists the primitives available for key management and maintenance. Upper layers also determine which security level to use when protecting outgoing frames.

Table 4.5 The APS Layer Security Primitives

APSME Security Primitives	Request	Confirm	Indication	Response	Description
APSME-TRANSPORT-KEY	section 4.4.2.1	-	section 4.4.2.2	-	Transports security material from one device to another.
APSME-UPDATE-DEVICE	section 4.4.3.1	-	section 4.4.3.2	-	Notifies the Trust Center when a new device has joined, or an existing device has left the network.
APSME-REMOVE-DEVICE	section 4.4.4.1	-	section 4.4.4.2	-	Used by the Trust Center to notify a router that one of the router's child devices, or the router itself, should be removed from the network.
APSME-REQUEST-KEY	section 4.4.5.1	-	section 4.4.5.2	-	Used by a device to request that the Trust Center send an application link key or trust center link key.
APSME-SWITCH-KEY	section 4.4.6.1	-	section 4.4.6.2	-	Used by the Trust Center to tell a device to switch to a new network key.
APSME-VERIFY-KEY	section 4.4.7.1	-	section 4.4.7.2	-	Used by a device to verify the link key used by the trust center.

APSME Security Primitives	Request	Confirm	Indication	Response	Description
APSME-CONFIRM-KEY	section 4.4.8.1		section 4.4.8.2	-	Used by the trust center to confirm a previous request to verify a link key.

4.4.1 Frame Security

The detailed steps involved in security processing of outgoing and incoming APS frames are described in sections 4.4.1.1 and 4.4.1.2, respectively.

4.4.1.1 Security Processing of Outgoing Frames

If the APS layer has a frame, consisting of a header *ApsHeader* and payload *Payload*, that needs security protection and *nwkSecurityLevel* > 0, it shall apply security as follows:

1. Obtain the security material and key identifier *KeyIdentifier* using the following procedure. If security material or key identifier cannot be determined, then security processing shall fail and no further security processing shall be done on this frame.
    a. If the frame is a result of a APSDE-DATA.request primitive:
        i. The security material associated with the destination address of the outgoing frame shall be obtained from the *apsDeviceKeyPairSet* attribute in the AIB. *KeyIdentifier* shall be set to '00' (that is, a data key).
        ii. Only entries with a KeyAttribute of PROVISIONAL or VERIFIED shall be used. Keys with other attributes shall not be used for encryption.
    b. If the frame is a result of an APS command that requires securing.
        i. An attempt shall be made to retrieve the security material associated with the destination address of the outgoing frame from the *apsDeviceKeyPairSet* attribute in the AIB. Only entries with a KeyAttribute of PROVISIONAL or VERIFIED shall be used. Keys with other attributes shall not be used for encryption.
        ii. For all cases except transport-key commands, *KeyIdentifier* shall be set to '00'(that is, a data key). For the case of transport-key commands, *KeyIdentifier* shall be set to '02' (that is, the key-transport key) when transporting a network key and shall be set to '03' (that is, the key-load key) when transporting an application link key or trust center link key. See section 4.5.3 for a description of the key-transport and key-load keys.
2. Extract the outgoing frame counter (and, if *KeyIdentifier* is 01, the key sequence number) from the security material obtained from step 1. If the outgoing frame counter value is equal to integer 2^{32}-1, or if the key cannot be obtained, security processing shall fail and no further security processing shall be done on this frame.
3. Obtain the security level from the *nwkSecurityLevel* attribute from the NIB.
4. Construct auxiliary header *AuxiliaryHeader* (see section 4.5.1). The security control field shall be set as follows:
    a. The security level sub-field shall be the security level obtained from step 3.
        i. The key identifier sub-field shall be set to *KeyIdentifier*.
        ii. The extended nonce sub-field shall be set as follows: If the ApsHeader indicates the frame type is an APS Command, then the extended nonce sub-field shall be set to 1. Otherwise if the TxOptions bit for include extended nonce is set (0x10) then the extended nonce sub-field shall be set to 1. Otherwise it shall be set to 0.

b. If the extended nonce sub-field is set to 1, then set the source address field to the 64-bit extended address of the local device.

c. The frame counter field shall be set to the outgoing frame counter from step 2.

d. If *KeyIdentifier* is '01', the key sequence number field shall be present and set to the key sequence number from step 3. Otherwise, the key sequence number field shall not be present.

5. Execute the CCM mode encryption and authentication operation, as specified in section A.2, with the following exceptions:

   a. The parameter *M* shall be obtained from Table 4.40 corresponding to the security level from step 3.

   b. The bit string *Key* shall be the key obtained from step 1.

   c. The nonce *N* shall be the 13-octet string constructed using the security control and frame counter fields from step 5 and the 64-bit extended address of the local device (see section 4.5.2.2).

   d. If the security level requires encryption, the octet string *a* shall be the string *ApsHeader* || *AuxiliaryHeader* and the octet string *m* shall be the string *Payload*. Otherwise, the octet string *a* shall be the string *ApsHeader* || *AuxiliaryHeader* || *Payload* and the octet string *m* shall be a string of length zero.

6. If the CCM mode invoked in step 6 outputs "invalid", security processing shall fail and no further security processing shall be done on this frame.

7. Let *c* be the output from step 6. If the security level requires encryption, the secured outgoing frame shall be *ApsHeader* || *AuxiliaryHeader* || *c*, otherwise the secured outgoing frame shall be *ApsHeader* || *AuxiliaryHeader* || *Payload* || *c.*

8. If the secured outgoing frame size will result in the MSDU being greater than *aMaxMACFrameSize* octets (see IEEE Std. 802.15.4 802 [B1]), security processing shall fail and no further security processing shall be done on this frame.

9. The outgoing frame counter from step 3 shall be incremented and stored in the appropriate location(s) of the NIB, AIB, and MAC PIB corresponding to the key that was used to protect the outgoing frame.

10. Over-write the security level sub-field of the security control field with the 3- bit all-zero string '000'.

4.4.1.2 Security Processing of Incoming Frames

If the APS layer receives a secured frame (consisting of a header *ApsHeader*, auxiliary header *AuxiliaryHeader*, and payload *SecuredPayload*) as indicated by the security sub-field of the APS header frame control field it shall perform security processing as follows:

1. Determine the sequence number *SequenceNumber*, key identifier *KeyIdentifier*, and received frame counter value *ReceivedFrameCounter* from the auxiliary header *AuxiliaryHeader*. If *ReceivedFrameCounter* is the 4-octet representation of the integer 2^{32}-1, security processing shall fail and no further security processing shall be done on this frame.

2. Determine the source address *SourceAddress* from the address-map table in the NIB, using the source address in the APS frame as the index. If the source address is incomplete or unavailable, determine if the device is joined and unauthorized. If joined and unauthorized it shall use the *apsDeviceKeyPairSet* that corresponds to its pre-installed link key. Otherwise, security processing shall fail and no further security processing shall be done on this frame.

3. Obtain the appropriate security material in the following manner. If the security material cannot be obtained, security processing shall fail and no further security processing shall be done on this frame.

   a. If *KeyIdentifier* is '00' (*i.e.*, a data key), the security material associated with the *SourceAddress* of the incoming frame shall be obtained from the *apsDeviceKeyPairSet* attribute in the AIB.

   b. If is '02' (*i.e.*, a key-transport key), the security material associated with the *SourceAddress* of the incoming frame shall be obtained from the *apsDeviceKeyPairSet* attribute in the AIB; the key for this operation shall be derived from the security material as specified in section 4.5.3 for the key-transport key.

c. If *KeyIdentifier* is ‘03’ (*i.e.*, a key-load key), the security material associated with the *SourceAddress* of the incoming frame shall be obtained from the *apsDeviceKeyPairSet* attribute in the AIB and the key for this operation shall be derived from the security material as specified in section 4.5.3 for the key-load key.

4. If the *apsLinkKeyType* of the associated link key is 0x00 (unique) and there is an incoming frame count *FrameCount* corresponding to *SourceAddress* from the security material obtained in step 3 and if *ReceivedFrameCount* is less than *FrameCount*, security processing shall fail and no further security processing shall be done on this frame.

5. Determine the security level *SecLevel* as follows. If the frame type sub-field of the frame control field of *ApsHeader* indicates an APS data frame, then *SecLevel* shall be set to the *nwkSecurityLevel* attribute in the NIB. Overwrite the security level sub-field of the security control field in the *AuxiliaryHeader* with the value of *SecLevel*.

6. Execute the CCM mode decryption and authentication checking operation as specified in section A.3, with the following instantiations:

   a. The parameter *M* shall be obtained from Table 4.40 corresponding to the security level from step 5.

      i. The bit string *Key* shall be the key obtained from step 3.

      ii. The nonce *N* shall be the 13-octet string constructed using the security control and frame counter fields from *AuxiliaryHeader*, and *SourceAddress* from step 2 (see section 4.5.2.2).

      iii. Parse the octet string *SecuredPayload* as *Payload1* || *Payload2*, where the rightmost string$Payload_2$ is an *M*-octet string. If this operation fails, security processing shall fail and no further security processing shall be done on this frame.

      iv. If the security level requires encryption, the octet string *a* shall be the string *ApsHeader* || *AuxiliaryHeader* and the octet string *c* shall be the string *SecuredPayload*. Otherwise, the octet string *a* shall be the string *ApsHeader* || *AuxiliaryHeader* || $Payload_1$ and the octet string *c* shall be the string $Payload_2$.

7. Return the results of the CCM operation:

   a. If the CCM mode invoked in step 6 outputs “invalid”, security processing shall fail and no further security processing shall be done on this frame.

   b. Let *m* be the output of step 6. If the security level requires encryption, set the octet string *UnsecuredApsFrame* to the string *ApsHeader* || *m*. Otherwise, set the octet string *UnsecuredApsFrame* to the string *ApsHeader* || *Payload*.

8. Set *FrameCount* to (*ReceivedFrameCount* + 1) and store both *FrameCount* and *SourceAddress* in the appropriate security material as obtained in step 3. If storing this frame count and address information will cause the memory allocation for this type of information to be exceeded, and the *nwkAllFresh* attribute in the NIB is TRUE, then security processing shall fail and no further security processing shall be done on this frame. Otherwise, security processing shall succeed.

4.4.1.3 Security Processing of APS Commands

A device that is not the trust center that receives an APS command shall determine if the message was sent by the trust center or another device for which it has a link key. If operating in a centralized security network and the message was not sent by the trust center then it shall discard the message and no further processing shall be done.

If operating in a centralized security network determining if the Trust Center sent the APS command shall be done as follows. If no APS encryption is present on the message then the device shall examine if there is an IEEE source address within the APS command frame. The IEEE source address shall be compared to the value of *apsTrustCenterAddress* in the AIB. If no IEEE source address is present in the APS command frame then the device shall verify if the NWK source of the message is 0x0000. If there is APS encryption present on the APS command then the device shall verify that the key used to secure the message corresponds to the *apsDeviceKeyPairSet* that has a DeviceAddress equal to the value of the *apsTrustCenterAddress* in the AIB.

If the message was sent by the trust center the device shall then consult the AIB attribute *apsLinkKeyType* associated with the sending device to determine if the key is a unique link key or Global Link key. It shall then consult Table 4.6 to determine the policy that shall be used.

Table 4.6 Security Policy for Accepting APS Commands in a Centralized Security Network

APS Command	Unique Trust Center Link Key (0x00)	Global Trust CenterLink Key (0x01)
Transport Key (0x05)	APS encryption is required as per device policy (see section 4.4.1.5).	APS encryption is required as per device policy (see section 4.4.1.5).
Update Device (0x06)	APS encryption required	APS encryption not required
Remove Device (0x07)	APS encryption required	APS encryption required
Request Key (0x08)	APS encryption required Trust Center Policy may further restrict, see section 4.4.1.5	APS encryption required Trust Center Policy may further restrict, see section 4.4.1.5
Switch Key (0x09)	APS encryption not required	APS encryption not required
Tunnel Data (0x0E)	APS encryption not required	APS encryption not required
Verify-Key (0x0F)	APS encryption not required.	APS encryption not required
Confirm-Key (0x10)	APS encryption required	APS encryption required.

Upon reception of an APS command that does not have APS encryption but APS encryption is required by Table 4.7, the device shall drop the message and no further processing shall take place. If APS encryption is not required for the command but the received message has APS encryption, the receiving device shall accept and process the message. Accepting additional security on messages is required to support legacy devices in the field.

In order to support backwards compatibility with devices in the field, provisions will also be added for new devices to ensure they can interoperate with the existing devices and their legacy requirements for APS encryption.

Table 4.7 Security Policy for Sending APS Commands in a Centralized Security Network

APS Command	**Unique Trust Center Link Key**	**Global Trust Center Link Key**
Transport Key (0x05)	APS encryption may be optionally used. See section 4.4.1.4	APS encryption may be optionally used. See section 4.4.1.4
Update Device (0x06)	APS encryption shall be used.	APS encryption shall be conditionally used as per section 4.4.1.4.
Remove Device (0x07)	APS encryption shall be used	APS encryption shall be used
Request Key (0x08)	APS encryption shall be used	APS encryption shall be used
Switch Key (0x09)	APS encryption shall not be used	APS encryption shall not be used
Tunnel Data (0x0E)	APS encryption shall not be used	APS encryption shall not be used
Verify-Key (0x0F)	APS encryption shall not be used	APS encryption shall not be used
Confirm-Key (0x10)	APS encryption shall be used	APS encryption shall be used

When the local device will transmit an APS command, it shall consult Table 4.6 above to determine the appropriate behavior. If APS encryption is required to be used, then the device shall APS encrypt the command prior to sending the message. If APS encryption is not to be used, the device shall not APS encrypt the message prior to sending the message. Conditional encryption of APS commands shall follow the procedure as defined by section 4.4.1.4.

4.4.1.4 Conditional Encryption of APS Commands

Devices may have requirements on when APS encryption must or must not be used. To ensure correct operation with those devices, the following procedure shall be undertaken as required by Table 4.6.

When sending an APS command that must be conditionally encrypted, the device shall send the APS command with APS encryption. If the receiving device is capable of accepting APS encrypted APS commands then the sending device may send APS encrypted APS commands. If the receiving device is not capable of receiving APS encrypted commands, then a response to the APS command will not be received. If the receiving device is not capable of receiving APS encrypted APS commands then the sending device can either not send the APS commands or send APS commands without APS encryption.

It is left up to the implementers to determine whether or not the receiving device is capable of receiving an APS command with APS encryption. A device may simply send two copies of the APS command, one with APS encryption and one without, in order to satisfy the requirements of interoperability with existing devices. Note this is not for APS datagrams this is for APS Command Frames.

Conditional encryption of APS commands shall only apply when the *apsLinkKeyType* with receiving device is set to Global Link key (0x01).

4.4.1.5 Acceptance of Commands Based on Security Policy

There are two commands that may be conditionally accepted based on the local security policies in place on the device.

The APS transport key command may be sent with or without APS encryption. The decision to do so is based on the trust center's security policies. The trust center may deem it acceptable to send a key without APS encryption based on the method of transport.

Conversely, a device receiving an APS transport key command may choose whether or not APS encryption is required. This is most often done during initial joining. For example, during joining a device that has no preconfigured link key would only accept unencrypted transport key messages, while a device with a pre-configured link key would only accept a transport key APS encrypted with its preconfigured key.

The higher level specification implemented by the device may dictate the policies in place for these commands.

A device that is in the joined and authorized state shall accept a broadcast NWK key update sent by the Trust Center using only NWK encryption. A device that is in state of joined and unauthorized shall require an APS encrypted transport key if it has a preconfigured link key.

4.4.1.6 Conditional Encryption of APS Data

Devices and application profiles may have requirements on when APS encryption must or must not be used with normal APS Data. If the device has a set of application data encryption policies, then it shall encrypt any outgoing messages the policy indicates must be protected. It shall also reject any incoming messages that are not APS encrypted when the policy indicates encryption is required.

If a device has requirements on encryption of APS data, it must establish application link keys with partner devices. In a centralized security network the trust center is used to broker this link key establishment. In a distributed security network the partner devices must establish a link key using an application defined method.

4.4.2 Transport-Key Services

The APSME provides services that allow an initiator to transport keying material to a responder. The different types of keying material that can be transported are shown in Table 4.14 to Table 4.17.

4.4.2.1 APSME-TRANSPORT-KEY.request

The APSME-TRANSPORT-KEY.request primitive is used for transporting a key to another device.

4.4.2.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-TRANSPORT-KEY.request     {
                                DestAddress,
                                StandardKeyType,
                                TransportKeyData
                                }
```

Table 4.8 specifies the parameters for the APSME-TRANSPORT-KEY.request primitive.

Table 4.8 APSME-TRANSPORT-KEY.request Parameters

Parameter Name	Type	Valid Range	Description
DestAddress	Device address	Any valid 64-bit address	The extended 64-bit address of the destination device.
StandardKeyType	Integer	0x00 – 0x06	Identifies the type of key material that should be transported; see Table 4.9.
TransportKeyData	Variable	Variable	The key being transported along with identification and usage parameters. The type of this parameter depends on the StandardKeyType parameter as follows: StandardKeyType = 0x01, Standard Network Key see Table 4.11 StandardKeyType = 0x03, Application Link Key see Table 4.12 StandardKeyType = 0x04, Trust Center Link Key, see Table 4.10

Table 4.9 StandardKeyType Parameter of the Transport-Key, Verify-Key, and Confirm-Key Primitives

Enumeration	Value	Description
Reserved	0x00	Reserved
Standard network key	0x01	Indicates that the key is a network key to be used in standard security mode
Reserved	0x02	Reserved
Application link key	0x03	Indicates the key is a link key used as a basis of security between two devices.
Trust-Center link key	0x04	Indicates that the key is a link key used as a basis for security between the Trust Center and another device.
Reserved	0x05 – 0xFF	Reserved

Table 4.10 TransportKeyData Parameter for a Trust Center Link Key

Parameter Name	Type	Valid Range	Description
Key	Set of 16 octets	Variable	The Trust Center link key.

Table 4.11 TransportKeyData Parameter for a Network Key

Parameter Name	Type	Valid Range	Description
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a network key by the Trust Center and used to distinguish network keys for purposes of key updates and incoming frame security operations.
NetworkKey	Set of 16 octets	Variable	The network key.
UseParent	Boolean	TRUE | FALSE	This parameter indicates if the destination device's parent shall be used to forward the key to the destination device: TRUE = Use parent FALSE = Do not use parent
ParentAddress	Device address	Any valid 64-bit address	If the UseParent is TRUE, then ParentAddress parameter shall contain the extended 64-bit address of the destination device's parent device; otherwise, this parameter is not used and need not be set.

Table 4.12 TransportKeyData Parameter for an Application Link Key

Parameter Name	Type	Valid Range	Description
PartnerAddress	Device address	Any valid 64-bit address	The extended 64-bit address of the device that was also sent this link key.
Key	Set of 16 octets	Variable	The application link key

4.4.2.1.2 When Generated

The ZDO on an initiator device shall generate this primitive when it requires a key to be transported to a responder device.

4.4.2.1.3 Effect on Receipt

The receipt of an APSME-TRANSPORT-KEY.request primitive shall cause the APSME to create a transport-key command packet (see section 4.4.9.2). If the StandardKeyType parameter is 0x04 (that is, Trust Center link key), the key descriptor field of the transport-key command shall be set as follows:

- The key sub-field shall be set to the Key sub-parameter of the TransportKeyData parameter.
- The destination address sub-field shall be set to the DestinationAddress parameter.
- The source address sub-field shall be set to the local device address.

This command frame shall be security-protected as specified in section 4.4.1. Then, if security processing succeeds, it is sent to the device specified by the ParentAddress sub-parameter of the TransportKeyData parameter by issuing a NLDE-DATA.request primitive.

If the DestinationAddress parameter is all zeros, the secured command frame shall be unicast to any and all rx-off-when-idle children of the device. These unicasts shall be repeated until successful, or a subsequent APSME-TRANSPORT-KEY.request primitive with the StandardKeyType parameter equal to 0x01 has been received, or a period of twice the recommended maximum polling interval has passed.

If the StandardKeyType parameter is 0x01 (that is, a network key), the key descriptor field of the transport-key command shall be set as follows:

- The key sub-field shall be set to the Key sub-parameter of the TransportKeyData parameter.
- The sequence number sub-field shall be set to the KeySeqNumber sub-parameter of the TransportKeyData parameter.
- The destination address sub-field shall be set to the DestinationAddress parameter.
- The source address sub-field shall be set to the local device address.

This command frame shall be security-protected as specified in section 4.4.1.1 and then, if security processing succeeds, sent to the device specified by the ParentAddress sub-parameter of the TransportKeyData parameter (if the UseParent sub-parameter of the TransportKeyData parameter is TRUE) or the DestinationAddress parameter (if the UseParent sub-parameter of the TransportKeyData parameter is FALSE) by issuing a NLDE-DATA.request primitive.

If the StandardKeyType parameter is 0x03 (that is, an application link key), the key descriptor field of the transport-key command shall be set as follows:

- The key sub-field shall be set to the Key sub-parameter of the TransportKeyData parameter.
- The partner address sub-field shall be set to the PartnerAddress sub-parameter of the TransportKeyData parameter.
- The initiator sub-field shall be set 1 (if the Initiator sub-parameter of the TransportKeyData parameter is TRUE) or 0 (if the Initiator sub-parameter of the TransportKeyData parameter is FALSE).

This command frame shall be security-protected as specified in sub- clause 4.4.1.1 and then, if security processing succeeds, sent to the device specified by the DestinationAddress parameter by issuing a NLDE-DATA.request primitive.

4.4.2.2 APSME-TRANSPORT-KEY.indication

The APSME-TRANSPORT-KEY.indication primitive is used to inform the ZDO of the receipt of keying material.

4.4.2.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-TRANSPORT-KEY.indication      {
                                    SrcAddress,
```



```
StandardKeyType,
TransportKeyData
}
```

Table 4.13 specifies the parameters of the APSME-TRANSPORT-KEY.indication primitive.

Table 4.13 APSME-TRANSPORT-KEY.indication Parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that is the original source of the transported key.
StandardKeyType	Octet	0x00 – 0x06	Identifies the type of key material that was be transported; see Table 4.9.
TransportKeyData	Variable	Variable	The key that was transported along with identification and usage parameters. The type of this parameter depends on the StandardKeyType parameter as follows: StandardKeyType = 0x01 see Table 4.11 StandardKeyType = 0x03 see Table 4.12 StandardKeyType = 0x04 see Table 4.10

4.4.2.2.2 When Generated

The APSME shall generate this primitive when it receives a transport-key command as specified in section 4.4.3.3.

4.4.2.2.3 Effect on Receipt

Upon receipt of this primitive, the ZDO is informed of the receipt of the keying material.

4.4.2.3 Upon Receipt of a Transport-Key Command

Upon receipt of a transport-key command, the APSME shall execute security processing as specified in, then check the key type sub-field.

Upon receipt of a secured transport-key command, the APSME shall check the key type sub-field. If the key type field is set to 0x03 or 0x04 (that is, application link or Trust Center link key) and the receiving device is operating in the joined and authorized state and the command was not secured using a distributed security link key or a Trust Center link key, the command shall be discarded. If the device is operating in the joined and authorized state it may accept a NWK broadcast transport key command with Key type field set to 0x01 (that is, network key) where the message has no APS encryption. If the key type field is set to 0x01 (that is, network key) and the command was not secured using a distributed security link key, Trust Center link key, the command shall be discarded.

If the key type field is set to 0x03 (that is, application link key), the APSME shall issue the APSME-TRANSPORT-KEY.indication primitive with: the SrcAddress parameter set to the source of the key-transport command (as indicated by the NLDE-DATA.indication SrcAddress parameter), and the StandardKeyType parameter set to the key type field. The TransportKeyData parameter shall be set as follows:

- The Key sub-parameter shall be set to the key field.
- The PartnerAddress sub-parameter shall be set to the partner address field.

- The Initiator parameter shall be set to TRUE, if the initiator field is 1. Otherwise it shall be set to 0.

If the key type field is set to 0x01 or 0x04, (that is, Trust Center link key, or a network key) and the destination field is equal to the local address, or if the key type field is set to 0x01 (that is, a network key), the destination field is the all-zero string, and the current network key type is equal to the value of the key type field, the APSME shall issue the APSME-TRANSPORT-KEY.indication primitive with the SrcAddress parameter set to the source address field of the key-transport command and the StandardKeyType parameter set to the key type field. The TransportKeyData parameter shall be set as follows: the Key sub-parameter shall be set to the key field and, in the case of a network key (that is, the key type field is set to 0x01), the KeySeqNumber sub-parameter shall be set to the sequence number field.

If the key type field is set to 0x01 (network key) and source address field is set to 0xFFFFFFFFFFFFFFFF this indicates a distributed security network, the APSME shall issue the APSME-TRANSPORT-KEY.indication primitive with the SrcAddress parameter set to the source address field of the key-transport command and the StandardKeyType parameter set to the key type field. The TransportKeyData parameter shall be set as follows: the Key subparameter shall be set to the key field and, in the case of a network key (that is, the key type field is set to 0x01), the KeySeqNumber sub-parameter shall be set to the sequence number field. The *apsTrustCenterAddress* should be set to 0xFFFFFFFFFFFFFFFF indicating a distributed security network.

If the key type field is set to 0x04 or 0x01 (that is, Trust Center link key or network key) and the destination address field is not equal to the local address, the APSME shall send the command to the address indicated by the destination address field by issuing the NLDE-DATA.request primitive with security disabled.

Upon receipt of a secured transport-key command with the key type field set to 0x01, if the destination field is all zeros and the source address field is set to the value of *apsTrustCenterAddress*, the router shall attempt to unicast this transport-key command to any and all rx-off-when-idle children. The router shall continue to do so until successful, or until a subsequent transport-key command with the key type field set to 0x01 or 0x05 has been received, or until a period of twice the recommended maximum polling interval has passed.

4.4.3 Update Device Services

The APSME provides services that allow a device (for example, a router) to inform another device (for example, a Trust Center) that a third device has changed its status (for example, joined or left the network).

4.4.3.1 APSME-UPDATE-DEVICE.request

The APSME shall issue this primitive when it wants to inform a device (for example, a Trust Center) that another device has a status that needs to be updated (for example, the device joined or left the network).

4.4.3.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

APSME-UPDATE-DEVICE.request	{
	DestAddress,
	DeviceAddress,
	Status,
	DeviceShortAddress
	}

Table 4.13 specifies the parameters for the APSME-UPDATE-DEVICE.request primitive.

Table 4.14 APSME-UPDATE-DEVICE.request Parameters

Parameter Name	Type	Valid Range	Description
DestAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that shall be sent the update information.
DeviceAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device whose status is being updated.
Status	Integer	0x00 – 0x07	Indicates the updated status of the device given by the DeviceAddress parameter: 0x00 = Standard device secured rejoin 0x01 = Standard device unsecured join 0x02 = Device left 0x03 = Standard device trust center rejoin 0x04 – 0x07 = Reserved
DeviceShortAddress	Network address	0x0000 - 0xfff7	The 16-bit network address of the device whose status is being updated.

4.4.3.1.2 When Generated

The APSME (for example, on a router or ZigBee coordinator) shall initiate the APSME-UPDATE-DEVICE.request primitive when it wants to send updated device information to another device (for example, the Trust Center).

4.4.3.1.3 Effect on Receipt

Upon receipt of the APSME-UPDATE-DEVICE.request primitive, the device shall first create an update-device command frame (see section 4.4.9.3). The device address field of this command frame shall be set to the DeviceAddress parameter, the status field shall be set according to the Status parameter, and the device short address field shall be set to the DeviceShortAddress parameter. This command frame shall be security-protected as specified in section 4.4.1.1 and then, if security processing succeeds, sent to the device specified in the DestAddress parameter by issuing a NLDE-DATA.request primitive.

4.4.3.2 APSME-UPDATE-DEVICE.indication

This primitive is issued to inform the APSME that it received an update-device command frame.

4.4.3.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-UPDATE-DEVICE.indication        {
                                      SrcAddress,
                                      DeviceAddress,
```



```
Status,
DeviceShortAddress
}
```

Table 4.15 specifies the parameters for the APSME-UPDATE-DEVICE.indication primitive.

Table 4.15 APSME-UPDATE-DEVICE.indication Parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device originating the update-device command.
DeviceAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device whose status is being updated.
Status	Integer	0x00 – 0x07	Indicates the updated status of the device given by the DeviceAddress parameter: 0x00 = Standard device secured rejoin 0x01 = Standard device unsecured join 0x02 = Device left 0x03 = Standard device trust center rejoin 0x04 – 0x07 = Reserved
DeviceShortAddress	Network Address	0x0000-0xffff	The 16-bit network address of the device whose status is being updated.

4.4.3.2.2 When Generated

The APSME shall generate this primitive when it receives an update-device command frame that is successfully decrypted and authenticated, as specified in section 4.4.1.2.

4.4.3.2.3 Effect on Receipt

Upon receipt of the APSME-UPDATE-DEVICE.indication primitive, the APSME will be informed that the device referenced by the DeviceAddress parameter has undergone a status update according to the Status parameter.

4.4.4 Remove Device Services

The APSME provides services that allow a device (for example, a Trust Center) to inform another device (for example, a router) that one of its children should be removed from the network.

These services may be used in distributed network security.

4.4.4.1 APSME-REMOVE-DEVICE.request

The APSME of a device (for example, a Trust Center) shall issue this primitive when it wants to request that a parent device (for example, a router) remove one of its children from the network. For example, a Trust Center can use this primitive to remove a child device that is not authorized to be on the network.

4.4.4.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-REMOVE-DEVICE.request        {
                                   ParentAddress,
                                   ChildAddress
                                   }
```

Table 4.16 specifies the parameters for the APSME-REMOVE-DEVICE.request primitive.

Table 4.16 APSME-REMOVE-DEVICE.request Parameters

Parameter Name	Type	Valid Range	Description
ParentAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that is the parent of the child device that is requested to be removed, or the router device that is requested to be removed.
TargetAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the target device that is requested to be removed. If a router device is requested to be removed, then the *ParentAddress* shall be the same as the *TargetAddress*.

4.4.4.1.2 When Generated

The APSME (for example, on a Trust Center) shall initiate the APSME-REMOVE-DEVICE.request primitive when it wants to request that a parent device (specified by the ParentAddress parameter) remove one of its child devices (as specified by the TargetAddress parameter), or if it wants to remove a router from the network.

If the device being removed is a router then the ParentAddress field shall be set to the EUI64 of that router and the TargetAddress shall be set to the same value.

4.4.4.1.3 Effect on Receipt

Upon receipt of the APSME-REMOVE-DEVICE.request primitive the device shall first create a remove-device command frame (see section 4.4.9.3). The address field of this command frame shall be set to the TargetAddress parameter. If the device to be removed is a router the ParentAddress and TargetAddress shall be the same. This command frame shall be security-protected as specified in section 4.4.1.1 and then, if security processing succeeds, sent to the device specified by the ParentAddress parameter by issuing a NLDE-DATA.request primitive.

4.4.4.2 APSME-REMOVE-DEVICE.indication

The APSME shall issue this primitive to inform the ZDO that it received a remove-device command frame.

4.4.4.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-REMOVE-DEVICE.indication      {
                                    SrcAddress,
                                    ChildAddress
                                    }
```

Table 4.17 specifies the parameters for the APSME-REMOVEDEVICE.indication primitive.

Table 4.17 APSME-REMOVE-DEVICE.indication Parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Ad-dress	Any valid 64-bit address	The extended 64-bit address of the device requesting that a child device be removed.
TargetAddress	Device Ad-dress	Any valid 64-bit address	The extended 64-bit address of the target device that is requested to be removed.

4.4.4.2.2 When Generated

The APSME shall generate this primitive when it receives a remove-device command frame that is successfully decrypted and authenticated, as specified in section 4.4.1.2.

4.4.4.2.3 Effect on Receipt

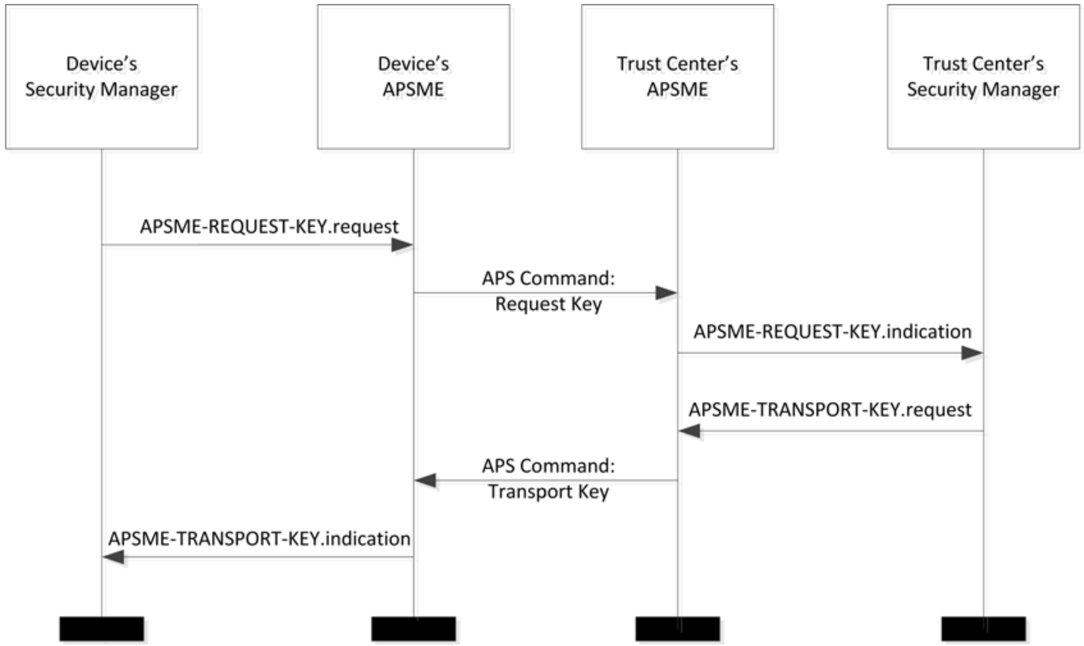
Upon receipt of the APSME-REMOVE-DEVICE.indication primitive the ZDO shall be informed that the device referenced by the TargetAddress parameter shall be removed from the network.

It shall generate an NLME-LEAVE.request and process it as described in 3.2.2.16.

4.4.5 Request Key Services

The APSME provides services that allow a non-trust center device to request an application or trust center link key from the Trust Center. Figure 4.4 shows the processing for the request key services.

Figure 4.4 Request Key Service Processing for Trust Center Link Key



4.4.5.1 APSME-REQUEST-KEY.request

This primitive allows the Security Manager to request a new trust center link key or a new end-to-end application link key.

4.4.5.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-REQUEST-KEY.request        {
                                 DestAddress,
                                 RequestKeyType,
                                 PartnerAddress
                                 }
```

Table 4.18 specifies the parameters for the APSME-REQUEST-KEY.request primitive.

Table 4.18 APSME-REQUEST-KEY.request Parameters

Parameter Name	Type	Valid Range	Description
DestAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device to which the request-key command should be sent.

RequestKeyType	Octet	0x02 and 0x04	The type of key being requested. See Table 4.19.
PartnerAddress	Device Address	Any valid 64-bit address	If the RequestKeyType parameter indicates an application key, this parameter shall indicate an extended 64-bit address of a device that shall receive the same key as the device requesting the key.

Table 4.19 describes the values of the RequestKeyType enumeration. Please note that this enumeration is different than the one for the StandardKeyType in Table 4.9.

Table 4.19 RequestKeyType Values

Value	Enumeration
0x00	Reserved
0x01	Reserved
0x02	Application Link Key
0x03	Reserved
0x04	Trust Center Link Key
0x05 – 0xFF	Reserved

4.4.5.1.2 When Generated

The Security Manager of a device shall generate the APSME-REQUEST-KEY.request primitive when it requires either a new end-to-end application link key or trust center link key. An application link key with the Trust Center is also known as a Trust Center Link Key.

4.4.5.1.3 Effect on Receipt

Upon receipt of the APSME-REQUEST-KEY.request primitive, the device shall first create an APS request-key command frame (see section 4.4.9.5). The RequestKeyType field of this command frame shall be set to the same value as the RequestKeyType parameter. If the RequestKeyType parameter is 0x02 (that is, an application link key), then the partner address field of this command frame shall be the PartnerAddress parameter. Otherwise, the partner address field of this command frame shall not be present.

This command frame shall be security-protected as specified in section 4.4.1.1 and then, if security processing succeeds, sent to the device specified by the DestAddress parameter by issuing a NLDE-DATA.request primitive.

4.4.5.2 APSME-REQUEST-KEY.indication

The APSME shall issue this primitive to inform the Security Manager that it received a request-key command frame.

4.4.5.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

APSME-REQUEST-KEY.indication	{
	SrcAddress,
	RequestKeyType,
	PartnerAddress
	}

Table 4.20 specifies the parameters for the APSME-REQUEST-KEY.indication primitive.

Table 4.20 APSME-REQUEST-KEY.indication Parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that sent the request-key command.
RequestKeyType	Octet	See Description.	The type of key being requested. See Table 4.19 for a list of types and valid values.
PartnerAddress	Device Address	Any valid 64-bit address	If the RequestKeyType parameter indicates an application key, this parameter shall indicate an extended 64-bit address of a device that shall receive the same key as the device requesting the key.

4.4.5.2.2 When Generated

The APSME shall generate this primitive when it receives a request-key command frame that is successfully decrypted and authenticated, as specified in section 4.4.1.2.

4.4.5.2.3 Effect on Receipt

Upon receipt of the APSME-REQUEST-KEY.indication primitive, the following shall be done:

1. If the device is not the Trust Center, the request shall be silently dropped and no further processing shall take place.
2. If the apsTrustCenterAddress of the AIB is 0xFFFFFFFFFFFFFFFF (indicating a distributed security network), the request shall be silently dropped and no further processing shall take place.

3. If the RequestKeyType is 0x04, Trust Center Link Key, then follow the procedure in section 4.7.3.6.
4. If the RequestKeyType is 0x02, Application Link Key, then follow the procedure in section 4.7.3.8.
5. If the RequestKeyType is any other value, the request shall be silently dropped and no further processing shall take place.

4.4.6 Switch Key Services

The APSME provides services that allow the Trust Center to inform another device that it should switch to a new active network key.

4.4.6.1 APSME-SWITCH-KEY.request

This primitive allows a device (for example, the Trust Center) to request that another device or all devices switch to a new active network key.

4.4.6.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-SWITCH-KEY.request      {
                              DestAddress,
                              KeySeqNumber
                              }
```

Table 4.21 specifies the parameters for the APSME-SWITCH-KEY.request primitive.

Table 4.21 APSME-SWITCH-KEY.request Parameters

Parameter Name	Type	Valid Range	Description
DestAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device to which the switch-key command is sent. This may be the broadcast address 0xFFFFFFFFFFFFFFFF.
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a network key by the Trust Center and used to distinguish network keys.

4.4.6.1.2 When Generated

The ZDO of a device (for example, the Trust Center) shall generate the APSME-SWITCH-KEY.request primitive when it wants to inform a device or all devices to switch to a new active network key.

4.4.6.1.3 Effect on Receipt

Upon receipt of the APSME-SWITCH-KEY.request primitive, the device shall first create a switch-key command frame (see section 4.4.9.6). The sequence number field of this command frame shall be set to the same value as the KeySeqNumber parameter.

If the DestAddress is not the broadcast address 0xFFFFFFFFFFFFFFFF, this command frame shall be security-protected as specified in section 4.4.1.1 and then, if security processing succeeds, sent to the device specified by the DestAddress parameter by issuing a NLDE-DATA.request primitive.

If the DestAddress is the broadcast address 0xFFFFFFFFFFFFFFFF then the command shall not be security protected at the APS layer. It shall be sent to the NWK broadcast address 0xFFFD by issuing a NLDE-DATA.request primitive.

4.4.6.2 APSME-SWITCH-KEY.indication

The APSME shall issue this primitive to inform the ZDO that it received a switch-key command frame.

4.4.6.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

APSME-SWITCH-KEY.indication	{
	SrcAddress,
	KeySeqNumber
	}

Table 4.22 specifies the parameters for the APSME-SWITCH-KEY.indication primitive.

Table 4.22 APSME-SWITCH-KEY.indication Parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that sent the switch-key command.
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a network key by the Trust Center and used to distinguish network keys.

4.4.6.2.2 When Generated

The APSME shall generate this primitive when it receives a switch-key command frame that is successfully decrypted and authenticated, as specified in section 4.4.1.2.

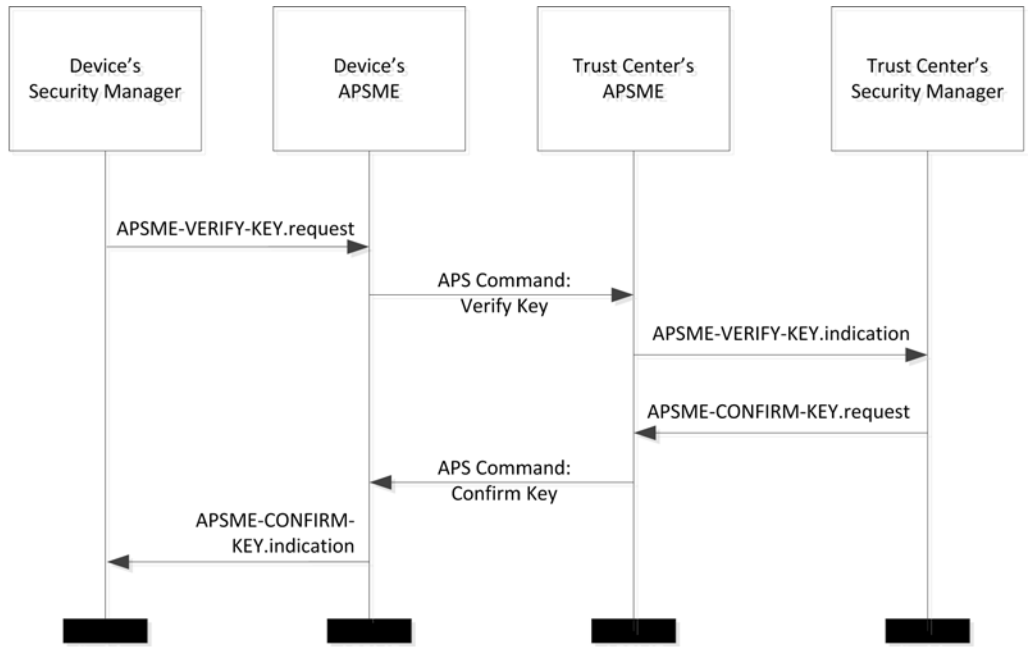
4.4.6.2.3 Effect on Receipt

Upon receipt of the APSME-SWITCH-KEY.indication primitive the ZDO shall be informed that the device referenced by the SrcAddress parameter is requesting that the network key referenced by the KeySeqNumber parameter become the new active network key.

4.4.7 [1]Verify-Key Services

Figure 4.5 illustrates the flow of service requests and the over-the-air messages for the verify key.

Figure 4.5 Verify-Key Processing



4.4.7.1 APSME-VERIFY-KEY.request

This primitive allows a device to request that the partner device verify the Link Key between the two devices.

4.4.7.1.1 Semantics of the Service Primitive

The primitive shall provide the following interface:

APSME-VERIFY-KEY.request	{
	DestAddress,
	StandardKeyType
	}

Table 4.23 specifies the parameters of the APSME-VERIFY-KEY.request primitive.

[1] Note: This is moved text. Moved to section 4.4.9.

Table 4.23 APSME-VERIFY-KEY.request Parameters

Parameter Name	Type	Valid Range	Description
DestAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device to which the verify-key command be sent.
StandardKeyType	Octet	0x00-0xFF	Type of key being verified. See Table 4.9.

4.4.7.1.2 When Generated

The Security Manager on an initiator device shall generate this primitive when it wants to verify its Trust Center link key with the Trust Center.

4.4.7.1.3 Effect on Receipt

On receipt of the APSME-VERIFY-KEY.request primitive the following shall be performed:

1. If the local device is the Trust Center, the request is invalid and no further processing shall be done.
2. If the StandardKeyType parameter is not equal to 0x04 (Trust Center Link Key), the request is invalid. No further processing shall be done.
3. If the apsTrustCenterAddress of the AIB is 0xFFFFFFFFFFFFFFFF (indicating a distributed security network), then the request is invalid. No further processing shall be done.
4. If the DestAddress parameter is not equal to the apsTrustCenterAddress of the AIB, then the request is invalid. No further processing shall be done.
5. The device shall find the corresponding entry in the apsDeviceKeyPairSet that has a DeviceAddress equal to the apsTrustCenterAddress of AIB. If no entry can be found, the operation has failed and no further processing shall be done.
6. The *Initiator Verify-Key Hash Value* shall be calculated according to section 4.5.3 using the LinkKey value of the corresponding apsDeviceKeyPairSet entry found in step 5.
7. The APSME shall generate an APS Command Verify-Key setting the StandardKeyType in the command to the StandardKeyType of this primitive, and setting the Hash value to the calculated Initiator Verify-Key Hash Value. The APS command shall not be APS encrypted.

4.4.7.2 APSME-VERIFY-KEY.indication

This primitive allows a Trust Center to be notified when a device is requesting to verify its Trust Center Link Key. It allows the Trust Center to know when a provisional link key has been replaced by a verified link key.

4.4.7.2.1 Semantics of the Service Primitive

The primitive shall provide the following interface:

```
APSME-VERIFY-KEY.indication            {
```

```
                                     SrcAddress,
                                     StandardKeyType,
                                     ReceivedInitiatorHashValue
                                     }
```

Table 4.24 specifies the parameters of the APSME-VERIFY-KEY.indication primitive.

Table 4.24 APSME-VERIFY-KEY.indication Parameters

Parameter Name	Type	Valid Range	Description
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that sent the verify-key command.
StandardKeyType	Octet	0x00-0xFF	Type of key being verified. See Table 4.9.
ReceivedInitiatorHashValue	Set of 16 octets	Variable	The initiator hash of the key being verified.

4.4.7.2.2 When Generated

The APSME shall generate this primitive when it receives an APS Command Verify Key.

4.4.7.2.3 Effect on Receipt

On receipt of the APSME-VERIFY-KEY.indication primitive the following shall be performed:

1. If the message is a NWK broadcast, the request shall be dropped and no further processing shall be done.
2. If the device is not the Trust Center, this is not a valid request. The device shall follow the procedure in section 4.4.7.2.3.1 setting the Status value to 0xa3 (ILLEGAL_REQUEST). No further processing shall be done.
3. If the StandardKeyType parameter is not equal to 0x04 (Trust Center Link Key), the request is invalid. The device shall follow the procedure in section 4.4.7.2.3.1 setting the Status value to 0xaa (NOT_SUPPORTED). No further processing shall be done.
4. If the apsTrustCenterAddress of the AIB is set to 0xFFFFFFFFFFFFFFFF, the device is operating in distributed Trust Center mode and this is not a valid request. The device shall follow the procedure in section 4.4.7.2.3.1 setting the Status value to 0xaa (NOT_SUPPORTED). No further processing shall be done.
5. The device shall find the corresponding entry in the *apsDeviceKeyPairSet* attribute of the AIB where the DeviceAddress matches the SrcAddress of this primitive and the KeyAttributes is UNVERIFIED_KEY (0x01) or VERIFIED_KEY (0x02). If no entry matching those criteria is found, the following shall be performed.
    a. The Security Manager shall follow the procedure in section 4.4.7.2.3.1 setting the Status value to 0xad (SECURITY_FAILURE).
    b. No further processing shall be done.

6. The device shall calculate the CalculatedInitiatorHashValue by using the LinkKey value in the corresponding *apsDeviceKeyPairSet* entry and the *Initiator Verify-Key Hash Value* cryptographic operation described in section 4.5.3.
7. The device shall compare the ReceivedInitiatorHashValue of the primitive with the CalculatedInitiatorHashValue. If the values do not match the operation has failed, the following shall be performed.
   a. The Security Manager shall follow the procedure in section 4.4.7.2.3.1 setting the Status value to 0xad (SECURITY_FAILURE).
   b. No further processing shall be done.
8. The device shall set the KeyAttributes of the corresponding apsDeviceKeyPairSet entry to VERIFIED_KEY (0x02).
9. The device shall attempt to find the entry in the apsDeviceKeyPairSet where the DeviceAddress of the entry matches the SrcAddress of this primitive and the KeyAttributes is set to PROVISIONAL_KEY (0x00).
   a. If an entry is found, that entry shall be deleted from the apsDeviceKeyPairSet. Processing shall continue.
   b. If no entry is found, then processing shall continue.
10. The device shall follow the procedure in section 4.4.7.2.3.1 setting the Status value to 0x00 (SUCCESS).

4.4.7.2.3.1 APSME-VERIFY-KEY.indication Response

The following shall be done when an APSME-VERIFY-KEY.indication indicates a response must be generated. This procedure takes a Status code as a parameter.

An APSME-CONFIRM-KEY.request shall be generated with the following values:

1. The Status code shall be set to the Status code passed to this procedure.
2. The DestAddress shall be set to the SrcAddress of the APSME-VERIFY-KEY.indication.
3. The StandardKeyType shall be set to the StandardKeyType of the APSME-VERIFY-KEY.indication.
4. The message shall be APS encrypted only if the Status code is SUCCESS.

4.4.8 Confirm-Key Services

4.4.8.1 APSME-CONFIRM-KEY.request

This primitive allows a Trust Center to respond to a device requesting to verify its Trust Center Link Key.

4.4.8.1.1 Semantics of the Service Primitive

The primitive shall provide the following interface:

APSME-CONFIRM-KEY.request	{
	Status
	DestAddress,
	StandardKeyType
	}

Table 4.25 specifies the parameters of the APSME-CONFIRM-KEY.request primitive.

Table 4.25 APSME-CONFIRM-KEY.request Parameters

Parameter Name	Type	Valid Range	Description
Status	Integer	0x00 – 0xFF	A value indicating the success or failure of a previous attempt to verify the trust center link key. See Table 2.27.
DestAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that sent the verify-key command.
StandardKeyType	Octet	0x00-0xFF	Type of key being verified. See Table 4.9.

4.4.8.1.2 When Generated

The Security Manager shall generate this primitive when it wants to respond to a previously received APSME-VERIFY-KEY.indication.

4.4.8.1.3 Effect on Receipt

On receipt of the APSME-CONFIRM-KEY.request primitive the following shall be performed:

1. If the device is not the Trust Center, this is not a valid request. The request shall be dropped and no further processing shall be done.
2. If the StandardKeyType parameter is not equal to 0x04 (Trust Center Link Key), the request is invalid. No further processing shall be done.
    a. If the apsTrustCenterAddress of the AIB is set to 0xFFFFFFFFFFFFFFFF, the device is operating in distributed Trust Center mode and this is not a valid request. The request shall be dropped and no further processing shall be done.
3. The device shall find the corresponding entry in the *apsDeviceKeyPairSet* attribute of the AIB by examining the DeviceAddress of all entries and comparing it to the DestAddress of this primitive. If no match is found, the request is invalid.
    a. The device shall send an APS Command Confirm Key Response to the DestAddress setting the StandardKeyType to the StandardKeyType of this primitive, the Status in the Command to FAILURE. The APS Command shall not be APS encrypted.
    b. No further processing shall be done.
4. The device shall send an APS Command Confirm Key Response to the DestAddress setting the StandardKeyType to the StandardKeyType of this primitive, the Status in the Command to the Status passed to this primitive. The APS Command shall be APS encrypted.
5. The device shall set the IncomingFrameCounter of the apsDeviceKeyPairSet entry to 0.

4.4.8.2 APSME-CONFIRM-KEY.indication

This primitive notifies a device of the result of a previous APSME-VERIFY-KEY.request and allows it to remove a provisional link key used for joining.

4.4.8.2.1 Semantics of the Service Primitive

The primitive shall provide the following interface:

```
APSME-CONFIRM-KEY.indication    {
                                Status
                                SrcAddress,
                                StandardKeyType,
                                }
```

Table 4.26 specifies the parameters of the APSME-CONFIRM-KEY.indication primitive.

Table 4.26 APSME-CONFIRM-KEY.indication Parameters

Parameter Name	Type	Valid Range	Description
Status	Integer	0x00 – 0xFF	The result of the APSME-VERIFY-KEY.request operation.
SrcAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that sent the verify-key command.
StandardKeyType	Octet	0x00-0xFF	Type of key being verified. See Table 4.9.

4.4.8.2.2 When Generated

The APSME shall generate this primitive when it receives an APS Command Confirm Key.

4.4.8.2.3 Effect on Receipt

On receipt of the APSME-CONFIRM-KEY.indication primitive the following shall be performed:

1. If the message is a NWK broadcast, the request shall be dropped and no further processing shall be done.
2. If the local device is the Trust Center, this primitive is invalid. No further processing shall be done.
3. If the Status parameter is not equal to 0x00 (Success), the operation was unsuccessful. No further processing shall be done.
4. If the StandardKeyType parameter is not equal to 0x04 (Trust Center Link Key), this primitive is invalid. No further processing shall be done.
5. If the apsTrustCenterAddress of the AIB is 0xFFFFFFFFFFFFFFFF (indicating a distributed security network), this primitive is invalid. No further processing shall be done.

6. If the SrcAddress parameter is not the equal to the apsTrustCenterAddress of the AIB, then this primitive shall be silently dropped. No further processing shall be done.
7. The device shall find the corresponding entry in the apsDeviceKeyPairSet of the AIB where the DeviceAddress is equal to the apsTrustCenterAddress of the AIB. If no entry can be found, no further processing shall be done.
8. The device shall set the keyAttributes of the corresponding apsDeviceKeyPairSet entry to 0x02 (VERIFIED_KEY).
9. The device shall set the IncomingFrameCounter of the corresponding apsDeviceKeyPairSet entry to 0.

4.4.9 Secured APDU Frame

The APS layer frame format consists of APS header and APS payload fields (see Figure 4.6). The APS header consists of frame control and addressing fields. When security is applied to an APDU frame, the security bit in the APS frame control field shall be set to 1 to indicate the presence of the auxiliary frame header. The format for the auxiliary frame header is given in section 4.5.1. The format of a secured APS layer frame is shown in Figure 4.6. The auxiliary frame header is situated between the APS header and payload fields.[2]

Figure 4.6 Secured APS Layer Frame Format

<table>
<tr><th>Octets: Variable</th><th>5 or 13</th><th colspan="2">Variable</th></tr>
<tr><td rowspan="2">Original APS header
([B7], clause 7.1)</td><td rowspan="2">Auxiliary
frame header</td><td>Encrypted payload</td><td>Encrypted message integrity code (MIC)</td></tr>
<tr><td colspan="2">Secure frame payload = output of CCM</td></tr>
<tr><td colspan="2">Full APS header</td><td colspan="2">Secured APS payload</td></tr>
</table>

[2] Note: Section 4.4.9 is moved text, not added. Moved from section 4.4.6.3

4.4.10 Command Frames

The APS layer command frame formats are given in this clause.

All APS command frames shall set their APS frame control field as follows:

1. Set the frame type sub-field to 0x01 (Command)
2. Set the delivery-mode sub-field to 0x00 (Unicast) or 0x10 (broadcast)
3. Set the ACK format bit to 0.
4. Set the ACK request bit to 0.
5. Set the extended nonce sub-field to 1.
6. Set the security bit according to section 4.4.1.3 Security Processing of APS Commands.

Command identifier values are shown in Table 4.27.

Table 4.27 Command Identifier Values

Command Identifier	Value
Reserved	0x01
Reserved	0x02
Reserved	0x03
Reserved	0x04
APS_CMD_TRANSPORT_KEY	0x05
APS_CMD_UPDATE_DEVICE	0x06
APS_CMD_REMOVE_DEVICE	0x07
APS_CMD_REQUEST_KEY	0x08
APS_CMD_SWITCH_KEY	0x09
Reserved	0x0A
Reserved	0x0B
Reserved	0x0C
Reserved	0x0D
APS_CMD_TUNNEL	0x0E
APS_CMD_VERIFY_KEY	0x0F

Command Identifier	Value
APS_CMD_CONFIRM_KEY	0x10

4.4.10.1 Transport-Key Commands

The transport-key command frame shall be formatted as illustrated in Figure 4.7. The optional fields of the APS header portion of the general APS frame format shall not be present.

Figure 4.7 Transport-Key Command Frame

Octets: 1	1	1	1	Variable
Frame control	APS counter	APS command identifier	Standard-KeyType	Key descriptor
APS header		Payload		

4.4.10.1.1 Command Identifier Field

The command identifier field shall indicate the transport-key APS command type (APS_CMD_TRANSPORT_KEY, seeTable 4.27).

4.4.10.1.2 StandardKeyType Field

This field is 8 -bits in length and describes the type of key being transported. The different types of keys are enumerated in Table 4.9.

4.4.10.1.3 Key Descriptor Field

This field is variable in length and shall contain the actual (unprotected) value of the transported key along with any relevant identification and usage parameters. The information in this field depends on the type of key being transported (as indicated by the StandardKeyType field — seeTable 4.9) and shall be set to one of the formats described in the following subsections.

4.4.10.1.3.1 Trust Center Link Key Descriptor Field

If the key type field is set to 4, the key descriptor field shall be formatted as shown in Figure 4.8.

Figure 4.8 Trust Center Link Key Descriptor Field in Transport-Key Command

Octets: 16	8	8
Key	Destination address	Source address

The key sub-field shall contain the link key that should be used for APS encryption.

The destination address sub-field shall contain the address of the device which should use this link key.

The source address sub-field shall contain the address of the Trust Center that sent the link key.

4.4.10.1.3.2 Network Key Descriptor Field

If the key type field is set to 1 this field shall be formatted as shown in Figure 4.9.

Figure 4.9 Network Key Descriptor Field in Transport-Key Command

Octets: 16	1	8	8
Key	Sequence number	Destination address	Source address

The key sub-field shall contain a network key.

The sequence number sub-field shall contain the sequence number associated with this network key.

The destination address sub-field shall contain the address of the device which should use this network key.

If the network key is sent to a broadcast address, the destination address subfield shall be set to the all-zero string and shall be ignored upon reception.

The source address field sub-shall contain the address of the device (for example, the Trust Center) which originally sent this network key.

The source address field shall contain 0xFFFFFFFFFFFFFFFF in a distributed security network. This indicates to the receiving device this is a distributed security network with no Trust Center.

4.4.10.1.3.3 Application Link Key Descriptor Field

If the key type field is set to 2 or 3, this field shall be formatted as shown in Figure 4.10.

Figure 4.10 Application Link Key Descriptor in Transport-Key Command

Octets: 16	8	1
Key	Partner address	Initiator flag

The key sub-field shall contain a link key that is shared with the device identified in the partner address sub-field.

The partner address sub-field shall contain the address of the other device that was sent this link key.

The initiator flag sub-field shall be set to 1 if the device receiving this packet requested this key. Otherwise, this sub-field shall be set to 0.

4.4.10.2 Update Device Commands

The APS command frame used for device updates is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present.

The update-device command frame shall be formatted as illustrated in Figure 4.11.

Figure 4.11 Update-Device Command Frame Format

Octets: 1	1	1	8	2	1
Frame control	APS counter	APS command identifier	Device Address	Device short address	Status
APS Header		Payload			

4.4.10.2.1 Command Identifier Field

The command identifier field shall indicate the update-device APS command type (APS_CMD_UPDATE_DEVICE, see Table 4.27).

4.4.10.2.2 Device Address Field

The device address field shall be the 64-bit extended address of the device whose status is being updated.

4.4.10.2.3 Device Short Address Field

The device short address field shall be the 16-bit network address of the device whose status is being updated.

4.4.10.2.4 Status Field

The status field shall be assigned a value as described for the Status parameter in Table 4.14.

4.4.10.3 Remove Device Commands

The APS command frame used for removing a device is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present. The remove-device command frame shall be formatted as illustrated in Figure 4.12.

Figure 4.12 Remove-Device Command Frame Format

Octets: 1	**1**	**1**	**8**
Frame control	APS counter	APS command identifier	Target address
APS Header		Payload	

4.4.10.3.1 Command Identifier Field

The command identifier field shall indicate the remove-device APS command type (APS_CMD_REMOVE_DEVICE, see Table 4.27).

4.4.10.3.2 Target Address Field

The target address field shall be the 64-bit extended address of the device that is requested to be removed from the network.

4.4.10.4 Request-Key Commands

The APS command frame used by a device for requesting a key is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present.

The request-key command frame shall be formatted as illustrated in Figure 4.13.

Figure 4.13 Request-Key Command Frame Format

Octets: 1	**1**	**1**	**1**	**0/8**
Frame control	APS counter	APS command identifier	RequestKeyType	Partner address
APS Header		Payload		

4.4.10.4.1 Command Identifier Field

The command identifier field shall indicate the request-key APS command type (APS_CMD_REQUEST_KEY, see Table 4.27).

4.4.10.4.2 RequestKeyType Field

The key type field shall be set to the key being requested. Note this Key Type is different than the StandardKeyType values used in Table 4.9 for other APS Commands or other APSME primitives. The RequestKeyType field values for the APS Command Request Key are defined in Table 4.19.

4.4.10.4.3 Partner Address Field

When the RequestKeyType field is 2 (that is, an application key), the partner address field shall contain the extended 64-bit address of the partner device that shall be sent the key. Both the partner device and the device originating the request-key command will be sent the key.

When the RequestKeyType field is 4 (that is, a trust center link key), the partner address field will not be present.

4.4.10.5 Switch-Key Commands

The APS command frame used by a device for switching a key is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present.

The switch-key command frame shall be formatted as illustrated in Figure 4.14.

Figure 4.14 Switch-key Command Frame Format

Octets: 1	**1**	**1**	**1**
Frame control	APS counter	APS command identifier	Sequence number
APS Header		Payload	

4.4.10.5.1 Command Identifier Field

The command identifier field shall indicate the switch-key APS command type (APS_CMD_SWITCH_KEY, see Table 4.27).

4.4.10.5.2 Sequence Number Field

The sequence number field shall contain the sequence number identifying the network key to be made active.

4.4.10.6 Tunnel Command

The APS command frame used by a device for sending a command to a device that lacks the current network key is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present. The tunnel-key command frame is sent unsecured.

The tunnel-key command frame shall be formatted as illustrated in Figure 4.15.

Figure 4.15 Tunnel Command Frame Format

Octets:1	**1**	**1**	**8**	**2**	**13**	**Variable**	**4**
Frame control	APS counter	APS command identifier	Destination address	Tunneled APS header	Tunneled auxiliary frame	Tunneled command	Tunneled APS MIC
APS Header		Payload					

4.4.10.6.1 Command Identifier Field

The command identifier field shall indicate the tunnel APS command type (APS_CMD_TUNNEL, see Table 4.27).

4.4.10.6.2 Destination Address

The destination address field shall be the 64-bit extended address of the device that is to receive the tunneled command.

4.4.10.6.3 Tunneled Auxiliary Frame Field

The tunneled auxiliary frame field shall be the auxiliary frame (see section 4.5.1) used to encrypt the tunneled command. The auxiliary frame shall indicate that a link key was used and shall include the extended nonce field.

4.4.10.6.4 Tunneled Command Field

The tunneled command field shall be the APS command frame to be sent to the destination.

4.4.10.7 Verify-Key Command

This APS command is used by a joining device to verify its updated link key with the peer device, such as the Trust Center.

The Verify-Key Command frame is formatted as illustrated in Figure 4.16.

Figure 4.16 Verify-Key Command Frame

Octets:1	**1**	**1**	**1**	**8**	**16**
Frame control	APS counter	APS command identifier	Standard Key Type	Source address	Initiator Verify-Key Hash Value
APS Header		APS Payload			

4.4.10.7.1 Command Identifier Field

The command identifier field shall indicate the verify-key request command type (APS_CMD_VERIFY_KEY, see Table 4.27).

4.4.10.7.2 StandardKeyType Field

This is the type of key being verified. See Table 4.9.

4.4.10.7.3 Source Address

This Source address field shall be the 64-bit extended device of the partner device that the destination shares the link key with.

4.4.10.7.4 Initiator Verify-Key Hash Value

This value is the outcome of executing the specialized keyed hash function specified in section B.1.4 using a key with the 1-octet string ‘0x03’ as the input string. The resulting value shall NOT be used as a key for encryption or decryption.

4.4.10.8 Confirm-Key Command

This APS command is used by a device (such as the trust center) to confirm its updated link key with the peer device.

The Confirm-Key command frame is formatted as illustrated in Figure 4.17.

Figure 4.17 Confirm-Key Command Frame

Octets:1	**1**	**1**	**1**	**1**	**8**
Frame control	APS counter	APS command identifier	Status	StandardKeyType	Destination address
APS Payload		APS Payload			

4.4.10.8.1 Command Identifier Field

The command identifier field shall indicate the Confirm-Key command type (APS_CMD_VERIFY_KEY_RESPONSE, see Table 4.27).

4.4.10.8.2 Status

This will be the 1-byte status code indicating the result of the operation. See Table 2.27.

4.4.10.8.3 StandardKeyType

This is the type of key being verified. See Table 4.9.

4.4.10.8.4 Destination Address

This destination address field shall be the 64-bit extended device of the source address of the Verify-Key message.

4.4.11 Security-Related AIB Attributes

The AIB contains attributes that are required to manage security for the APS layer. Each of these attributes can be read or written using the APSME-GET.request and APSME-SET.request primitives, respectively. The security-related attributes contained in the APS PIB are presented in Table 4.29.

11081 **Table 4.28 AIB Security Attributes**

Attribute	Identifier	Type	Range	Description	Default
apsDeviceKeyPairSet	0xaa	Set of key-pair descriptor entries. See Table 4.39.	Variable	A set of key-pair descriptors containing link keys shared with other devices.	-
apsTrustCenterAddress	0xab	Device address	Any valid 64-bit address	Identifies the address of the device's Trust Center. If this value is 0xFFFFFFFFFFFFFFFF, this means that there is no Trust Center in the network and the network is operating in distributed security mode.	-
apsSecurityTimeOutPeriod	0xac	Integer	0x0000-0xFFFF	The period of time a device will wait for the next expected security protocol frame (in milliseconds).	Defined in stack profile
trustCenterPolicies	0xad	-	Variable	A set of polices encoded in the trust center on how it deals with various security events. See Table 4.32.	

11082

11083 **Table 4.29 Elements of the Key-Pair Descriptor**

Name	Type	Range	Description	Default
DeviceAddress	Device address	Any valid 64-bit address	Identifies the address of the entity with which this key-pair is shared.	-
KeyAttributes	Enumeration	0x00 – 0x02	This indicates attributes about the key. 0x00 = PROVISIONAL_KEY 0x01 = UNVERIFIED_KEY 0x02 = VERIFIED_KEY	

LinkKey	Set of 16 octets	-	The actual value of the link key.	-
OutgoingFrameCounter	Set of 4 octets	0x00000000-0xFFFFFFFF	Outgoing frame counter for use with this link key.	0x00000000
IncomingFrameCounter	Set of 4 octets	0x00000000-0xFFFFFFFF	Incoming frame counter value corresponding to *DeviceAddress.*	0x00000000
apsLinkKeyType	Enumeration	0x00 – 0x01	The type of link key in use. This will determine the security policies associated with sending and receiving APS messages. 0x00 = Unique Link Key 0x01 = Global Link Key	0x00

4.5 Common Security Elements

This clause describes security-related features that are used in more than one ZigBee layer. The NWK and APS layers shall use the auxiliary header as specified in section 4.5.1 and the security parameters specified in section 4.5.2. The formatting of all frames and fields in this specification are depicted in the order in which they are transmitted by the NWK layer, from left to right, where the leftmost bit is transmitted first in time. Bits within each field are numbered from 0 (leftmost and least significant) to k-1 (rightmost and most significant), where the length of the field is k bits. Fields that are longer than a single octet are sent to the next layer in the order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.

4.5.1 Auxiliary Frame Header Format

The auxiliary frame header, as illustrated by Figure 4.18, shall include a security control field and a frame counter field, and may include a sender address field and key sequence number field.

Figure 4.18 Auxiliary Frame Header Format

Octets: 1	**4**	**0/8**	**0/1**
Security control	Frame counter	Source address	Key sequence number

4.5.1.1 Security Control Field

The security control field shall consist of a security level, a key identifier, and an extended nonce sub-field and shall be formatted as shown in Figure 4.19.

Figure 4.19 Security Control Field Format

Bit: 0-2	3-4	5	6-7
Security level	Key identifier	Extended nonce	Reserved

4.5.1.1.1 Security Level Sub-Field

The security level identifier indicates how an outgoing frame is to be secured, how an incoming frame purportedly has been secured; it also indicates whether or not the payload is encrypted and to what extent data authenticity over the frame is provided, as reflected by the length of the message integrity code (MIC). The bit-length of the MIC may take the values 0, 32, 64 or 128 and determines the probability that a random guess of the MIC would be correct. The security properties of the security levels are listed in Table 4.29 Note that security level identifiers are not indicative of the relative strength of the various security levels. Also note that security levels 0 and 4 should not be used for frame security.

Table 4.30 Security Levels Available to the NWK, and APS Layers

Security Level Identifier	Security Level Sub-Field	Security Attributes	Data Encryption	Frame Integrity (length M of MIC, in Number of Octets)
0x00	‘000’	None	OFF	NO (M = 0)
0x01	‘001’	MIC-32	OFF	YES (M=4)
0x02	‘010’	MIC-64	OFF	YES (M=8)
0x03	‘011’	MIC-128	OFF	YES (M=16)
0x04	‘100’	ENC	ON	NO (M = 0)
0x05	‘101’	ENC-MIC-32	ON	YES (M=4)
0x06	‘110’	ENC-MIC-64	ON	YES (M=8)
0x07	‘111’	ENC-MIC-128	ON	YES (M=16)

4.5.1.1.2 Key Identifier Sub-Field

The key identifier sub-field consists of two bits that are used to identify the key used to protect the frame. The encoding for the key identifier sub-field shall be as listed in Table 4.30. Key derivation is described in section 4.5.3.

Table 4.31 Encoding of the Key Identifier Sub-Field

Key Identifier	Key Identifier Sub-Field (Figure 4.19)	Description
0x00	‘00’	A data key.
0x01	‘01’	A network key.
0x02	‘10’	A key-transport key.
0x03	‘11’	A key-load key.

4.5.1.1.3 Extended Nonce Sub-Field

The extended nonce sub-field shall be set to 1 if the sender address field of the auxiliary header is present. Otherwise, it shall be set to 0.

4.5.1.2 Counter Field

The counter field is used to provide frame freshness and to prevent processing of duplicate frames.

4.5.1.3 Source Address Field

The source address field shall only be present when the extended nonce sub-field of the security control field is 1. When present, the source address field shall indicate the extended 64-bit address of the device responsible for securing the frame.

4.5.1.4 Key Sequence Number Field

The key sequence number field shall only be present when the key identifier subfield of the security control field is 1 (that is, a network key). When present, the key sequence number field shall indicate the key sequence number of the network key used to secure the frame.

4.5.2 Security Parameters

This section specifies the parameters used for the CCM security operations.

4.5.2.1 CCM Mode of Operation and Parameters

Applying security to a NWK or APS frame on a particular security level corresponds to a particular instantiation of the AES-CCM mode of operation as specified in section B.1.2.

The nonce shall be formatted as specified in section4.5.2.2.

Table 4.29 gives the relationship between the security level sub-field of the security control field (Figure 4.19), the security level identifier, and the CCM encryption/authentication properties used for these operations.

4.5.2.2 CCM Nonce

The nonce input used for the CCM encryption and authentication transformation and for the CCM decryption and authentication checking transformation consists of data explicitly included in the frame and data that both devices can independently obtain. Figure 4.20 specifies the order and length of the subfields of the CCM nonce. The nonce's security control and frame counter fields shall be the same as the auxiliary header's security control and frame counter fields (as defined in section 4.5.1) of the frame being processed. The nonce's source address field shall be set to the extended 64-bit IEEE address of the device originating security protection of the frame. When the extended nonce sub-field of the auxiliary header's security control field is 1, the extended 64-bit IEEE address of the device originating security protection of the frame shall correspond to the auxiliary header's source address field (as defined in section 4.5.1) of the frame being processed.

Figure 4.20 CCM Nonce

Octets: 8	**4**	**1**
Source address	Frame counter	Security control

4.5.3 Cryptographic Key Hierarchy

The link key established between two (or more) devices is used to determine related secret keys, including data keys, key-transport keys, and key-load keys. These keys are determined as follows:

1. *Key-Transport Key*. This key is the outcome of executing the specialized keyed hash function specified in section B.1.4 under the link key with the 1-octet string '0x00'as the input string.
2. *Key-Load Key*. This key is the outcome of executing the specialized keyed hash function specified in section B.1.4 under the link key with as input string the 1-octet string '0x02'as the input string.
3. *Data Key*. This key is equal to the link key.

All keys derived from the link key shall share the associated frame counters. Also, all layers of ZigBee shall share the active network key and associated outgoing and incoming frame counters.

4.5.4 Implementation Requirements

This clause provides requirements that should be followed to ensure a secure implementation.

4.5.4.1 Random Number Generator

A ZigBee device generating random keys for distribution requires a strong method of random number generation. For example, when link keys are pre-installed (for example, in the factory), a random number may not be needed.

In all cases that require random numbers, it is critical that the random numbers are not predictable or have enough entropy, so an attacker will not be able determine them by exhaustive search. Random number generation shall meet the random number tests specified in FIPS140- 2 [B13]. Methods for generation of random numbers include:

1. Base the random number on random clocks and counters within the ZigBee hardware;
2. Base the random number on random external events;
3. Seed each ZigBee device with a good random number from an external source during production. This random number can then be used as a seed to generate additional random numbers.

A combination of these methods can be used. Since the random number generation is likely integrated into the ZigBee IC, its design — and hence the ultimate viability of any encryption/security scheme — is left up to the IC manufacturers.

4.6 Functional Description

This section provides detailed descriptions of how the security services shall be used in a ZigBee network. A description of the security initialization responsibilities for a device starting a network is given in section 4.6.1. A brief description of the Trust Center application is given in section 4.6.2. Detailed security procedures are given in section 4.6.3.

4.6.1 ZigBee Security Initialization

The device starting a network shall configure the security level of the network by setting the *nwkSecurityLevel* attribute in the NIB. If the *nwkSecurityLevel* attribute is set to zero, the network will be unsecured, otherwise it will be secured.

The *key* value of the *nwkSecurityMaterialSet* attribute shall be set to any non-zero, random number within the range of all possible values. See section 4.5.4.1 for the requirements of random number generation. The *KeySeqNumber* of the nwkSecurityMaterialSet shall be set to 0.

If it is a centralized security network then the device shall configure the address of the Trust Center by setting the AIB attribute *apsTrustCenterAddress*. The device forming the network may also set the apsTrustCenterAddress to 0xFFFFFFFFFFFFFFFF indicating a distributed security network.

4.6.2 Trust Center Application

The Trust Center application runs on a device trusted by devices within a ZigBee network to distribute keys for the purpose of network and end-to-end application configuration management. The Trust Center shall configure network security policies and shall be used to help establish end-to-end application keys. These keys shall be generated at random unless a key establishment protocol is used.

4.6.2.1 Distributed Security Mode

In Distributed Security Mode, there is no unique Trust Center in the network. Keys are distributed to joining devices by routers in the network using the standard transport key commands, or by other out of band methods.

4.6.2.2 Centralized Security Mode

The centralized security mode of the Trust Center is designed for applications where a centralized security device and set of security policies is required. In this mode, the Trust Center may maintain a list of devices, link keys and network keys with all the devices in the network; however, it shall maintain a network key and controls policies of network admittance. In this mode, the *nwkAllFresh* attribute in the NIB shall be set to FALSE.

Each device that joins the network securely shall either have a Global Link key or a unique link key depending upon the application in use. It is required that the trust center have prior knowledge of the value of the link key and the type (Global or unique) in order to securely join the device to the network. A Global Link key has the advantage that the memory required by the Trust Center does not grow with the number of devices in the network. A unique link key has the advantage of being unique for each device on the network and application communications can be secured from other devices on the network. Both types of keys may be used on the network, but a device shall only have one type in use per device-key pair.

The security policy settings for centralized security are further detailed in section 4.7.1.

4.6.3 Security Procedures

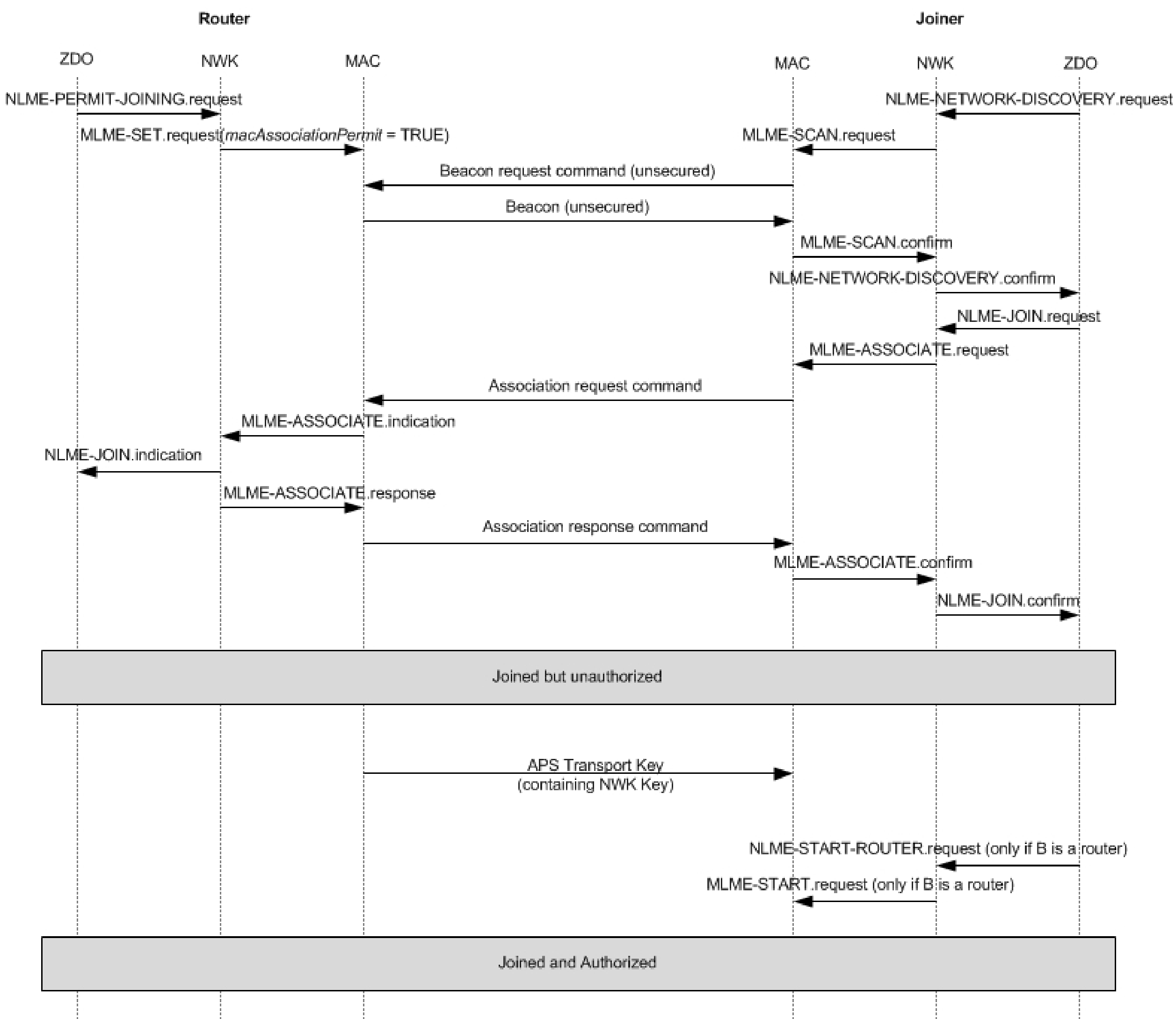
This section gives message sequence charts for joining a secured network, authenticating a newly joined device, updating the network key, recovering the network key, establishing end-to-end application keys, and leaving a secured network.

4.6.3.1 Joining a Secured Network

When a device prepares to join a secured network it shall set the AIB attribute *apsLinkKeyType* for its link key with the trust center according to the kind of key it has. If it is using the default trust center link key, or another Global Link key, it shall set *apsLinkKeyType* to 0x01. If it is using a unique link key it shall set *apsLinkKeyType* to 0x00.

Figure 4.21 shows an example message sequence chart ensuing from when a joiner device communicates with a router device to join a secured network. A device that is operating in a network and has missed a network key update may also use these procedures to receive the latest network key.

Figure 4.21 Example of Joining a Secured Network



The joiner device may begin the join procedure by issuing an NLME-NETWORK- DISCOVERY.request primitive. This primitive will invoke an MLME-SCAN.request primitive which may cause the transmission of an unsecured beacon request frame (depending on whether the scan is an active or passive scan).

The joiner device receives beacons from nearby routers and the NWK layer issues an NLME-NETWORK-DISCOVERY.confirm primitive. The NetworkList parameter of this primitive will indicate all of the nearby PANs. In Figure 4.21, the shown router device has already been placed in a state such that its beacons have the "association permit" sub-field set to "1" (permit association).

The joiner device shall decide which PAN to join and shall issue the NLME-JOIN.request primitive to join that PAN. If the joiner already has a network key for this PAN, the SecurityEnable parameter for the NLME-JOIN.request primitive shall be set to TRUE; otherwise it shall be set to FALSE. As shown in Figure 4.26, the NLME-JOIN.request primitive causes an association request or rejoin request command to be sent to the router.

Upon receipt of an association request MAC command, the router shall issue an MLME-ASSOCIATE.indication primitive. Next, the NWK layer will issue an NLME-JOIN.indication primitive to the router's ZDO. The router shall now know the joiner device's address and security capabilities. The router will also issue an MLME-ASSOCIATE.response. This primitive will cause an association response command to be sent to the joiner.

Alternatively, upon receipt of a rejoin request network command, the NWK layer will issue an NLME-JOIN.indication primitive to the router's ZDO. The router shall now know the joiner device's address, security capabilities, and whether the network key was used to secure the rejoin request command. The router will also issue a rejoin response command to the joiner.

Upon receipt of the association response MAC command or the rejoin response network command, the joiner shall issue the NLME-JOIN.confirm primitive. The joiner is now declared "joined, but unauthorized" to the network. The authorization routine (see section 4.6.3.2) shall follow.

If the joiner is not a router, it is declared "joined and authorized" immediately following the successful completion of the authorization routine.

If the joiner is a router, it is declared "joined and authorization" only after the successful completion of the authorization routine followed by the initiation of routing operations. Routing operations shall be initiated by the joiner's ZDO issuing the NLME-START.request primitive to cause the MLME-START.request primitive to be sent to the MAC layer of the joiner.

If the router refuses the joiner, its association response frame or rejoin response frame shall contain the association status field set to a value other than 0x00, and, after this parameter reaches the ZDO of the joiner in the NLME-JOIN.confirm primitive, the joiner shall not begin the authorization routine.

4.6.3.2 Authorization

Once a device joins a secured network and is declared "joined but unauthorized", it must be authorized by receiving an APS transport key command containing the active network key.

4.6.3.2.1 Router Operation

If the *apsTrustCenterAddress* is all 0xFFFFFFFFFFFFFFFF, this indicates a Distributed Security network. If the apsTrustCenterAddress is any other value, it indicates a Centralized Security network.

In centralized security networks, if the router is not the Trust Center, it shall begin the authorization procedure immediately after receipt of the NLME-JOIN.indication primitive by issuing an APSME-UPDATE-DEVICE.request primitive with the DestAddress parameter set to the *apsTrustCenterAddress* in the AIB and the DeviceAddress parameter set to the address of the newly joined device. The Status parameter of this primitive shall be set by the NLME-JOIN.indication primitive parameters according to Table 4.31.

In a Distributed Security Network no Update Device message is generated. The router shall issue an APSME-TRANSPORT-KEY.request with the StandardKeyType set to 0x01 (Standard Network Key) and the key value from the nwkSecurityMaterialSet of the NIB with a KeySeqNumber equal to the nwkActiveSeqNumber of the NIB. The message shall be APS encrypted with the Distributed Security Global Key in the apsDeviceKeyPairSet.

Table 4.32 Mapping of NLME-JOIN.indication Parameters to Update Device Status

NLME-JOIN.indication Parameters			Update Device Status	
Capability Information Bit 6	Method (RejoinNetwork parameter)	Request Secured	Status	Description
0	NWK Rejoin (0x02)	TRUE	0x00	Standard security device secured rejoin
0	MAC Association (0x00)	FALSE	0x01	Standard security device unsecured join
0	NWK Rejoin (0x02)	FALSE	0x03	Standard security device unsecured rejoin

If the router is the Trust Center, it shall begin the authorization procedure by simply operating as a Trust Center.

The router shall not forward messages to a child device, or respond to ZDO requests or NWK command requests on that child's behalf, while the value of the relationship field entry in the corresponding *nwkNeighborTable* in the NIB is 0x05 (unauthorized child).

4.6.3.2.2 Trust Center Operation

The Trust Center role in the authorization procedure shall be activated upon receipt of an incoming update-device command or immediately after receipt of the NLME-JOIN.indication primitive (in the case where the router is the Trust Center). The Trust Center behaves differently depending on the following factors:

Whether the Trust Center decides to allow any device to perform a first time join (for example, the Trust Center is in a mode that allows new devices to join).

If the Trust Center Policies require prior knowledge of the device to allow joining

If, at any time during the authorization procedure, the Trust Center decides not to allow the new device to join the network (for example, a policy decision or a failed higher level key-establishment protocol), it shall take actions to remove the device from the network. If the Trust Center is not the router of the newly joined device, it may remove the device from the network by issuing the APSME-REMOVE-DEVICE. request primitive with the ParentAddress parameter set to the address of the router originating the update-device command and the ChildAddress parameter set to the address of the joined (but unauthorized) device. Alternatively the Trust Center may let an unauthorized device just timeout; in that case the Trust Center will not send a removal message.

4.6.3.2.2.1 Initial Key Distribution

After being activated for the authorization procedure, the Trust Center shall determine whether or not to allow the device onto the network. This decision will be based on its own security policies, see section 4.7.1. If it decides to allow the device onto the network, it shall send the device the active network key by issuing the APSME-TRANSPORT-KEY.request primitive with the DestAddress parameter set to the address of the newly joined device, and the StandardKeyType parameter set to 0x01 (that is, standard network key).

The KeySeqNumber sub-parameter of the APSME-TRANSPORT-KEY.request shall be set to the sequence count value for the active network key and the NetworkKey sub-parameter shall be set to the active network key. The UseParent sub-parameter shall be set to FALSE if the Trust Center is the router; otherwise, the UseParent sub-parameter shall be set to TRUE and the ParentAddress sub-parameter shall be set to the address of the router originating the update-device command.

4.6.3.2.3 Joining Device Operation

After successfully joining or rejoining a secured network, the joining device shall set the *nwkSecurityLevel* attribute in the NIB to the values indicated by the stack profile.

A joined and authorized device shall always apply NWK layer security to outgoing frames unless the frame is destined for a newly joined but unauthorized child.

In a secured network, if the device does not become authorized within a preconfigured amount of time, it shall leave the network (see section 4.6.3.6.3).

4.6.3.2.3.1 Preconfigured Trust Center Link Key

The joining device shall be preconfigured with a Trust Center link key and wait to receive an active network key encrypted with its preconfigured link key. Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with the StandardKeyType parameter set to 0x01 (that is, the standard network key), the joining device shall set the *apsTrustCenterAddress* attribute in its AIB to the SrcAddress parameter of the APSME-TRANSPORT-KEY.indication primitive. The joining device is now considered authorized and shall enter the normal operating state for standard security mode.

If the *apsTrustCenterAddress* is set to 0xFFFFFFFFFFFFFFFF the network is in distributed security mode. The device shall enter the normal operating state.

Additional application layer security authentication or initializion may be required by the higher layer specification.

If the joining device did not receive the key via the APSME-TRANSPORT-KEY.indication within the *apsSecurityTimeOutPeriod* since receiving the NLME-JOIN.confirm primitive, it shall reset and may choose to start the joining procedure again.

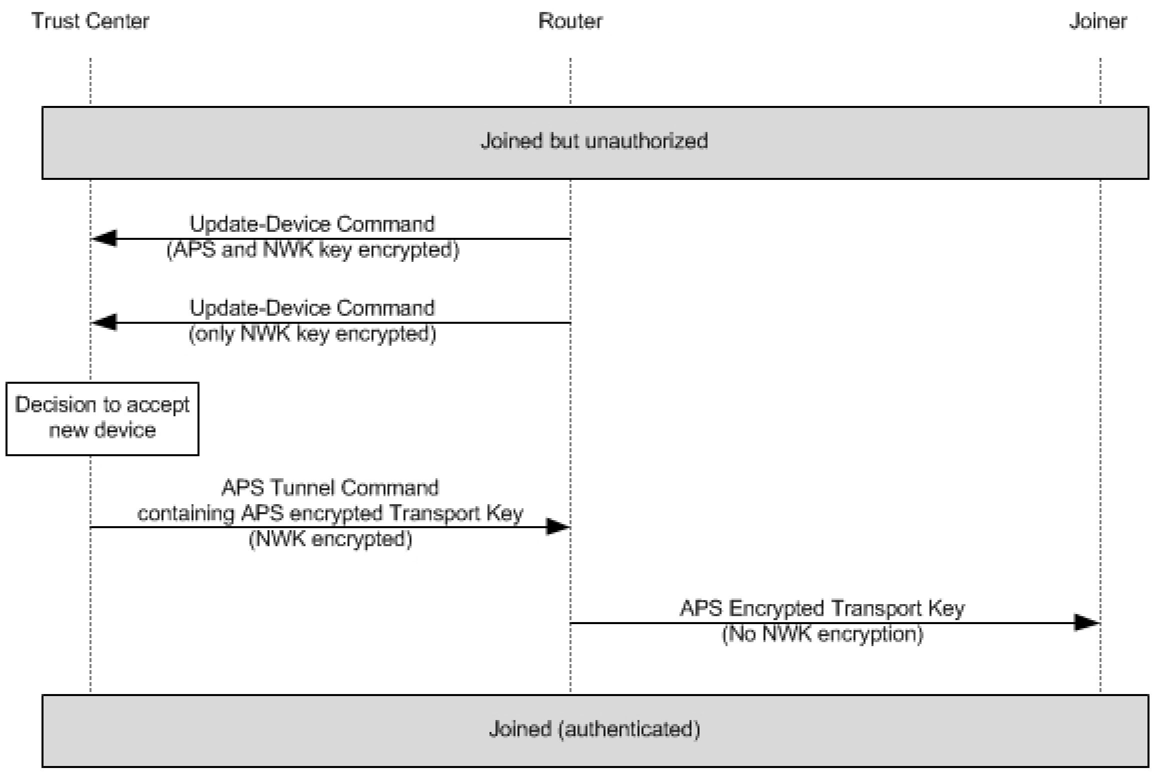
4.6.3.2.4 Message Sequence Charts

Figure 4.22 shows the message sequence charts for the authorization procedure when the joining is not to the Trust Center directly, but through an intermediate router.

The update-device and tunnel commands communicated between the Trust Center and the router shall be secured at the NWK layer by the active network key. The transport-key command sent from the router to the joiner shall not be secured at the network layer. Two copies of the update-device APS command shall be generated by the intermediary router if the apsDeviceKeyPairSet entry for the TC indicates the apsLinkKeyType is 0x01 (Global). One copy shall encrypted at both the APS and the NWK layer, while the other copy shall only be encrypted at the NWK layer. This is done due to an interoperability issue where previously certified Trust Centers may have requirements on the encryption that it accepts for the update device message.

A device with apsDeviceKeyPairSet that has an apsLinkKeyType of 0x00 (Unique Link Key) does not have to generate two update device messages.

Figure 4.22 - Multi-hop Join and Trust Center Rejoin Diagram



4.6.3.3 Rejoining Security

Devices shall follow the procedures described in this section as necessary to support rejoining, in conjunction with the mechanism described in section 2.5.4.5.2.2.

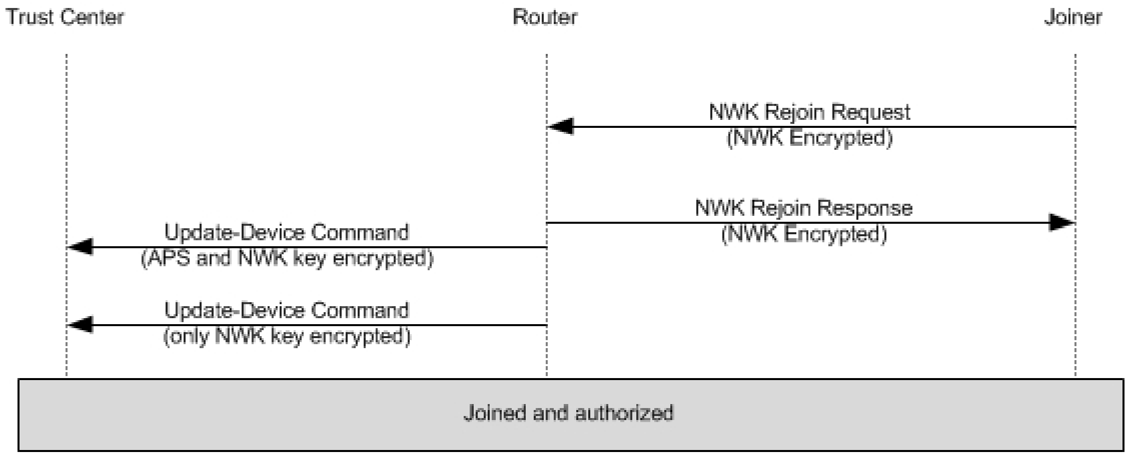
A device does not have to verify its trust center link key with the APSME-VERIFY-KEY services after a rejoin.

4.6.3.3.1 Secure Rejoin

When a device is rejoining and secures the NWK rejoin request command with the active network key, no further authorization is required beyond validation of the NWK security. Both centralized and distributed networks may use Secure Rejoin.

Figure 4.23 shows the flow of messages during a secure rejoin. Note that in Distributed network security the APS Command Update Device shall not be sent.

Figure 4.23 - Secure Rejoin

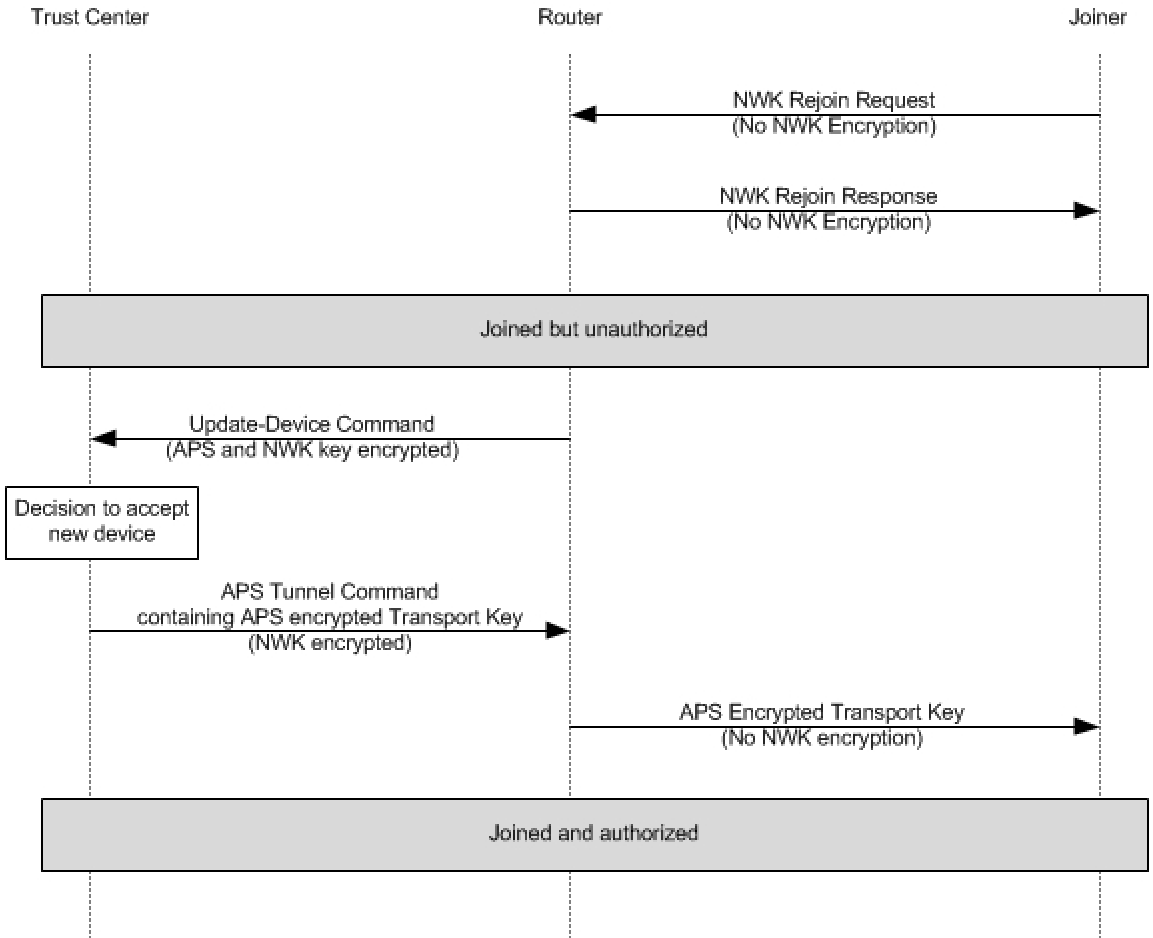


4.6.3.3.2 Trust Center Rejoin

A Trust Center Rejoin is used when a device may no longer have the current network key and therefore should not secure the NWK rejoin command. If the network is using a different network key then the device using the old network key will be rejected. A Trust Center rejoin is a NWK Rejoin command where the command is sent without NWK layer security and allows a device to request the current active network key.

Figure 4.24 illustrates a trust center rejoin operation.

Figure 4.24 - Trust Center Rejoin



A Trust Center Rejoin shall only be allowed in a centralized security network. Attempts to use a Trust Center rejoin in a distributed security network shall be rejected.

The following sections describe the behavior of the devices in the network and the orphaned devices.

4.6.3.3.3 Coordinator and Router Operation

This text describes the security operations for support of rejoining which are to be carried out by the ZigBee coordinator and by ZigBee routers that are already operating on the network. These devices will receive rejoin requests by orphaned devices and will act as follows.

Following the steps described in section 2.5.4.5.2.2, an orphaned device (router or end device) shall be provisionally accepted onto the network by the coordinator or router for at least *apsSecurityTimeOutPeriod* milliseconds. During this period it shall be required to send at least one correctly formed ZigBee message secured with the network key to the new parent. If this message successfully passes all the security processing steps described in this document, it shall be accepted as a member of the network.

This specification neither specifies nor requires any action from the router or coordinator in the case that a message from an orphaned device fails security processing above that required by text elsewhere in this document.

4.6.3.3.4 Rejoining Device Operation

Following the steps described in section 2.5.4.5.2.2, an orphaned device (router or end device) shall be provisionally accepted onto the network for at least *apsSecurityTimeOutPeriod* milliseconds. During this period, it shall be required to send at least one ZigBee message, secured with the network key to the new parent.

As normal, the device shall not accept an unsecured network key (having no NWK security) from the Trust Center.

Note that a ZigBee device may also carry out an orphan scan as described in section 2.5.5.5.2.2. In this case it shall, at this time, also follow the steps described in this sub-section.

4.6.3.4 Network Key Update

The Trust Center and network devices shall follow the procedures described in this section when updating the active network key. Updating of the network key is not possible when operating in distributed security mode.

4.6.3.4.1 Trust Center Operation

When updating a standard network key with a new key of the same type, the Trust Center may broadcast or unicast the key update. If it chooses to broadcast the new key to all devices on the network it issues the APSME-TRANSPORT-KEY.request primitive with the DestAddress parameter set to the broadcast address and the StandardKeyType parameter set to 0x01 (that is, a network key).

For a unicast key update the Trust Center shall issue multiple APSME-TRANSPORT-KEY.request primitive with the DestAddress set to each device it wants to notify of the new key.

The TransportKeyData sub-parameters shall be set as follows for both unicast and broadcast key updates:

- The KeySeqNumber sub-parameter shall be set to the sequence count value for the new network key.
- The NetworkKey sub-parameter shall be set to the new network key.
- The UseParent sub-parameter shall be set to FALSE.

If the sequence count for the previously distributed network key is represented as N, then the sequence count for this new network key shall be $(N+1)$ mod 256.

The Trust Center may cause a switch to this new key by issuing the APSME-SWITCH-KEY.request primitive with the DestAddress parameter set to the broadcast address and the KeySeqNumber parameter set to the sequence count value for the updated network key. The switch key shall not be unicast. It shall be encrypted at the network layer with the current network key.

In centralized security mode, the Trust Center may maintain a list of all of the devices in the network. To update the active network key using this list, the Trust Center may first send the new network key to each device on this list and then ask the network to switch to the new key.

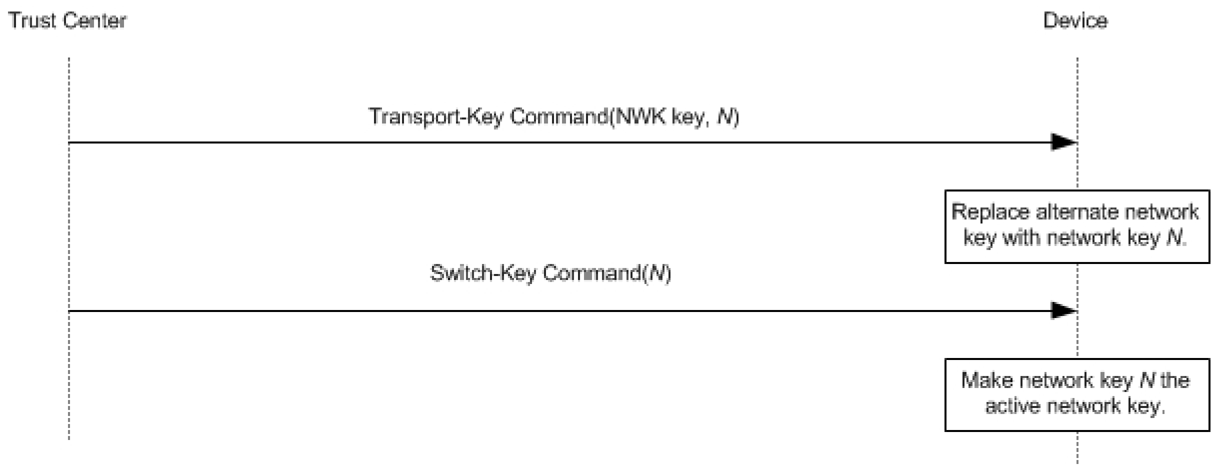
4.6.3.4.2 Network Device Operation

Devices shall be capable of storing 2 network keys, the current and an alternate.

When in the normal operating state and upon receipt of a APSME-TRANSPORT-KEY.indication primitive with the StandardKeyType parameter set to 0x01 (that is, a network key), a device shall accept the TransportKeyData parameters as a network key only if the SrcAddress parameter is the same as the Trust Center's address (as maintained in the *apsTrustCenterAddress* attribute of the AIB). If accepted, the key and sequence number data contained in the TransportKeyData parameter shall replace the alternate network key. Otherwise, the key and sequence number data contained in the TransportKeyData parameter shall replace the active network key. In either case, all incoming frame counters and the outgoing frame counter of the appropriate network key shall be set to 0.

When in the normal operating state and upon receipt of an APSME-SWITCH-KEY.indication primitive, a device shall switch its active network key to the one designated by the KeySeqNumber parameter only if the SrcAddress parameter is the same as the Trust Center's address (as maintained in the *apsTrustCenterAddress* attribute of the AIB). Figure 4.25 illustrates the procedure.

Figure 4.25 Example Network Key-Update Procedure



4.6.3.4.3 Message Sequence Chart

An example of a successful network key-update procedure for two devices is shown in Figure 4.25. In this example, the Trust Center sends a network key with sequence number *N* to the device. All devices are required to be capable of storing two network keys, an active and alternate. Upon receipt of the transport-key command, the device replaces its alternate network key with the new network key. Next, upon receipt of the switch-key command, the device makes the new network key the active network key.

4.6.3.5 End-to-End Application Key Establishment

An initiator device, a Trust Center, and a responder device shall follow the procedures described in this section when establishing a link key for purposes of end-to-end application security between initiator and responder devices.

4.6.3.5.1 Device Operation

The initiator device shall begin the procedure to establish a link key with a responder device by issuing the APSME-REQUEST-KEY.request primitive. The DestAddress parameter shall be set to the address of its Trust Center, the RequestKeyType parameter shall be set to 0x02 (that is, application link key), and the PartnerAddress parameter shall be set to the address of the responder device.

In a distributed security network where there is not a trust center to authorize the distribution of application link keys, an initiator device may issue an APSME-TRANSPORT-KEY.request to a responder device based on application policies on the device.

4.6.3.5.1.1 Upon Receipt of a Link Key

Upon receipt of an APSME-TRANSPORT-KEY.indication primitive with the StandardKeyType parameter set to 0x03 (that is, application link key), a device may accept the TransportKeyData parameters as a link key with the device indicated by the PartnerAddress parameter only if the SrcAddress parameter is the same as the *apsTrustCenterAddress* attribute of the AIB. If accepted, the *apsDeviceKeyPairSet* attribute in AIB table will be updated. A key-pair descriptor in the AIB shall be created (or updated if already present) for the device indicated by the PartnerAddress parameter, by setting the DeviceAddress element to the PartnerAddress parameter, the LinkKey element to the link key from the TransportKeyData parameter, and the OutgoingFrameCounter and IncomingFrameCounter elements to 0 unless the value is the same as the previous link key.

In the case of a distributed security network, a device may accept an APSME-TRANSPORT-KEY.indication primitive with the StandardKeyType parameter set to 0x03 (that is, application link key) from a partner device since no trust center exists. The device and this partner can then establish an application link key based on the application level policies of the device.

4.6.3.5.2 Trust Center Operation

Upon receipt of APSME-REQUEST-KEY.indication primitives with the StandardKeyType parameter set to 0x02 (that is, application link key).

The Trust Center shall issue two APSME-TRANSPORT-KEY.request primitives with the StandardKey-Type parameter shall be set to 0x03 (that is, application link key). The first primitive shall have the DestAddress parameter set to the address of the device requesting the key. The TransportKeyData sub-parameters shall be set as follows:

- The PartnerAddress sub-parameter shall be set to the PartnerAddress sub-parameter of the APSME-REQUEST-KEY.indication primitive's TransportKeyData parameter.
- The Initiator sub-parameter shall be set to TRUE.
- The Key sub-parameter shall be set to a new key K (link key).

The key shall have been generated in a random fashion. The second primitive shall have the DestAddress parameter set to the PartnerAddress sub-parameter of the APSME-REQUEST-KEY.indication primitive's TransportKeyData parameter. The TransportKeyData sub-parameters shall be set as follows:

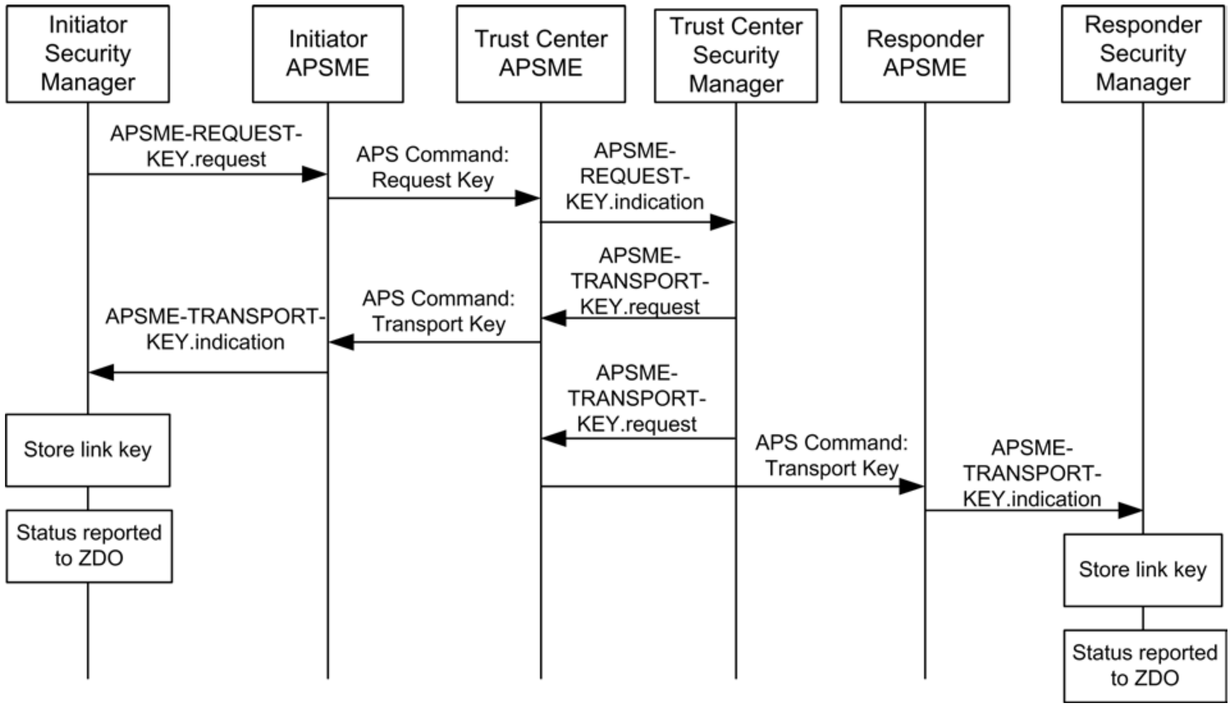
- The PartnerAddress sub-parameter shall be set to the address of the device requesting the key.
- The Initiator sub-parameter shall be set to FALSE.
- The Key sub-parameter shall be set to *K*.

4.6.3.5.3 Message Sequence Chart

An example message sequence chart of the end-to-end application key establishment procedure is shown Figure 4.26. The procedure begins with the transmission of the request-key command from the initiator to the Trust Center.

The Trust Center shall now send transport-key commands containing the application link to the initiator and responder devices. Upon completion (or time-out), the status of the protocol is reported to the ZDOs of the initiator and responder devices. If successful, the initiator and responder will now share a link key and secure communications will be possible.

Figure 4.26 Example End-to-End Application Key Establishment Procedure



4.6.3.6 Network Leave

A device, its router, and the Trust Center shall follow the procedures described in this section when the device is to leave the network.

4.6.3.6.1 Trust Center Operation

If a Trust Center wants a device to leave and if the Trust Center is not the router for that device, the Trust Center shall issue the APSME-REMOVE-DEVICE.request primitive, with the ParentAddress parameter set to the router's address and the ChildAddress parameter set to the address of the device it wishes to leave the network.

The Trust Center will also be informed of devices that leave the network. Upon receipt of an APSME-UPDATE-DEVICE.indication primitive with the Status parameter set to 0x02 (that is, device left), the DeviceAddress parameter shall indicate the address of the device that left the network and the SrcAddress parameter shall indicate the address of parent of this device.

4.6.3.6.2 Router Operation

Routers are responsible for receiving remove-device commands and for sending update-device commands.

Upon receipt of an APSME-REMOVE-DEVICE.indication primitive, if the SrcAddress parameter is equal to the *apsTrustCenterAddress* attribute of the AIB then the command shall be accepted. The router shall ignore APSME-REMOVE-DEVICE.indication primitives with the SrcAddress parameter not equal to the *apsTrustCenterAddress* attribute of the AIB.

If the DeviceAddress corresponds to the local device's address, then the device shall remove itself from the network according to section 4.6.3.6.3. If the DeviceAddress corresponds to address of a child device then a router shall issue an NLME-LEAVE.request primitive with the DeviceAddress parameter the same as the DeviceAddress parameter of the APSME-REMOVE-DEVICE.indication primitive and the rejoin parameter set to 0. Other fields are defined by the stack profile.

If the DeviceAddress does not correspond to the local device address, nor does it correspond to a child device of the router, the command shall be discarded.

Upon receipt of an NLME-LEAVE.indication primitive with the DeviceAddress parameter set to one of its children and with the Rejoin Parameter = 0, a router that is not also the Trust Center shall issue an APSME-UPDATE-DEVICE.request primitive with:

- The DstAddress parameter set to the address of the Trust Center.
- The Status parameter set to 0x02 (that is, device left).
- The DeviceAddress parameter set to the DeviceAddress parameter of the NLME-LEAVE.indication primitive.

If the router is the Trust Center, it should simply operate as the Trust Center and shall not issue the APSME-UPDATE-DEVICE.request primitive (see section 4.6.3.6.1).

4.6.3.6.3 Leaving Device Operation

Devices are responsible for receiving and sending leave messages. The following rules apply to all three types of leave messages: NWK Leave Command, ZDO Mgmt Leave, and APS Command: Remove Device.

In a secured ZigBee network, leave messages shall be secured with the active network key and sent with security enabled at the level indicated by the *nwkSecurityLevel* attribute in the NIB.

In a secured ZigBee network, leave messages shall be received and processed only if secured with the active network key and received with security enabled at the level indicated by the *nwkSecurityLevel* attribute in the NIB.

A device shall only send a NWK leave message (request or announcement) if it has the active network key. A device that wishes to leave the network and does not have the active network key shall quietly leave the network without sending a NWK leave announcement.

4.6.3.6.4 Message Sequence Charts

Figure 4.27 shows an example message sequence chart in which a Trust Center asks a router to remove one of its children from the network. If a Trust Center wants a device to leave and if the Trust Center is not the router for that device, the Trust Center shall send the router a remove-device command with the address of the device it wishes to leave the network. In a secure network, the remove-device command shall be secured with a link key if present; otherwise shall be secured with the active network key. Upon receipt of the remove-device command, a router shall send a leave command to the device to leave the network.

Figure 4.27 Example Remove-Device Procedure

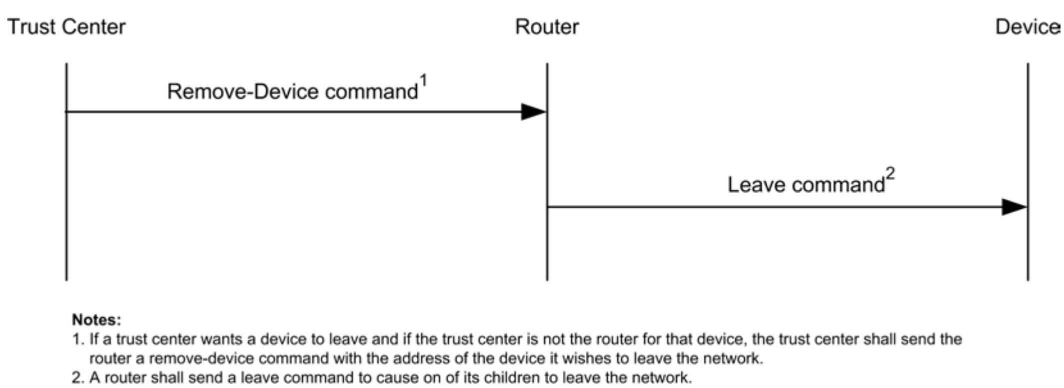
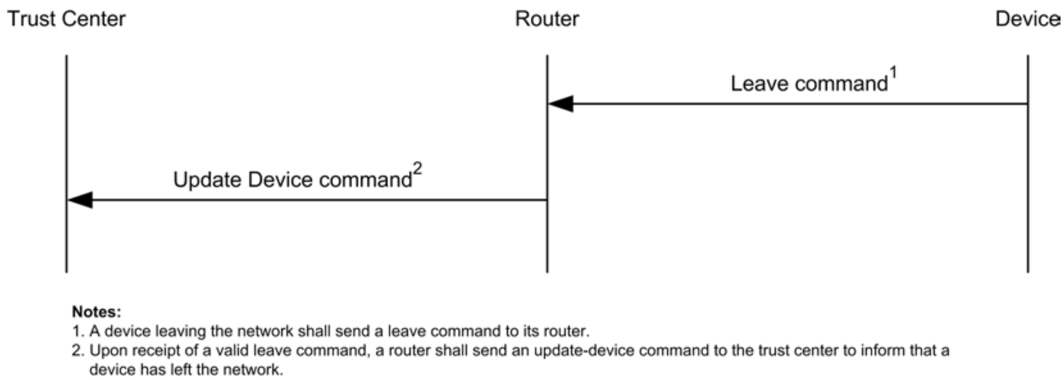


Figure 4.28 shows an example message sequence chart whereby a device notifies its router that it is leaving the network. In this example, the device sends a leave command (secured with the active network key) to its router. The router then sends an update-device command to the Trust Center. In a secured network, the update-device command must be secured with the link key, if present, or the active network key.

Figure 4.28 Example Device-Leave Procedure



4.6.3.7 Command Tunneling

Devices shall follow the procedures described in this section to allow secure communication between the Trust Center and a remote device that does not have the current network key.

4.6.3.7.1 Trust Center Operation

To embed a command in a tunnel command, the Trust Center shall first apply security protection as specified in section 4.4.1.1 and then, if security processing succeeds, the secured command frame shall be embedded in a Tunnel command frame as follows:

1. The APS header fields shall be set to the values of the APS header fields of the command to be embedded.
2. The destination address field shall be set to the 64-bit extended address of the destination device.
3. The tunneled auxiliary frame field shall be set to the auxiliary frame of the secured command, with following changes:
    - The extended nonce sub-field shall be set to 1;
    - The source address field shall be set to the 64-bit extended address of the Trust Center;
    - The tunneled command shall be set to the secured payload of the embedded command.

The tunneled command shall then be sent to the parent or other neighbor of the destination device.

4.6.3.7.2 Parent Operations

Upon receipt of an APS tunnel command, a router shall extract the embedded command as follows:

1. The APS header fields shall be set to the values of the APS header fields of the tunnel command.
2. The auxiliary frame field shall be set to the value of the tunnelled auxiliary frame field of the tunnel command.
3. The APS payload field shall be set to the tunnelled command field of the tunnel command.

The extracted command shall be sent to the destination indicated by the destination address field by issuing the NLDE-DATA.request primitive with security disabled.

4.6.3.7.3 Tunneled Data Destination Operation

The following applies to the end destination of the tunneled data payload after the parent has extracted and transmitted the payload from the APS tunnel command. Upon receipt of a message secured at the APS layer and with an extended nonce in the APS auxiliary frame, the message shall be processed as usual, except that the message shall not be looked up in, or added to, the APS duplicate rejection table.

4.7 Security Operations in Centralized Security Networks

The security services provided here offer a range of options within a ZigBee network. For interoperable and consistent field behavior, a more defined set of policies and processes is defined here. The basis for these operations is that the device forming a network can establish security policies believed appropriate for the network and that a joining device will acknowledge and use the policies in place in the network. Joining is therefore based on the forming device setting policies within the allowed settings in this section and the joining device having the appropriate flexibility to adapt to these settings.

4.7.1 Trust Center Policies

The Trust Center is a critical security component in a ZigBee network. The policies that the Trust Center puts in place control what devices get on the network and how they do so in a secure manner. Security is not an end unto itself but a means to establish a reasonable level of protection of the application and data that is being transmitted across the ZigBee network. Often an increase in security increases the overhead in management, requires additional time and functional states while security is negotiated, and can detract from a user experience by requiring them to go through additional steps that seem unnecessary. Therefore a balance must be struck between the hardening the network against attacks and the ability to use the network for the applications it was intended for.

It is important to understand the security decisions that are being made in the network and the design of the Trust Center application is at the heart of those decisions. This section presents the options and settings for the Trust Center and requires a series of choices to be set on network start up.

4.7.2 Trust Center Link Keys

Support for link keys shall be required for all devices. Link keys offer an additional level of security for devices to be able to send messages with end-to-end security instead of just with the hop-by-hop security provided by network encryption.

In addition, link keys are crucial for providing a simple authorization mechanism. The Trust Center can send devices a copy of the network key that is intended only for a specific device using that device's link key to encrypt the message.

4.7.3 Trust Center Policy Values

The following is a list of configuration values that relate to the Trust Center's policy decisions that are part of the security related AIB in Table 4.29. They will be used to describe requirements for dictating the network security policies. The trust center can use these policies to create higher or lower sets of security and controls on the network. For example:

- A system can be set up with centralized security such that any device can join the network. In such a permissive network, trust center link keys are still updated from the global value used initially for joining.

- A system can also be set up with trust center policies that only allow known devices. A user must then install the IEEE address and a link key for the new device into the trust center prior to the device joining. This could be done using install code based keys. This validates to the joining device that it is on a network that knows its identity during the joining process. The trust center in this network can also update the trust center link keys of joining devices so secure key updates and rejoining can be conducted during the lifetime of the network.

Table 4.32 describes the Trust Center policy values trustCenterPolicies of the AIB and their usage.

Table 4.33 Trust Center Policy Values

Attribute	Identifier	Type	Range	Description	Usage
allowJoins	0xad	boolean	TRUE or FALSE	This boolean indicates whether the Trust Center is currently allowing devices to join the network. A value of TRUE means that the Trust Center is allowing devices that have never been sent the network key or a trust center link key, to join the network.	This is set to FALSE in centralized security networks that do not want to allow new devices on the network.
useWhiteList	0xae	boolean	TRUE or FALSE	This boolean indicates whether the Trust Center allows any device with any IEEE to join or allows only known devices. A value of FALSE means that the Trust Center will allow any IEEE address to join the network. A value of TRUE means that the Trust Center will only allow IEEE addresses listed in *apsDeviceKeyPairSet* to join the network.	This is set to TRUE in centralized security networks that only allow devices known to them to join or rejoin. Trust centers that set this to TRUE shall provide a user interface or out of bands means to update the trust center with IEEE address of new devices to join the network.
allowInstallCodes	0xaf	enumeration	0x00 – 0x10	This enumeration indicates if the Trust Center requires install codes to be used with joining devices. 0x00 – do not support Install Codes 0x01 – Support but do not require use of Install Codes 0x02 – Require the use of Install Codes by joining devices	This is set to 0x02 if the trust center requires install codes in new devices. If this is set to 0x02 then useWhiteList would normally be set to TRUE. Trust Centers that support setting 0x01 or 0x02 shall provide a user interface or out of band means to input the Install Code.
updateTrustCenterLinkKeysRequired	0xb3	Boolean	TRUE or FALSE	This boolean indicates whether or not devices are required to attempt to update their Trust Center Link Keys after joining. If set to TRUE, the device must attempt a procedure to update its link key after joining the network.	This is set to TRUE in centralized security networks.
allowRejoins	0xb6	Boolean	TRUE or FALSE	This value indicates if the trust center allows rejoins using well known or default keys. A setting	This is set to FALSE in centralized security networks.

				of FALSE means rejoins are only allowed with trust center link keys where the KeyAttributes of the apsDeviceKeyPairSet entry indicates VERIFIED_KEY.	
allowTrustCenterLinkKeyRequests	0xb7	enumeration	0x00 – 0x02	This value controls whether devices are allowed to request a Trust Center Link Key after they have joined the network. It may have the following values: 0x00 – never 0x01 – any device may request 0x02 – Only devices in the *apsDeviceKeyPairSet* with a KeyAttribute value of PROVISIONAL_KEY may request.	This is set to 0x00 in networks with higher level protocols for establishing link keys. This is set to either 0x01 or 0x02 in centralized security networks.
networkKeyUpdatePeriod	0xb9	Integer	0x00000000 – 0xFFFFFFFF	The period, in minutes, of how often the network key is updated by the Trust Center. A period of 0 means the Trust Center will not periodically update the network key (it may still update key at other times).	This is used in the Trust Center of centralized security networks to establish the network key update period. When this time is up the Trust Center updates the network key.
networkKeyUpdateMethod	0xba	enumeration	0x00 – 0x01	This value describes the method the Trust Center uses to update the network key. 0x00 – Broadcast using only network encryption 0x01 – Unicast using network encryption and APS encryption with a device's link key.	This is used in centralized security networks to establish the policy for updating the network key.
allowApplicationKeyRequests	0xbb	enumeration	0x00 – 0x02	This value determines how the Trust Center shall handle attempts to request an application link key with a partner node. 0x00 – never 0x01 – Any device may request an application link key with any device (except the Trust Center) 0x02 – Only those devices listed in *applicationKeyRequestList* may request and receive application link keys.	This is used in centralized security networks to establish the Trust Center policy on providing Application Link keys between devices on the network. It is normally set to 0x01 allowing any device to request a link key with another device to support those applications that want to encrypt application payload.
applicationKeyRequestList	0xbc	List of address pairs	Variable	This is a list of IEEE pairs of devices, which are allowed to establish application link keys between one another. The first IEEE address is the initiator, the second is the responder. If the responder address is set to 0xFFFFFFFFFFFFFFFF, then the initiator is allowed to request an application link key with any device. If the responder's address is	This list is normally not used in centralized security networks unless the Trust Center policy restricts those devices allowed to request link keys.

				not 0xFFFFFFFFFFFFFFFF, then it may also initiate an application link key request. This list is only valid if *allowApplicationkeyRequests* is set to 0x02.	
allowRemoteTcPolicyChange	0xbd	Boolean	TRUE or FALSE	This policy indicates whether a node on the network that transmits a ZDO Mgmt_Permit_Join with a significance set to 1 is allowed to effect the local Trust Center's policies.[3]	

4.7.3.1 Allowing Devices to Join

If the Trust Center receives notification that a device is joining the network via the APSME-UPDATE-DEVICE.indication with the Status field set to Standard device unsecured join (0x01), the following procedure shall be performed:

1. If allowJoins is set to FALSE, the following shall be done.
    a. The Trust Center shall proceed to the process of rejecting the join described in section 4.7.3.4. No further processing shall be done.
2. If useWhiteList is set to TRUE, the following shall be done.
    a. Search the apsDeviceKeyPairSet for an address that matches the IEEE of the joining device. If none is found, it shall proceed to the process of rejecting the join described in section 4.7.3.4. No further processing shall be done.
3. The device has been authorized for admission to the network and the following shall be performed.
    a. Examine apsDeviceKeyPairSet for an address that matches the IEEE of the joining device, if none is found the following shall be performed.
        i. Add a new entry setting the DeviceAddress of the apsDeviceKeyPairSet to the DeviceAddress of the APSME-UPDATE-DEVICE.indication and the LinkKey of the apsDeviceKeyPairSet to the value 5A 69 67 42 65 65 41 6C 6C 69 61 6E 63 65 30 39 (ZigBeeAlliance09).
    b. Generate an APSME-TRANSPORT-KEY.request primitive with the following parameters.
        i. Set the DestAddress to the DeviceAddress of the APSME-UPDATE-DEVICE.indication.
        ii. Set the StandardKeyType to Standard Network Key (0x01).
        iii. Set the TransportKeyData to the Key field of the active network key entry in the nwkSecurityMaterialSet NIB attribute.

4.7.3.2 Remote Device Changing Trust Center Policy

In some networks it may be permissible for a joined device to request that the Trust Center allow an unjoined device to be commissioned on the network. This can be accomplished through the ZDO Mgmt_Permit_Join_Req sent to the Trust Center with the TC_Significance field set to 1. Upon receipt of this request, the following procedure shall be executed.

[3] CCB 1550

1. If allowRemoteTcPolicyChange is set to 0, then the operation shall be denied and the status of 0xa3 (ILLEGAL_REQUEST) passed back to the ZDO. No further processing shall be done.
2. If useWhiteList is set to TRUE, then the operation is invalid and the status of 0xaa NOT_SUPPORTED) shall be passed back to the ZDO. No further processing shall be done.
3. The operation is allowed by the Trust Center and a status of 0x00 (SUCCESS) shall be passed back to the ZDO.

When the new device requests to join the network the trust center will still process the joining device as described in section 4.7.3.1.[4]

4.7.3.3 Determining the Link Key for Encryption or Decryption by the Trust Center

If the Trust Center has determined that a message shall be sent with APS encryption or has been received and must be decrypted, it must determine what link key to use for the operation. The Trust Center shall examine the IEEE address of the destination (if encrypting) or source (if decrypting) and search the *apsDeviceKeyPairSet* for a matching address entry. If a match is found, it will use the associated link key to APS encrypt or decrypt the message.

If no matching entry is found then no link key is defined and processing of the message shall be stopped. The message will not be sent or received because there is no link key that can be used.

See sections 4.4.1.1 and 4.4.1.2 for incoming and outgoing frame processing.

4.7.3.4 Rejecting the Join or Rejoin

A join or rejoin is processed via an APSME-UPDATE-DEVICE.indication. Following the decision to reject a join or rejoin the following shall be done by the Trust Center.

1. If the Status of APSME-UPDATE-DEVICE.indication was Standard Device Unsecured Join (0x01) or Standard Device Trust Center Rejoin (0x03), the following shall be done.
    a. The joining or rejoining device does not have the current network key and will be left to timeout.
    b. No further processing shall be done.
2. If the Status of the APSME-UPDATE-DEVICE.indication was Standard Device Secured Rejoin (0x00), the following shall be done.
    a. Follow the procedure in section 4.7.3.5.

4.7.3.5 Removing Devices

The Trust Center has the ability to remove devices in the network via the APS Remove Device command. This message can be used to force well-behaved devices to leave the network. This is useful if the Trust Center determines after they have joined that they are not on the correct network or that the device is unable to communicate in a required application specific way.

It is important to note that this is not a secure means of removing a device. Once a malicious device has the current network key the only way to force it off the network is to distribute a new network key in a manner that prevents the malicious device from obtaining the new key. See section 4.7.3.10.

[4] CCB 1550

4.7.3.6 Processing Trust Center Link Key Requests

The Trust Center link key is a crucial element in joining the network when a preconfigured key is in place, or when a device attempts to rejoin after a missed network key update. It is also the means by which application keys are established with other devices on the network. The process in ZigBee for transporting a new link key to the device requires the previous link key as an authentication mechanism. In addition it uses APS commands which do not have support for APS retries. As a result it is possible for devices to get out of sync with regard to the Trust Center link key currently in use. To avoid this risk the Trust Center may decide to prohibit requests for new trust center link keys when one is already in place.

The following describes the process when the Trust Center is notified of an APS Request key via the APSME-REQUEST-KEY.indication with the RequestKeyType set to 0x04 (Trust Center Link Key):

1. If the APS Command Request Key message is not APS encrypted, the device shall drop the message and no further processing shall take place.
2. The device shall verify the key used to encrypt the APS command. If the SrcAddress of the APSME-REQUEST-KEY.indication primitive does not equal the value of the DeviceAddress of the corresponding apsDeviceKeyPairSet entry used to decrypt the message, the message shall be dropped and no further processing shall take place.
3. If the RequestKeyType is set to 0x04, Trust Center Link Key, the following shall be performed:
    a. If *allowTrustCenterLinkKeyRequests* is 0, then no more processing is done. The request is silently rejected.
    b. If *allowTrustCenterLinkKeyRequests* is 1, then the following is performed:
        i. Follow the procedure in section 4.7.3.6.1.
    c. If allowTrustCenterLinkKeyRequests is 2, do the following.
        i. Find an entry in the apsDeviceKeyPairSet of the AIB where the DeviceAddress of the entry matches the PartnerAddress of the APSME-REQUEST-KEY.indication primitive, and the KeyAttributes has a value of PROVISIONAL_KEY (0x00). If no entry can be found matching those criteria, then the request shall be silently dropped and no more processing shall be done.
        ii. Otherwise, follow the procedure in section 4.7.3.6.1.

4.7.3.6.1 Procedure for Generating and Sending a new Trust Center Link Key

This procedure takes an IEEE address DeviceAddress.

1. Create a new 128-bit key, KeyValue. This may be done using a random number generator, or programmatically using an algorithm.
2. Create a new entry in the apsDeviceKeyPairSet.
    a. Set the DeviceAddress of the entry to the DeviceAddress passed to this procedure.
    b. Set the LinkKey value of the entry to the KeyValue previously generated in this procedure.
    c. Set the KeyAttributes of the entry to UNVERIFIED_KEY (0x01).
    d. Set the ApsLinkKeyType of the entry to Unique Link Key (0x00).
3. If there is no space in the apsDeviceKeyPairSet attribute then processing shall fail and no further steps are executed in this procedure.

4. Issue an APSME-TRANSPORT-KEY.request primitive with the DestAddress set to the DeviceAddress, the StandardKeyType set to 0x04 (Trust Center Link Key), and the TransportKeyData set to the KeyValue.

4.7.3.7 Alternate methods of Updating the Trust Center Link Key

The process of using APS request key or unsolicited transport key messages for updating the Trust Center link key has several problems. The main problem is that of synchronization. Neither side knows whether or not the other side is now using the new key, and future attempts to update the key require knowledge of the current key that is being used.

A better mechanism is a mutual authentication protocol that has the following properties:

1. The protocol must use one or more shared secrets that are not transmitted over the air during the protocol negotiation.
2. The protocol must allow both sides to inject random data in the key generation to prevent one device from controlling the result of the key generation.
3. The protocol must not require knowledge of a previously generated Trust Center link key in order to generate a new one.

The Certificate Based Key-Exchange has all of these properties.

4.7.3.8 Processing Application Link Key Requests

Devices may use the Trust Center to establish application link keys with one another. Those devices can leverage the secure communications channel they have established with the Trust Center in order to establish secure communications with other devices. The Trust Center policy dictates whether or not it will answer application link key requests. Trust Center shall only allow application link key requests it receives that are encrypted with the device's Trust Center link key. Any application link key request that is not APS encrypted shall be dropped. In addition, if the Trust Center does not have a link key in *apsDeviceKeyPairSet* for the responder device listed in the APS Request Key Command, it shall drop the request. The purpose of the using the Trust Center to establish an application link key is leverage the trust each device has with the Trust Center (through their Trust Center Link Key).

The Trust Center shall ignore any requests made to establish application link keys with itself. ZigBee provides no protocol mechanism to differentiate whether a Trust Center link key or an application link key was used to encrypt a message. Therefore a device cannot determine what key to use when decrypting the message.

It is worth noting that devices are not required to use the Trust Center to establish application link keys, and that some application profiles allow devices to establish link keys without the trust center. The application profile in use by the device may require that the Trust Center be utilized to do this.

Application link key requests are initiated by the requesting device may occur at any time. Therefore the Trust Center shall not change its handling of those requests based on whether it is currently operating in commissioning or operational mode.

Upon receipt of an APSME-REQUEST-KEY.indication with the RequestKeyType set to 0x02 (Application Link Key) the following shall be performed:

1. If the PartnerAddress of the primitive is equal to the apsTrustCenterAddress of the AIB, the request shall be dropped and no further processing shall be done.

2. If the Trust Center policy of allowApplicationLinkKeyRequests is 0x00, then the request shall be dropped and no further processing shall be done.
3. If the Trust Center policy of allowApplicationLinkKeyRequests is 0x01, then the Trust Center shall do the following.
    a. Run the procedure in section 4.7.3.8.1 using SrcAddress from the primitive as the InitiatorAddress in the procedure, and PartnerAddress from the primitive as the ResponderAddress in the procedure.
    b. No further processing shall be done after that.
4. If the Trust Center policy of allowApplicationLinkKeyRequests is 0x02, then the following shall be performed.
    a. Find an entry in the allowApplicationKeyRequestList where the SrcAddress of the primitive matches the Initiator Address of the entry, and the PartnerAddress of the primitive matches the Responder Address of the entry.
    b. If no entry is found, then the request shall be dropped and no further processing done.
    c. If an entry is found, follow the procedure in section 4.7.3.8.1.

4.7.3.8.1 Procedure for Generating and Sending Application Link Keys

This procedure takes two IEEE addresses, InitiatorAddress and ResponderAddress.

1. The Trust Center shall generate a random 128-bit key KeyValue for the application link key.
2. It shall issue an APSME-TRANSPORT-KEY.request with the StandardKeyType set to 0x03, Application Link Key, the TransportKeyData set to KeyValue, and the DestAddress set to InitiatorAddress.
3. It shall issue a sceond APSME-TRANSPORT-KEY.request with the StandardKeyType set to 0x03, Application Link Key, the TransportKeyData set to KeyValue, and the DestAddress set to ResponderAddress.

4.7.3.9 Key Lifetime

How long a network key or trust-center link key remains in use is up to the trust center. The longer a key is in use the more chance there is of it becoming compromised. On the other hand, updating a key too often adds management overhead and increases the risk that problems during key transmission will disrupt the network. A balance must be struck between the needs of security and the temporary disruption a new key can cause.

4.7.3.9.1 Link Key Lifetime

- It is advisable that the trust center have a policy for link keys to be changed periodically. This is can be difficult for sleepy end devices, which must check with the trust center periodically to receive any newly-available key.
- It is recommended that old, unused link keys be deleted from the Trust Center to prevent them from being used. This requires that devices periodically communicate with the Trust Center using APS security to allow it to keep track of usage of the keys.
- Often a link key is used to initially join the network and thus it is uncertain how long the key may have been in use before joining the network. Preconfigured link keys may be extremely long lived and thus increases the need to update the link key as soon as the device joins the network.

- Link keys that are established using higher level protocols are not updated based on trust center policies but on the higher level application policies.

4.7.3.9.2 Network Key Lifetime

The trust center shall periodically distribute and then switch to a new network key. There are two main reasons for doing this:

1. An update and switch resets the outgoing NWK frame counter of all devices on the network. This lengthens the life of the network, since once the frame counter of a device gets to all 0xFFFFFFFF it cannot send network encrypted traffic.
2. It reduces the risk of a network key being compromised through attacks

If a trust center detects that the frame counter for any device in its neighbor table is greater than 0x80000000 it should update the network key.

Trust centers should update the network key at least once per year. It is not recommended to update the network key more than once every 30 days except when required by the application or profile.

Trust centers that do not have a real time clock or other means of tracking time are recommended to perform a network key update when their outgoing frame counter reaches 0x40000000.

4.7.3.10 Updating the Network Key

Updating the Network key is one of the core responsibilities of the Trust Center. It helps to insure that a key does not remain in use for too long and thus is not too susceptible to compromise.

4.7.3.10.1 Period of Updates

The network key shall be updated periodically. How often an update is sent out is based on the *nwkKeyTrustCenterUpdatePeriod*.

4.7.3.10.2 Sleepy Devices

Sleepy devices may miss many network events, such as a channel change, PAN ID change, or a parent that has left. Sleepy devices may not be awake at the point when the Trust Center is updating the network key, regardless of whether the key is broadcast or unicast. If the sleeping device happens to poll within nwkTransactionPersistenceTime for a unicast key update, or nwkBroadcastDeliveryTime for a broadcast key update, the update message shall be delivered. Otherwise the delivery of the key update to the sleepy device will timeout and the sleepy device will not receive the update.

The sleepy device should consider the network key update another one of those events and will need to handle that case when waking up. A child that sends a message to its parent and receives a MAC ack but no response at the application layer may have missed a key update and therefore should try to perform a rejoin. If the parent has switched to the newer key, the sleeping device must perform a trust center rejoin.

4.7.3.10.3 Broadcast Network Key Updates

Broadcast key updates are the simplest mechanism for distributing new network keys. The new network key is broadcast using the existing network key to encrypt it. There is no way to exclude a device that has the current network key from this kind of key update.

4.7.3.10.4 Unicast Network Key Updates

A more secure way of sending out network key updates is by using unicast messages encrypted with each device’s link key. This requires that all authorized devices on the network have a link key so that the Trust Center can individually update them in a secure manner. A Trust Center that wishes to securely remove a previously authorized device should use this mechanism as it can be used to exclude a device from the network.

If this unicast method is used by the trust center, it is required that the Trust Center maintain a list of all the routers on the network and send key updates to only those devices. Sleepy devices are unlikely to be awake when the Trust Center decides to change the network key. Sending to only routers also reduces the amount of network traffic that the Trust Center has to generate in order to update the network.

4.7.3.10.5 Key Switch

Regardless of the mechanism used to perform a key update (broadcast or unicast), it is required that the APS key switch command be broadcast. Devices will implicitly switch the network key when they hear another device using the newer key. This mechanism insures that even if the device did not receive the formal key switch, it will start using the new key

A device can determine if the new network key is actively being used by examining the key sequence number in the NWK auxiliary header of packets it receives. If it receives a message that passes decryption using the new key sequence number then it shall switch to using the newer network key and stop sending message encrypted using the old network key.

4.7.3.10.6 Old Network Keys

A network key update and switch does not preclude the use of the previous network key. A device is allowed to accept messages encrypted using the last network key, as this insures a smooth transition to the new key. A device shall never send messages using the old key.

To completely deprecate a key's use, the Trust Center will have to perform an update and switch twice. If using a broadcast key update, the Trust Center should make sure that both the key update and the key switch broadcasts have completely expired before sending a second set to update and switch.

4.8 Security Operations in Distributed Security Networks

In distributed security networks, there is not a single trust center in control of the network. Each router can act as a parent and transport keys to joining devices. In addition, if a device already has a network key from an out of band installation method or commissioning, the device is accepted into the network without any trust center authorization.

4.8.1 Trust Center Address

In distributed security networks the trust center address is 0xFFFFFFFFFFFFFFFF. This address is used in transport key commands as the source address to indicate the network is in a distributed security model.

4.8.2 Network Key Updates

Network key updates are not done in distributed security networks.

4.8.3 Link Keys

Link keys are only used to APS encrypt transport key commands during joining in distributed security networks. The key type stored internally shall be 0x01 (Global Link Key).

4.8.4 Application Link Keys

Devices may require use of application link keys for APS data. In a distributed security network the partner devices must use a higher level protocol to establish the application link key without the trust center involvement or permissions.

4.8.5 Requesting Keys

There is no facility to process or answer APSME-REQUEST-KEY primitives. All APS Command Request Key frames shall be dropped and no further processing shall be done.

4.9 Device Operations

Devices joining the network shall also have policies that dictate what security they expect from the network. The following are the settings that can be used to adjust their security policy.

4.9.1 Joining Device Policy Values

A joining device may have a set of policy values enumerated in Table 4.33. However, it normally sets these policy values upon joining based on if the network is a centralized or distributed security model. All devices shall support joining either network and adapting their security policies accordingly unless their application profile mandates joining only one type of network.

Table 4.34 Joining Device Policy Values

Name	Type	Range	Description	Usage
requestNewTrustCenterLinkKey	Boolean	TRUE or FALSE	This boolean indicates whether the device will request a new Trust Center Link key after joining. A value of TRUE means the device shall send an APS request key command to the Trust Center with *RequestKeyType* 0x04. If the request is not answered *requestLinkKeyTimeout* seconds then the device will leave the network. A value of FALSE means the device will not request a new link key.	This is set to TRUE in centralized security networks to ensure devices have a trust center link key for rejoining or key updates. Note this value is set to FALSE in a distributed security network.
requestLinkKeyTimeout	Integer	0 – 10	This integer indicates the maximum time in seconds that a device will wait for a response to a request for a Trust Center link key.	This is ignored in a distributed security network.
acceptNewUnsolicitedApplicationLinkKey	Boolean	TRUE or FALSE	This boolean indicates whether the device will accept a new unsolicited application link key sent to it by the Trust Center.	

4.9.2 Trust Center Address

A device will not know the address of the Trust Center prior to joining. The *apsTrustCenterAddress* in the AIB shall be initially set to 0x0000000000000000. Upon joining a device shall receive an APS Transport key and the source address shall indicate the address of the trust center. The *apsTrustCenterAddress* in the AIB will be set to the address in the received packet.

A value of 0xFFFFFFFFFFFFFFFF for the *apsTrustCenterAddress* in the AIB indicates a distributed security network and the device settings should be adjusted accordingly.

See section 4.4.1.5 for a description of when and how the trust center address of APS commands are validated.

4.9.3 Trust Center Link Keys

All devices in a centralized security network shall obtain an updated Trust Center link key when they first join the network and the Trust Center supports this behavior. An updated trust center link key protects the device from compromise if the original joining key is discovered. The application may utilize a key establishment algorithm if one is available. If such an algorithm is not available, the Request Key services of the APSME must be used.

Prior to revision 21 of this specification, there was not an interoperable mechanism to update the link key so. Therefore a Trust Center operating on a prior revision is not assumed to have support for this behavior. Determining the Trust Center revision can be done using the Server Mask and the ZDO Node Descriptor Request. Initiation of this process is done by the higher application.

Once the device has obtained an updated Trust Center link key it shall ignore any APS commands from the Trust Center that are not encrypted with that key.

4.9.4 Receiving new Link Keys

It is possible a device's security policy may restrict application link keys sent to it by the trust center for use with another partner device. This could be because the device wishes to control which other devices it shares link keys with, or because it uses some other mechanism to establish application link keys with devices besides the trust center.

There are instances where higher level application policies determine what data is shared with application link keys. For example, networks where updated Trust Center link keys must be established through the Certificate Based Key Exchange protocol.

If the devices receives a transport key command containing an application link key, but it has not sent a request for one, and acceptNewUnsolicitedApplicationLinkKey is set to FALSE, it shall ignore the message.

4.9.5 Requesting a Link Key

A device shall attempt to update its trust center link key as part of its initial joining operations in a centralized security network. Trust Centers prior to the revision 21 version of this specification did not support updating trust center link keys via the APSME request key method. Determination of whether the trust center supports this behavior is left up to the higher level application. The application may use either the APSME Request Key facilities or an alternative key establishment protocol.

If the device is requesting a trust center link key using the APSME, it shall start a timer after sending the initial request. Once the timer has reached *requestLinkKeyTimeout*, the device shall no longer accept a transport key message containing a new Trust Center link key unless the device initiates a new request.

If the device is requesting an application link key and acceptNewUnsolicitedApplicationLinkKey is set to FALSE, it shall start a timer after sending the initial request. Once the timer has reached requestLinkKeyTimeout the device shall no longer accept a transport key message containing a new application link key unless it initiates a new request.

A device that did not request a new application link key and has acceptNewUnsolicitedApplicationLinkKey set to FALSE shall silently drop the APS Transport Key Command for an application link key. It shall not process the command.

This page intentionally left blank.

ANNEX A CCM* MODE OF OPERATION

CCM* is a generic combined encryption and authentication block cipher mode. CCM* is only defined for use with block ciphers having a 128-bit block size, such as AES-128 [B8]. The CCM* principles can easily be extended to other block sizes, but doing so will require further definitions.

The CCM* mode coincides with the original CCM mode specification [B21] for messages that require authentication and, possibly, encryption, but does also offer support for messages that require only encryption. As with the CCM mode, the CCM* mode requires only one key. The security proof for the CCM mode([B22] and [B23]) carries over to the CCM* mode described here. The design of the CCM* mode takes into account the results of [B24], thus allowing it to be securely used in implementation environments in which the use of variable-length authentication tags, rather than fixed-length authentication tags only, is beneficial.

Prerequisites: The following are the prerequisites for the operation of the generic CCM* mode:

1. A block-cipher encryption function E shall have been chosen, with a 128-bit block size. The length in bits of the keys used by the chosen encryption function is denoted by *keylen*.
2. A fixed representation of octets as binary strings shall have been chosen (for example, most-significant-bit first order or least-significant-bit-first order).
3. The length L of the message length field, in octets, shall have been chosen. Valid values for L are the integers 2, 3,..., 8 (the value L=1 is reserved).
4. The length M of the authentication field, in octets, shall have been chosen. Valid values for M are the integers 0, 4, 6, 8, 10, 12, 14, and 16. (The value M=0 corresponds to disabling authenticity, since then the authentication field contains an empty string.)

A.1 Notation and Representation

Throughout this specification, the representation of integers as octet strings shall be fixed. All integers shall be represented as octet strings in most-significant-octet first order. This representation conforms to the conventions in Section 4.3 of ANSI X9.63-2001 [B7].

A.2 CCM* Mode Encryption and Authentication Transformation

The CCM* mode forward transformation involves the execution, in order, of an input transformation (A.2.1), an authentication transformation (A.2.2), and encryption transformation (A.2.3).

Input: The CCM* mode forward transformation takes as inputs:

1. A bit string *Key* of length *keylen* bits to be used as the key. Each entity shall have evidence that access to this key is restricted to the entity itself and its intended key-sharing group member(s).
2. A nonce N of 15-L octets. Within the scope of any encryption key *Key*, the nonce value shall be unique.
3. An octet string m of length $l(m)$ octets, where $0 \le l(m) \le 28L$.
4. An octet string a of length $l(a)$ octets, where $0 \le l(a) < 2^{64}$.

The nonce N shall encode the potential values for M such that one can uniquely determine from N the value of M actually used. The exact format of the nonce N is outside the scope of this specification and shall be determined and fixed by the actual implementation environment of the CCM* mode.

Note: The exact format of the nonce *N* is left to the application, to allow simplified hardware and software implementations in particular settings. Actual implementations of the CCM* mode may restrict the values of *M* that are allowed throughout the life-cycle of the encryption key *Key* to a strict subset of those allowed in the generic CCM* mode. If so, the format of the nonce *N* shall be such that one can uniquely determine from *N* the actually used value of *M* in that particular subset. In particular, if *M* is fixed and the value $M=0$ is not allowed, then there are no restrictions on *N*, in which case the CCM* mode reduces to the CCM mode.

A.2.1 Input Transformation

This step involves the transformation of the input strings *a* and *m* to the strings *AuthData* and *PlainTextData*, to be used by the authentication transformation and the encryption transformation, respectively.

This step involves the following steps, in order:

1. Form the octet string representation $L(a)$ of the length $l(a)$ of the octet string *a*, as follows:
    a. If $l(a)=0$, then $L(a)$ is the empty string.
    b. If $0 < l(a) < 2^{16}-2^{8}$, then $L(a)$ is the 2-octets encoding of $l(a)$.
    c. If $2^{16}-2^{8} \leq l(a) < 2^{32}$, then $L(a)$ is the right-concatenation of the octet 0xff, the octet 0xfe, and the 4-octets encoding of $l(a)$.
    d. If $2^{32} \leq l(a) < 2^{64}$, then $L(a)$ is the right-concatenation of the octet 0xff, the octet 0xff, and the 8-octets encoding of $l(a)$.
2. Right-concatenate the octet string $L(a)$ with the octet string *a* itself. Note that the resulting string contains and *a* encoded in a reversible manner.
3. Form the padded message *AddAuthData* by right-concatenating the resulting string with the smallest non-negative number of all-zero octets such that the octet string *AddAuthData* has length divisible by 16.
4. Form the padded message *PlaintextData* by right-concatenating the octet string *m* with the smallest non-negative number of all-zero octets such that the octet string *PlaintextData* has length divisible by 16.
5. Form the message *AuthData* consisting of the octet strings *AddAuthData* and *PlaintextData*:

    $AuthData = AddAuthData \mid\mid PlaintextData$

A.2.2 Authentication Transformation

The data *AuthData* that was established above shall be tagged using the tagging transformation as follows:

1. Form the 1-octet *Flags* field consisting of the 1-bit *Reserved* field, the 1-bit *Adata* field, and the 3-bit representations of the integers *M* and *L*, as follows:

    $Flags = Reserved \mid\mid Adata \mid\mid M \mid\mid L$

    Here, the 1-bit *Reserved* field is reserved for future expansions and shall be set to ‘0’. The 1-bit *Adata* field is set to ‘0’ if $l(a)=0$, and set to ‘1’ if $l(a)>0$. The *L* field is the 3-bit representation of the integer L-1, in most-significant-bit-first order. The *M* field is the 3-bit representation of the integer $(M-2)/2$ if $M>0$ and of the integer 0 if $M=0$, in most-significant-bit-first order.
2. Form the 16-octet B_0 field consisting of the 1-octet *Flags* field defined above, the 15-*L* octet nonce field *N*, and the *L*-octet representation of the length field $l(m)$, as follows:

    $B_0 = Flags \mid\mid Nonce\ N \mid\mid l(m)$
3. Parse the message *AuthData as* $B_1 \mid\mid B_2 \mid\mid \ldots \mid\mid B_t$, where each message block B_i is a 16-octet string.

    The CBC-MAC value X_{t+1} is defined by:

    $X_0 := 0_{128};\ X_{i+1} := E(Key, X_i \oplus B_i)$ *for* $i=0, \ldots, t$.

    Here, $E(K, x)$ is the cipher-text that results from encryption of the plaintext *x* using the established block-cipher encryption function *E* with key *Key*; the string 0^{128} is the 16-octet all-zero bit string.

The authentication tag T is the result of omitting all but the leftmost M octets of the CBC-MAC value X_{n+1} thus computed.

A.2.3 Encryption Transformation

The data *PlaintextData* that was established in section A.2.1 (step 4) and the authentication tag T that was established in section A.2.2 (step 3) shall be encrypted using the encryption transformation as follows:

1. Form the 1-octet *Flags* field consisting of two 1-bit *Reserved* fields, and the 3- bit representations of the integers *0* and L, as follows:

   Flags = Reserved || Reserved || 0 || L

   Here, the two 1-bit *Reserved* fields are reserved for future expansions and shall be set to ‘0’. The L field is the 3-bit representation of the integer L-1, in most-significant- bit-first order. The *‘0’* field is the 3-bit representation of the integer 0, in most-significant-bit-first order.

   Define the 16-octet A_i field consisting of the 1-octet *Flags* field defined above, the 15-L octet nonce field N, and the L-octet representation of the integer i, as follows:

   A_i = *Flags || Nonce N || Counter i, for i=0, 1, 2, …*

   Note that this definition ensures that all the A_i fields are distinct from the B_0 fields that are actually used, as those have a *Flags* field with a non-zero encoding of M in the positions where all A_i fields have an all-zero encoding of the integer 0 (see section A.2.2, step 1).

   Parse the message *PlaintextData* as $M_1 || ... ||M_t$, where each message block M_i is a 16-octet string.

   The ciphertext blocks $C_1, ... , C_t$ are defined by:

   $C_i := E(Key, A_i) \oplus M_i$ *for* $i=1, 2, ... , t$

   The string *Ciphertext* is the result of omitting all but the leftmost $l(m)$ octets of the string $C_1 || ... || C_t$

   Define the 16-octet encryption block S_0 by:

   $S_0 := E(Key, A_0)$

2. The encrypted authentication tag U is the result of XOR-ing the string consisting of the leftmost M octets of S_0 and the authentication tag T.

Output: If any of the above operations has failed, then output ‘invalid’. Otherwise, output the right-concatenation of the encrypted message *Ciphertext* and the encrypted authentication tag U.

A.3 CCM* Mode Decryption and Authentication Checking Transformation

Input: The CCM* inverse transformation takes as inputs:

1. A bit string *Key* of length *keylen* bits to be used as the key. Each entity shall have evidence that access to this key is restricted to the entity itself and its intended key-sharing group member(s).
2. A nonce N of 15-L octets. Within the scope of any encryption key *Key*, the nonce value shall be unique.
3. An octet string c of length $l(c)$ octets, where $0 \leq l(c)-M < 2^{8L}$.
4. An octet string a of length $l(a)$ octets, where $0 \leq l(a) < 2^{64}$.

A.3.1 Decryption Transformation

The decryption transformation involves the following steps, in order:

1. Parse the message c as $C ||U$, where the rightmost string U is an M-octet string. If this operation fails, output ‘invalid’ and stop. U is the purported encrypted authentication tag. Note that the leftmost string C has length $l(c)-M$ octets.

2. Form the padded message *CiphertextData* by right-concatenating the string *C* with the smallest non-negative number of all-zero octets such that the octet string *CiphertextData* has length divisible by 16.
3. Use the encryption transformation in section A.2.3, with the data *CipherTextData* and the tag *U* as inputs.
4. Parse the output string resulting from applying this transformation as $m \| T$, where the rightmost string *T* is an *M*-octet string. *T* is the purported authentication tag. Note that the leftmost string *m* has length *l(c)-M* octets.

A.3.2 Authentication Checking Transformation

The authentication checking transformation involves the following steps:

1. Form the message *AuthData* using the input transformation in section A.2.1, with the string *a* and the octet string *m* that was established in section A.3.1 (step 4) as inputs.
2. Use the authentication transformation in section A.2.2, with the message *AuthData* as input.
3. Compare the output tag *MACTag* resulting from this transformation with the tag *T* that was established in section A.3.1 (step 4). If *MACTag=T*, output 'valid'; otherwise, output 'invalid' and stop.

Output: If any of the above verifications has failed, then output 'invalid' and reject the octet string *m*. Otherwise, accept the octet string *m* and accept one of the key sharing group member(s) as the source of *m*.

A.4 Restrictions

All implementations shall limit the total amount of data that is encrypted with a single key. The CCM* encryption transformation shall invoke not more than 2^{61} block-cipher encryption function operations in total, both for the CBC-MAC and for the CTR encryption operations.

At CCM* decryption, one shall verify the (truncated) CBC-MAC before releasing any information, such as, *Plaintext*. If the CBC-MAC verification fails, only the fact that the CBC-MAC verification failed shall be exposed; all other information shall be destroyed.

ANNEX B SECURITY BUILDING BLOCKS

This annex specifies the cryptographic primitives and mechanisms that are used to implement the security protocols in this standard.

B.1 Symmetric-Key Cryptographic Building Blocks

The following symmetric-key cryptographic primitives and data elements are defined for use with all security-processing operations specified in this standard.

B.1.1 Block-Cipher

The block-cipher used in this specification shall be the Advanced Encryption Standard AES-128, as specified in FIPS Pub 197. This block-cipher has a key size *keylen* that is equal to the block size, in bits, *i.e.*, *keylen*=128.

B.1.2 Mode of Operation

The block-cipher mode of operation used in this specification shall be the CCM* mode of operation, as specified in section A.2.3, with the following instantiations:

1. Each entity shall use the block-cipher E as specified in section B.1.1.
2. All octets shall be represented as specified in the "Conventions and Abbreviations."
3. The parameter L shall have the integer value 2.
4. The parameter M shall have one of the following integer values: 0, 4, 8, or 16.

B.1.3 Cryptographic Hash Function

The cryptographic hash function used in this specification shall be the blockcipher based cryptographic hash function specified in section B.6, with the following instantiations:

1. Each entity shall use the block-cipher E as specified in section B.1.1.
2. All integers and octets shall be represented as specified in section 1.2.1.

The Matyas-Meyer-Oseas hash function (specified in section B.6) has a message digest size *hashlen* that is equal to the block size, in bits, of the established block-cipher.

B.1.4 Keyed Hash Function for Message Authentication

The keyed hash message authentication code (HMAC) used in this specification shall be HMAC, as specified in the FIPS Pub 198 [B9], with the following instantiations:

1. Each entity shall use the cryptographic hash *H* function as specified in section B.1.3**.**.
2. The block size *B* shall have the integer value 16 (this block size specifies the length of the data integrity key, in bytes, that is used by the keyed hash function, *i.e.*, it uses a 128-bit data integrity key).
3. The output size *HMAClen* of the HMAC function shall have the same integer value as the message digest parameter *hashlen* as specified in section B.1.3.

B.1.5 Specialized Keyed Hash Function for Message Authentication

The specialized keyed hash message authentication code used in this specification shall be as specified in section B.1.4.

B.1.6 Challenge Domain Parameters

The challenge domain parameters used in the specification shall be as specified in section B.3.1, with the following instantiation: (*minchallengelen*, *maxchallengelen*)=(128,128).

All challenges shall be validated using the challenge validation primitive as specified in section B.4.

B.2 Key Agreement Schemes

B.2.1 Symmetric-Key Key Agreement Scheme

The symmetric-key key agreement protocols in this standard shall use the full symmetric-key with key confirmation scheme, with the following instantiations:

1. Each entity shall use the HMAC-scheme as specified in section B.1.4.
2. Each entity shall use the specialized HMAC-scheme as specified in section B.1.5.
3. Each entity shall use the cryptographic hash function as specified in section B.1.3.
4. The parameter *keydatalen* shall have the same integer value as the key size parameter *keylen* as specified in section B.1.1.
5. The parameter *SharedData* shall be the empty string; parameter *shareddatalen* shall have the integer value 0.
6. The optional parameters $Text_1$ and $Text_2$ as specified in section B.7.1 and section B.7.2 shall both be the empty string.
7. Each entity shall use the challenge domain parameters as specified in section B.1.6.
8. All octets shall be represented as specified in section 1.2.1.

B.3 Challenge Domain Parameter Generation and Validation

This section specifies the primitives that shall be used to generate and validate challenge domain parameters.

Challenge domain parameters impose constraints on the length(s) of bit challenges a scheme expects. As such, this establishes a bound on the entropy of challenges and, thereby, on the security of the cryptographic schemes in which these challenges are used. In most schemes, the challenge domain parameters will be such that only challenges of a fixed length will be accepted (for example, 128-bit challenges). However, one may define the challenge domain parameters such that challenges of varying length might be accepted. Doing so is useful in contexts in which entities that wish to engage in cryptographic schemes might have a bad random number generator onboard. Allowing both entities that engage in a scheme to contribute sufficiently long inputs enables each of them to contribute sufficient entropy to the scheme.

In this standard, challenge domain parameters will be shared by a number of entities using a scheme determined by the standard. The challenge domain parameters may be public; the security of the system does not rely on these parameters being secret.

B.3.1 Challenge Domain Parameter Generation

Challenge domain parameters shall be generated using the following routine.

Input: This routine does not take any input.

Actions: The following actions are taken:

1. Choose two nonnegative integers *minchallengelen* and *maxchallengelen*, such that *minchallengelen* ≤ *maxchallengelen.*

Output: Challenge domain parameters *D*=(*minchallengelen*, *maxchallengelen*).

B.3.2 Challenge Domain Parameter Verification

Challenge domain parameters shall be verified using the following routine.

Input: Purported set of challenge domain parameters D=(minchallengelen, maxchallengelen).

Actions: The following checks are made:

1. Check that *minchallengelen* and *maxchallengelen* are non-negative integers.
2. Check that *minchallengelen* ≤ *maxchallengelen*.

Output: If any of the above verifications has failed, then output 'invalid' and reject the challenge domain parameters. Otherwise, output 'valid' and accept the challenge domain parameters.

B.4 Challenge Validation Primitive

It is used to check whether a challenge to be used by a scheme in the standard has sufficient length (for example, messages that are too short are discarded, due to insufficient entropy).

Input: The input of the validation transformation is a valid set of challenge domain parameters *D*=(*minchallengelen*, *maxchallengelen*), together with the bit string *Challenge*.

Actions: The following actions are taken:

1. Compute the bit-length *challengelen* of the bit string *Challenge*.

2. Verify that *challengelen* ∈ [*minchallengelen*, *maxchallengelen*]. (That is, verify that the challenge has an appropriate length.)

Output: If the above verification fails, then output 'invalid' and reject the challenge. Otherwise, output 'valid' and accept the challenge.

B.5 Secret Key Generation (SKG) Primitive

This section specifies the SKG primitive that shall be used by the symmetric-key key agreement schemes specified in this standard.

This primitive derives a shared secret value from a challenge owned by an entity U_1 and a challenge owned by an entity U_2 when all the challenges share the same challenge domain parameters. If the two entities both correctly execute this primitive with corresponding challenges as inputs, the same shared secret value will be produced.

The shared secret value shall be calculated as follows:

Prerequisites: The following are the prerequisites for the use of the SKG primitive:

1. Each entity shall be bound to a unique identifier (e.g., distinguished names).

   All identifiers shall be bit strings of the same length *entlen* bits. Entity U_1's identifier will be denoted by the bit string U_1. Entity U_2's identifier will be denoted by the bit string U_2.
2. A specialized MAC scheme shall be chosen, with tagging transformation as specified in Section 5.7.1 of ANSI X9.63-2001 [B7]. The length in bits of the keys used by the specialized MAC scheme is denoted by *mackeylen*.

Input: The SKG primitive takes as input:

- A bit string *MACKey* of length *mackeylen* bits to be used as the key of the established specialized MAC scheme.
- A bit string QEU_1 owned by U_1.
- A bit string QEU_2 owned by U_2.

Actions: The following actions are taken:

1. Form the bit string consisting of U_1's identifier, U_2's identifier, the bit string QEU_1 corresponding to U_1's challenge, and the bit string QEU_2 corresponding to QEU_2's challenge:

   $MacData = U_1 \| U_2 \| QEU_1 \| QEU_2$
2. Calculate the tag *MacTag* for *MacData* under the key *MacKey* using the tagging transformation of the established specialized MAC scheme:

   $MacTag = MACMacKey(MacData)$
3. If the tagging transformation outputs 'invalid', output 'invalid' and stop.
4. Set Z=*MacTag*.

Output: The bit string Z as the shared secret value.

B.6 Block-Cipher-Based Cryptographic Hash Function

This section specifies the Matyas-Meyer-Oseas hash function, a cryptographic hash function based on block-ciphers. We define this hash function for blockciphers with a key size equal to the block size, such as AES-128, and with a particular choice for the fixed initialization vector *IV* (we take *IV*=0). For a more general definition of the Matyas-Meyer-Oseas hash function, refer to Section 9.4.1 of [B19].

Prerequisites: The following are the prerequisites for the operation of Matyas- Meyer-Oseas hash function:

1. A block-cipher encryption function *E* shall have been chosen, with a key size that is equal to the block size. The Matyas-Meyer-Oseas hash function has a message digest size that is equal to the block size of the established encryption function. It operates on bit strings of length less than 22^n, where n is the block size, in octets, of the established block-cipher.
2. A fixed representation of integers as binary strings or octet strings shall have been chosen.

Input: The input to the Matyas-Meyer-Oseas hash function is as follows:

1. A bit string M of length l bits, where $0 \leq l < 22^n$

Actions: The hash value shall be derived as follows:

1. If the message M has length less than 2^n bits, pad this message according to the following procedure:
   a. Right-concatenate to the message M the binary consisting of the bit ‘1’ followed by k ‘0’ bits, where k is the smallest non-negative solution to the equation:
   $$l+1+k \equiv 7n \pmod{8n} \qquad (1)$$
   b. Form the padded message M' by right-concatenating to the resulting string the n-bit string that is equal to the binary representation of the integer l.
2. Otherwise pad this message according to the following method:
   a. Right concatenate to the message M the binary consisting of the bit '1' followed by k '0' bits, where k is the smallest non-negative solution to the equation:
   $$l + 1 + k \equiv 5n \pmod{8n} \qquad (2)$$[1]
   b. Form the padded message M' by right-concatenating to the resulting string the 2n-bit string that is equal to the binary representation of the integer l and right-concatenating to the resulting string the n-bit all-zero bit string.
3. Parse the padded message M' as $M_1 \,||\, M_2 \,||\, \ldots \,||\, M_t$ where each message block M_i is an n-octet string.
4. The output $Hash_t$ is defined by
   $$Hash_0 = 0^{8n};\ Hash_j = E(Hash_{j-1}, M_j) \oplus M_j \text{ for } j=1,\ldots,t \qquad (3)$$
   Here, $E(K, x)$ is the ciphertext that results from encryption of the plaintext x, using the established block-cipher encryption function E with key K; the string 0^{8n} is the n-octet all-zero bit string.

Output: The bit string $Hash_t$ as the hash value.

Note that the cryptographic hash function operates on bit strength of length less than 22^n bits, where n is the block size (or key size) of the established block cipher, in bytes. For example, the Matyas-Meyer-Oseas hash function with AES- 128 operates on bit strings of length less than 232 bits. It is assumed that all hash function calls are on bit strings of length less than 22^n bits. Any scheme attempting to call the hash function on a bit string exceeding 22^n bits shall output 'invalid' and stop.

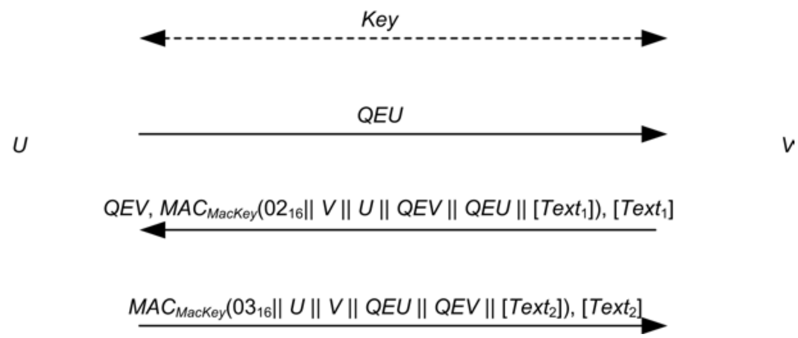
[1] CCB 1434

B.7 Symmetric-Key Authenticated Key Agreement Scheme

This section specifies the full symmetric-key key agreement with key confirmation scheme. A MAC scheme is used to provide key confirmation. Note that all key exchanges and random challenges shall be assumed within data strings in network transmission order.

Figure B.1 illustrates the messaging involved in the use of the full symmetric-key key agreement with key confirmation scheme.

Figure B.1 Symmetric-Key Authenticated Key Agreement Scheme



The scheme is ‘asymmetric’, so two transformations are specified. *U* uses the transformation specified in section B.7.1 to agree on keying data with *V* if *U* is the protocol’s initiator, and *V* uses the transformation specified in section B.7.2 to agree on keying data with *U* if *V* is the protocol’s responder. The essential difference between the role of the initiator and the role of the responder is that the initiator sends the first pass of the exchange.

If *U* executes the initiator transformation, and *V* executes the responder transformation with the shared secret keying material as input, then *U* and *V* will compute the same keying data.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system’s challenge domain parameters *D*=(*minchallengelen*, *maxchallengelen*).
2. Each entity shall have access to a bit string *Key* of length *keylen* bits to be used as the key. Each party shall have evidence that access to this key is restricted to the entity itself and the other entity involved in the symmetric-key authenticated key agreement scheme.
3. Each entity shall be bound to a unique identifier (for example, distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*’s identifier will be denoted by the bit string *U*. Entity *V*’s identifier will be denoted by the bit string *V*.
4. Each entity shall have decided which MAC scheme to use as specified in Section 5.7 of ANSI X9.63-2001 [B7]. The length in bits of the keys used by the chosen MAC scheme is denoted by *mackeylen*.
5. A cryptographic hash function shall have been chosen for use with the key derivation function.
6. A specialized MAC scheme shall have been chosen for use with the secret key generation primitive with tagging transformation as specified in Section 5.7.1 of ANSI X9.63-2001 [B7]. The length in bits of the keys used by the specialized MAC scheme is denoted by *keylen*.
7. A fixed representation of octets as binary strings shall have been chosen. (for example, most-significant-bit-first order or least-significant-bit-first order).

B.7.1 Initiator Transformation

U shall execute the following transformation to agree on keying data with *V* if *U* is the protocol's initiator. *U* shall obtain an authentic copy of *V*'s identifier and an authentic copy of the static secret key *Key* shared with *V*.

Input: The input to the initiator transformation is:

1. An integer *keydatalen* that is the length in bits of the keying data to be generated.
2. (Optional) A bit string *SharedData* of length *shareddatalen* bits that consists of some data shared by *U* and *V*.
3. (Optional) A bit string $Text^2$ that consists of some additional data to be provided from *U* to *V*.

Ingredients: The initiator transformation employs the challenge generation primitive specified in Section 5.3 of ANSI X9.63-2001 [B7], the challenge validation primitive in section B.3.2, the SKG primitive in section B.5, the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7], and one of the MAC schemes in Section 5.7 of ANSI X9.63-2001 [B7].

Actions: Keying data shall be derived as follows:

1. Use the challenge generation primitive in Section 5.3 of ANSI X9.63-2001 [B7] to generate a challenge *QEU* for the challenge domain parameters *D*. Send *QEU* to *V*.
2. Then receive from *V* a challenge *QEV'*, purportedly owned by *V*. If this value is not received, output 'invalid' and stop.
3. Verify that *QEV'* is a valid challenge for the challenge domain parameters *D* as specified in section B.3.2. If the validation primitive rejects the challenge, output 'invalid' and stop.
4. Use the SKG primitive given in section B.5 to derive a shared secret bit string *Z* from the challenges $Q1=QEU$ owned by *U* and $Q_2=QEV'$ owned by *V*, using as key the shared key *Key*. If the SKG primitive outputs 'invalid', output 'invalid' and stop.
5. Use the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7] with the established hash function to derive keying data *KKeyData* of length *mackeylen*+*keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].
6. Parse the leftmost *mackeylen* bits of *KKeyData* as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
7. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV'*, the bit string *QEU*, and if present $Text_1$:

   $MacData_1 = 02_{16} \| V \| U \| QEV' \| QEU \| [Text_1]$
8. Verify that $MacTag_1'$ is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7.2 of ANSI X9.63-2001 [B7]. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.
9. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU* corresponding to *U*'s challenge, the bit string *QEV'* corresponding to *V*'s challenge, and optionally a bit string $Text_2$:

   $MacData_2 = 03_{16} \| U \| V \| QEU \| QEV' \| [Text_2]$
10. Calculate the tag $MacTag_2$ on $MacData_2$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7.1 of ANSI X9.63-2001 [B7]:

    $$MacTag_2 = MAC_{MacKey}(MacData_2) \quad (4)$$
11. If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $MacTag_2$ and, if present, $Text_2$ to *V*.
12. Receive from *V* an optional bit string $Text_1$, and a purported tag $MacTag_1'$. If these values are not received, output 'invalid' and stop.

Output: If any of the above verifications has failed, then output ‘invalid’ and reject the bit strings *KeyData* and $Text_1$. Otherwise, output ‘valid’, accept the bit string *KeyData* as the keying data of length *keydatalen* bits shared with *V* and accept *V* as the source of the bit string $Text_1$ (if present).

B.7.2 Responder Transformation

V shall execute the following transformation to agree on keying data with *U* if *V* is the protocol’s responder. *V* shall obtain an authentic copy of *U*’s identifier and an authentic copy of the static secret key *Key* shared with *U*.

Input: The input to the responder transformation is:

1. A challenge *QEU*' purportedly owned by *U*.
2. An integer *keydatalen* that is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits that consists of some data shared by *U* and *V*.
4. (Optional) A bit string $Text_1$ that consists of some additional data to be provided from *V* to *U*.

Ingredients: The responder transformation employs the challenge generation primitive specified in Section 5.3 of ANSI X9.63-2001 [B7], the challenge validation primitive specified in section B.3.2, the SKG primitive given in section B.5, the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7], and one of the MAC schemes in Section 5.7 of ANSI X9.63-2001 [B7].

Actions: Keying data shall be derived as follows:

1. Verify that *QEU*' is a valid challenge for the challenge domain parameters *D* as specified in section B.3.2. If the validation primitive rejects the challenge, output ‘invalid’ and stop.
2. Use the challenge generation primitive in Section 5.3 of ANSI X9.63-2001 [B7] to generate a challenge *QEV* for the challenge domain parameters *D*. Send to *U* the challenge *QEV*.
3. Then receive from *U* an optional bit string $Text_2$ and a purported tag $MacTag_2'$. If this data is not received, output ‘invalid’ and stop.
4. Form the bit string consisting of the octet 03_{16}, *U*’s identifier, *V*’s identifier, the bit string *QEU*' corresponding to *U*’s purported challenge, the bit string *QEV* corresponding to *V*’s challenge, and the bit string $Text_2$ (if present):

   $MacData_2 = 03_{16} \| \mathrm{U} \| \mathrm{V} \| \mathrm{QEU'} \| \mathrm{QEV} \| [\mathrm{Text}_2]$
5. Verify that $MacTag_2'$ is the valid tag on $MacData_2$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7 ANSI X9.63-2001 [B7]. If the tag checking transformation outputs ‘invalid’, output ‘invalid’ and stop.
6. Use the SKG primitive in section B.5 to derive a shared secret bit string *Z* from the challenges $Q_1=QEU'$ owned by *U* and $Q_2=QEV$ owned by *V*, using as key the shared key *Key*. If the SKG primitive outputs ‘invalid’, output ‘invalid’ and stop.
7. Use the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7] with the established hash function to derive keying data *KKeyData* of length *mackeylen*+*keydatalen* bits from the shared secret value Z and the shared data [*SharedData*].
8. Parse the leftmost *mackeylen* bits of *KKeyData* as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
9. Form the bit string consisting of the octet 02_{16}, *V*’s identifier, *U*’s identifier, the bit string *QEV*, the bit string *QEU*', and, optionally, a bit string $Text_1$:

   $MacData_1 = 02_{16} \| \mathrm{V} \| \mathrm{U} \| \mathrm{QEV} \| \mathrm{QEU'} \| [\mathrm{Text}_1]$
10. Calculate the tag $MacTag_1$ for $MacData_1$ under the key MacKey using the tagging transformation of the appropriate MAC scheme specified in Section 5.7 of ANSI X9.63-2001 [B7]:

    $\mathrm{MacTag}_1 = \mathrm{MAC}_{MacKey}(\mathrm{MacData}_1)$

If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send to *U*, if present the bit string $Text_1$, and $MacTag_1$.

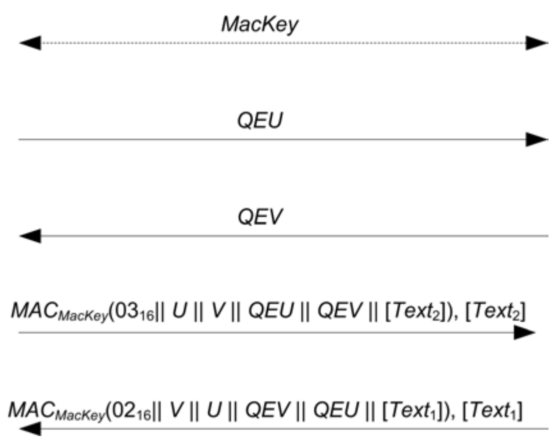
Output: If any of the above verifications has failed, then output 'invalid' and reject the bit strings *KeyData* and $Text_2$. Otherwise, output 'valid', accept the bit string *KeyData* as the keying data of length *keydatalen* bits shared with *U* and accept *U* as the source of the bit string $Text_2$ (if present).

B.8 Mutual Symmetric-Key Entity Authentication

This section specifies the mutual symmetric-key entity authentication scheme. A MAC scheme is used to provide key confirmation.

Figure B.2 illustrates the messaging involved in the use of mutual symmetric-key entity authentication scheme.

Figure B.2 Mutual Symmetric-Key Entity Authentication Scheme



The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in section B.8.1 to establish authenticity of, and optionally obtain authenticated data from, *V* by means of sharing a key and communicating cooperatively with *V*. *V* uses the transformation specified in section B.8.2 to establish authenticity of, and optionally obtain authenticated data from, *U* by means of sharing a key and communicating cooperatively with *U*.

The essential difference between the role of the initiator and the role of the responder is that the initiator sends the first pass of the exchange.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's challenge domain parameters *D=(minchallengelen, maxchallengelen)*. These parameters shall have been generated using the parameter generation primitive in section B.3.1. Furthermore, the parameters shall have been validated using the parameter validation primitive in section B.3.2.
2. Each entity shall have access to a bit string *MacKey* of length *mackeylen* bits to be used as the key. Each party shall have evidence that access to this key is restricted to the entity itself and the other entity involved in the mutual entity authentication scheme.
3. Each entity shall be bound to a unique identifier (for example, distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*'s identifier will be denoted by the bit string *U*. Entity *V*'s identifier will be denoted by the bit string *V*.
4. Each entity shall have decided which MAC scheme to use as specified in Section 5.7 of ANSI X9.63-2001 [B7]. The length in bits of the keys used by the chosen MAC scheme is denoted by *mackeylen*.

5. A fixed representation of octets as binary strings shall have been chosen (for example, most-significant-bit-first order or least-significant-bit-first order).

B.8.1 Initiator Transformation

U shall execute the following transformation to establish authenticity of, and optionally obtain authenticated data from, *V* by means of sharing a key and communicating cooperatively with *V*. *U* shall obtain an authentic copy of *V*'s identifier and an authentic copy of the secret key *MacKey* shared with *V*.

Input: The input to the initiator transformation is:

1. (Optional) A bit string $Text_2$ that consists of some additional data to be provided from *U* to *V*.

Ingredients: The initiator transformation employs the challenge generation primitive specified in Section 5.3 of ANSI X9.63-2001 [B7], the challenge validation primitive specified in section B.4, and one of the MAC schemes in Section 5.7 of ANSI X9.63-2001 [B7].

Actions: Entity authentication shall be established as follows:

1. Use the challenge generation primitive given in Section 5.3 of ANSI X9.63- 2001 [B7] to generate a challenge *QEU* for the challenge domain parameters D. Send *QEU* to *V*.
2. Then receive from *V* a challenge *QEV'* purportedly owned by *V*. If this value is not received, output 'invalid' and stop.
3. Verify that *QEV'* is a valid challenge for the challenge domain parameters *D* as specified in section B.3.1. If the validation primitive rejects the challenge, output 'invalid' and stop.
4. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU* corresponding to *U*'s challenge, the bit string *QEV'* corresponding to *V*'s purported challenge, and optionally a bit string $Text_2$:

   $MacData_2 = 03_{16} \,||\, \mathrm{U} \,||\, \mathrm{V} \,||\, \mathrm{QEU} \,||\, \mathrm{QEV'} \,||\, [\mathrm{Text}_2]$.
5. Calculate the tag $MacTag_2$ on $MacData_2$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7.1 of ANSI X9.63-2001 [B7]:

   $MacTag_2 = \mathrm{MAC}_{MacKey}\,(MacData_2)$.

   If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $MacTag_2$ and, if present, bit string $Text_2$ to *V*.
6. Receive from *V* an optional bit string $Text_1$, and a purported tag $MacTag_1'$. If these values are not received, output 'invalid' and stop.
7. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV'* corresponding to *V*'s purported challenge, the bit string *QEU* corresponding to *U*'s challenge, and if present $Text_1$:

   $MacData_1 = 02_{16} \,||\, V \,||\, U \,||\, QEV' \,||\, QEU \,||\, [Text_1]$.
8. Verify that $MacTag_1'$ is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7.2 of ANSI X9.63-2001 [B7]. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

Output: If any of the above verifications has failed, then output 'invalid' and reject the authenticity of *V* and reject the entity authentication from *V*. Otherwise, output 'valid', accept the authenticity of *V* and accept the entity authentication from *V* of the authenticated bit string $Text_1$ (if present).

B.8.2 Responder Transformation

V shall execute the following transformation to establish authenticity, of and optionally obtain authenticated data from, *U* by means of sharing a key and communicating cooperatively with *U*. *V* shall obtain an authentic copy of *U*'s identifier and an authentic copy of the secret key *MacKey* shared with *U*.

Input: The input to the responder transformation is:

1. A challenge *QEU'* purportedly owned by *U*.
2. (Optional) A bit string $Text_1$ that consists of some additional data to be provided from *V* to *U*.

Ingredients: The responder transformation employs the challenge generation primitive specified in Section 5.3 of ANSI X9.63-2001 [B7], the challenge validation primitive specified in section B.4, and one of the MAC schemes in Section 5.7 of ANSI X9.63-2001 [B7].

Actions: Entity authentication shall be established as follows:

1. Verify that *QEU'* is a valid challenge for the challenge domain parameters *D* as specified in section B.3.1. If the validation primitive rejects the challenge, output 'invalid' and stop.
2. Use the challenge generation primitive in Section 5.3 of ANSI X9.63-2001 [B7] to generate a challenge *QEV* for the challenge domain parameters *D*. Send *QEV* to *U*.
3. Then receive from *U* an optional bit string $Text_2$ and a purported tag $MacTag_2'$. If this data is not received, output 'invalid' and stop.
4. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU'* corresponding to *U*'s purported challenge, the bit string *QEV* corresponding to *V*'s challenge, and the bit string $Text_2$ (if present):

   $MacData_2 = 03_{16} \,||\, U \,||\, V \,||\, QEU' \,||\, QEV \,||\, [Text_2]$
5. Calculate the tag $MacTag_2$ for $MacData_2$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7.1 of ANSI X9.63-2001 [B7]:

   $MacTag_2 = MAC_{MacKey}(MacData_2)$.

   If the tagging transformation outputs 'invalid', output 'invalid' and stop.
6. Verify that $MacTag_2'$ is the valid tag on $MacData_2$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7.2 of ANSI X9.63-2001 [B7]. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.
7. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV* corresponding to *V*'s challenge, the bit string *QEU'* corresponding to *U*'s purported challenge and optionally a bit string $Text_1$:

   $MacData_1 = 02_{16} \,||\, V \,||\, U \,||\, QEV \,||\, QEU' \,||\, [Text_1]$.
8. Calculate the tag $MacTag_1$ for $MacData_1$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7.1 of ANSI X9.63-2001 [B7]:

   $MacTag_1 = MAC_{MacKey}(MacData_1)$.

   If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send MacTag1 and, if present, bit string $Text_1$ to *U*.

Output: If any of the above verifications has failed, then output 'invalid' and reject the authenticity of *U* and reject the entity authentication from *U*. Otherwise, output 'valid', accept the authenticity of *U* and accept the entity authentication from *U* of the authenticated bit string $Text_2$ (if present).

This page intentionally left blank.

ANNEX C TEST VECTORS FOR CRYPTOGRAPHIC BUILDING BLOCKS

This annex provides sample test vectors for the ZigBee community, aimed at with the intent of assisting in building interoperable security implementations. The sample test vectors are provided as is, pending independent validation.

C.1 Data Conversions

For test vectors, see Appendix J1 of ANSI X9.63-2001 [B7].

C.2 AES Block Cipher

This annex provides sample test vectors for the block-cipher specified in section B.1.1.

For test vectors, see FIPS Pub 197 [B8].

C.3 CCM* Mode Encryption and Authentication Transformation

This annex provides sample test vectors for the mode of operation as specified in section B.1.2.

Prerequisites: The following prerequisites are established for the operation of the mode of operation:

1. The parameter *M* shall have the integer value 8.

Input: The inputs to the mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF

2. The nonce *N* of 15-*L*=13 octets to be used:

   Nonce = A0 A1 A2 A3 A4 A5 A6 A7 || 03 02 01 00 || 06

3. The octet string *m* of length *l(m)=23* octets to be used:

   m = 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E

4. The octet string *a* of length *l(a)*=8 octets to be used:

   a = 00 01 02 03 04 05 06 07

C.3.1 Input Transformation

This step involves the transformation of the input strings *a* and *m* to the strings *AuthData* and *PlainTextData*, to be used by the authentication transformation and the encryption transformation, respectively.

1. Form the octet string representation *L(a)* of the length *l(a)* of the octet string *a*:

   L(a) = 00 08

2. Right-concatenate the octet string *L(a)* and the octet string *a* itself:

   L(a) || a = 00 08 || 00 01 02 03 04 05 06 07

3. Form the padded message *AddAuthData* by right-concatenating the resulting string with the smallest non-negative number of all-zero octets such that the octet string *AddAuthData* has length divisible by 16:

   AddAuthData = 00 08 || 00 01 02 03 04 05 06 07 || 00 00 00 00 00 00

4. Form the padded message *PlaintextData* by right-concatenating the octet string m with the smallest non-negative number of all-zero octets such that the octet string *PlaintextData* has length divisible by 16:

   PlaintextData = 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 ||
   18 19 1A 1B 1C 1D 1E || 00 00 00 00 00 00 00 00 00

5. Form the message *AuthData* consisting of the octet strings *AddAuthData* and *PlaintextData*:

   AuthData = 00 08 00 01 02 03 04 05 06 07 00 00 00 00 00 00 ||
   08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
   18 19 1A 1B 1C 1D 1E 00 00 00 00 00 00 00 00 00

C.3.2 Authentication Transformation

The data *AuthData* that was established above shall be tagged using the following tagging transformation:

1. Form the 1-octet *Flags* field as follows:

   Flags = 59

2. Form the 16-octet B_0 field as follows:

   B_0 = 59 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 17

3. Parse the message *AuthData* as $B_1 \| B_2 \| B_3$, where each message block B_i is a 16-octet string.

4. The CBC-MAC value X_4 is calculated as follows:

i	B_i	X_i
0	*59 A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 00 17*	*00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00*
1	*00 08 00 01 02 03 04 05 06 07 00 00 00 00 00 00*	*F7 74 D1 6E A7 2D C0 B3 E4 5E 36 CA 8F 24 3B 1A*
2	*08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17*	*90 2E 72 58 AE 5A 4B 5D 85 7A 25 19 F3 C7 3A B3*
3	*18 19 1A 1B 1C 1D 1E 00 00 00 00 00 00 00 00 00*	*5A B2 C8 6E 3E DA 23 D2 7C 49 7D DF 49 BB B4 09*
4	*æ*	*B9 D7 89 67 04 BC FA 20 B2 10 36 74 45 F9 83 D6*

The authentication tag T is the result of omitting all but the leftmost M=8 octets of the CBC-MAC value X_4:

T = B9 D7 89 67 04 BC FA 20

C.3.3 Encryption Transformation

The data *PlaintextData* shall be encrypted using the following encryption transformation:

1. Form the 1-octet Flags field as follows:

   Flags = 01

2. Define the 16-octet A_i field as follows:

i	A_i
0	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 00
1	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 01
2	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 02

3. Parse the message *PlaintextData* as $M_1 \| M_2$, where each message block M_i is a 16-octet string.
4. The ciphertext blocks C_1, C_2 are computed as follows:

i	AES(Key,A_i)	$C_i = AES(Key,A_i) \oplus M_i$
1	12 5C A9 61 B7 61 6F 02 16 7A 21 66 70 89 F9 07	1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10
2	CC 7F 54 D1 C4 49 B6 35 46 21 46 03 AA C6 2A 17	D4 66 4E CA D8 54 A8 35 46 21 46 03 AA C6 2A 17

5. The string *Ciphertext* is the result of omitting all but the leftmost *l(m)*=23 octets of the string $C_1 \| C_2$:

   CipherText = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8

6. Define the 16-octet encryption block S_0 by:

   S_0 = E(Key, A_0)= B3 5E D5 A6 DC 43 6E 49 D6 17 2F 54 77 EB B4 39

7. The encrypted authentication tag *U* is the result of XOR-ing the string consisting of the leftmost *M*=8 octets of S_0 and the authentication tag *T*:

   U = 0A 89 5C C1 D8 FF 94 69

   Output: the right-concatenation *c* of the encrypted message *Ciphertext* and the encrypted authentication tag *U*:

   c = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8 || 0A 89 5C C1 D8 FF 94 69

C.4 CCM* Mode Decryption and Authentication Checking Transformation

This annex provides sample test vectors for the inverse of the mode of operation as specified in section B.1.2.

Prerequisites: The following prerequisites are established for the operation of the mode of operation:

1. The parameter *M* shall have the integer value 8.

Input: The inputs to the inverse mode of operation are:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF

2. The nonce *N* of 15-*L*=13 octets to be used:

   Nonce = A0 A1 A2 A3 A4 A5 A6 A7 || 03 02 01 00 || 06

3. The octet string *c* of length *l(c)*=31 octets to be used:

   c = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8 || 0A 89 5C C1 D8 FF 94 69

4. The octet string *a* of length *l(a)*=8 octets to be used:

   a = 00 01 02 03 04 05 06 07

C.4.1 Decryption Transformation

The decryption transformation involves the following steps, in order:

1. Parse the message *c* as *C* ||*U*, where the rightmost string *U* is an *M*-octet string:

   C = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8;
   U = 0A 89 5C C1 D8 FF 94 69

2. Form the padded message *CiphertextData* by right-concatenating the string *C* with the smallest non-negative number of all-zero octets such that the octet string *CiphertextData* has length divisible by 16.

   CipherTextData = 1A 55 A3 6A BB 6C 61 0D 06 6B 33 75 64 9C EF 10 || D4 66 4E CA D8 54 A8 || 00 00 00 00 00 00 00 00

3. Form the 1-octet *Flags* field as follows:

   Flags = 01

4. Define the 16-octet A_i field as follows:

i	A_i
0	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 00
1	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 01
2	01 || A0 A1 A2 A3 A4 A5 A6 A7 03 02 01 00 06 || 00 02

5. Parse the message *CiphertextData* as $C_1 \| C_2$, where each message block C_i is a 16-octet string.
6. The ciphertext blocks P_1, P_2 are computed as follows.

I	AES(Key,A_i)	P_i= AES(Key,A_i) ⊕ C_i
1	12 5C A9 61 B7 61 6F 02 16 7A 21 66 70 89 F9 07	08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
2	CC 7F 54 D1 C4 49 B6 35 46 21 46 03 AA C6 2A 17	18 19 1A 1B 1C 1D 1E 00 00 00 00 00 00 00 00 00

7. The octet string *m* is the result of omitting all but the leftmost *l(m)*=23 octets of the string $P_1 \| P_2$:

   m = 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 || 18 19 1A 1B 1C 1D 1E

8. Define the 16-octet encryption block S_0 by

   $S_0 = E(Key, A_0)$ = B3 5E D5 A6 DC 43 6E 49 D6 17 2F 54 77 EB B4 39

9. The purported authentication tag *T* is the result of XOR-ing the string consisting of the leftmost *M*=8 octets of S_0 and the octet string *U*:

   T = B9 D7 89 67 04 BC FA 20

C.4.2 Authentication Checking Transformation

The authentication checking transformation involves the following steps:

1. Form the message *AuthData* using the input transformation in Input Transformation, with the string *a* as inputs and the octet string *m* that was established in section 1.4.1 (step 7):

   AuthData = 00 08[1] 01 02 03 04 05 06 07 00 00 00 00 00 00 00 ||
   08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
   18 19 1A 1B 1C 1D 1E 00 00 00 00 00 00 00 00 00

2. Use the authentication transformation in section C.3.2, with the message *AuthData* to compute the authentication tag *MACTag* as input:

   MACTag = B9 D7 89 67 04 BC FA 20

3. Compare the output tag *MACTag* resulting from this transformation with the tag *T* that was established in section 4.1 (step 9):

   T = B9 D7 89 67 04 BC FA 20 = *MACTag*

[1] CCB 1520

Output: Since *MACTag=T*, output 'valid' and accept the octet string *m* and accept one of the key sharing group member(s) as the source of *m*.

C.5 Cryptographic Hash Function

This annex provides sample test vectors for the cryptographic hash function specified in clause B.1.3.

C.5.1 Test Vector Set 1

Input: The input to the cryptographic hash function is as follows:

1. The bit string *M* of length *l*=8 bits to be used:

   M=C0

Actions: The hash value shall be derived as follows:

1. Pad the message *M* by right-concatenating to *M* the bit '1' followed by the smallest non-negative number of '0' bits, such that the resulting string has length 14 (**mod** 16) octets:

   C0 || 80 00 00 00 00 00 00 00 00 00 00 00 00

2. Form the padded message M' by right-concatenating to the resulting string the 16-bit string that is equal to the binary representation of the integer l:

   M' = C0 || 80 00 00 00 00 00 00 00 00 00 00 00 00 || 00 08

3. Parse the padded message *M'* as M_1, where each message block M_i is a 16-octet string.
4. The hash value Hash1 is computed as follows:

i	**Hash$_i$**	**M$_i$**
0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	æ
1	AE 3A 10 2A 28 D4 3E E0 D4 A0 9E 22 78 8B 20 6C	C0 80 00 00 00 00 00 00 00 00 00 00 00 00 00 08

Output: the 16-octet string $Hash = Hash_1$ = AE 3A 10 2A 28 D4 3E E0 D4 A0 9E 22 78 8B 20 6C.

C.5.2 Test Vector Set 2

Input: The input to the cryptographic hash function is as follows:

1. The bit string *M* of length *l*=128 bits to be used:

   M=C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF

Actions: The hash value shall be derived as follows:

1. Pad the message *M* by right-concatenating to *M* the bit '1' followed by the smallest non-negative number of '0' bits, such that the resulting string has length 14 (**mod** 16) octets:

   C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF ||
   80 00 00 00 00 00 00 00 00 00 00 00 00

2. Form the padded message *M'* by right-concatenating to the resulting string the 16-bit string that is equal to the binary representation of the integer *l*:

M' = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF ||
80 00 00 00 00 00 00 00 00 00 00 00 00 00 || 00 80

3. Parse the padded message M' as $M_1 \| M_2$, where each message block M_i is a 16-octet string.

4. The hash value $Hash_2$ is computed as follows:

i	Hash$_i$	M$_i$
0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	æ
1	84 EE 75 E5 4F 9A 52 0F 0B 30 9C 35 29 1F 83 4F	C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
2	A7 97 7E 88 BC 0B 61 E8 21 08 27 10 9A 22 8F 2D	80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08

Output: the 16-octet string $Hash = Hash_2$ = A7 97 7E 88 BC 0B 61 E8 21 08 27 10 9A 22 8F 2D.

C.5.3 Test Vector Set 3

Input: The input to the cryptographic hash function is as follows:

1. The bit string M of length l = 65528 bits to be used.
2. 8191 bytes (sequence of 0, 1, 2, … 255, 0, 1, 2, …)
3. This test vector is beneath the threshold of a 216 bit string so the first padding method described in clause B.6 is utilized.

Actions: The hash value shall be derived as follows:

1. Pad the message by right-concatenating to M the bit 1 followed by the smallest non-negative number of '0' bits, such that the resulting string has length 14 (mod 16) octets:

   00 01 02 03 04 … FB FC FD FE || 80 00 00 00 00 00 00 00 00 00 00 00 00 00

2. Form the padded message M' by right-concatenating to the resulting string the 16-bit string that is equal to the binary representation of the integer l:

   00 01 02 03 04 … FB FC FD FE || 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 || FF F8

3. Parse the padded message M' as M_1, where each message block M_i is a 16-octet string.
4. The hash value $Hash_1$ is computed as follows using 16-byte hash block operations:

i	Hash$_i$	M$_i$
0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	-
1	7A CB 0D DA B8 D3 EA 7B 97 9E 4C 6D 1A EB AC 8D	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
...	...	...
i - 1	C3 22 D1 D3 9D 10 86 43 82 06 BD EB 26 41 66 1C	F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE 80
i	24 EC 2F E7 5B BF FC B3 47 89 BC 06 10 E7 F1 65	00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF F8

C.5.4 Test Vector 4

Input: The input to the cryptographic hash function is as follows:

1. The bit string M of length l = 65536 bits to be used.
2. 8192 bytes (sequence of 0, 1, 2, … 255, 0, 1, 2, …)
3. This test vector is above the threshold of a 216 bit string so the second padding method described in clause B.6 is utilized.

Actions: The hash value shall be derived as follows.

1. Pad the message by right-concatenating to *M* the bit 1 followed by the smallest non-negative number of '0' bits, such that the resulting string has length 10 (mod 16) octets:

   00 01 02 03 04 … FB FC FD FE FF || 80 00 00 00 00 00 00 00 00 00

2. Form the padded message M' by right-concatenating to the resulting string the 32-bit string that is equal to the binary representation of the integer l:

   00 01 02 03 04 … FB FC FD FE FF || 80 00 00 00 00 00 00 00 00 00 || 00 01 00 00

3. Concatenate a 16-bit string of zeros for the padding normally used by the first padding method described in clause B.6.

   00 01 02 03 04 … FB FC FD FE FF || 80 00 00 00 00 00 00 00 00 00 || 00 01 00 00 || 00 00

4. Parse the padded message M' as M_1, where each message block M_i is a 16-octet string.
5. The hash value $Hash_1$ is computed as follows using 16-byte hash block operations:

i	$Hash_i$	M_i
0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	-
1	7A CB 0D DA B8 D3 EA 7B 97 9E 4C 6D 1A EB AC 8D	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
...	...	...
i - 1	4E 55 0D CE 34 31 42 96 41 BA D0 C7 BC 44 34 67	F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
i	DC 6B 06 87 F0 9F 86 07 13 1C 17 0B 3B D3 15 91[2]	80 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00

C.5.5 Test Vector 5

Input: The input to the cryptographic hash function is as follows:

1. The bit string M of length l = 65608 bits to be used.
2. 8201 bytes (sequence of 0, 1, 2, … 255, 0, 1, 2, …)
3. This test vector is above the threshold of a 216 bit string so the second padding method described in clause B.6 is utilized.

[2] CCB 1519

Actions: The hash value shall be derived as follows.

1. Pad the message by right-concatenating to *M* the bit 1 followed by the smallest non-negative number of '0' bits, such that the resulting string has length 10 (mod 16) octets:

   00 01 02 03 04 … 04 05 06 07 08 || 80

2. Form the padded message *M'* by right-concatenating to the resulting string the 32-bit string that is equal to the binary representation of the integer l:

   00 01 02 03 04 … 04 05 06 07 08 || 80 || 00 01 00 48

3. Concatenate a 16-bit string of zeros for the padding normally used by the first padding method described in clause B.6.

   00 01 02 03 04 … 04 05 06 07 08 || 80 || 00 01 00 48 || 00 00

4. Parse the padded message M' as M_1, where each message block M_i is a 16-octet string.
5. The hash value $Hash_1$ is computed as follows using 16-byte hash block operations:

i	$Hash_i$	M_i
0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	-
1	7A CB 0D DA B8 D3 EA 7B 97 9E 4C 6D 1A EB AC 8D	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
...	...	...
i - 1	4E 55 0D CE 34 31 42 96 41 BA D0 C7 BC 44 34 67	F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
i	72 C9 B1 5E 17 8A A8 43 E4 A1 6C 58 E3 36 43 A3	00 01 02 03 04 05 06 07 08 80 00 01 00 48 00 00

C.5.6 Test Vector 6

Input: The input to the cryptographic hash function is as follows:

1. The bit string M of length l = 65616 bits to be used.
2. 8202 bytes (sequence of 0, 1, 2, … 255, 0, 1, 2, …)
3. This test vector is above the threshold of a 216 bit string so the second padding method described in clause B.6 is utilized.

Actions: The hash value shall be derived as follows.

1. Pad the message by right-concatenating to M the bit 1 followed by the smallest non-negative number of '0' bits, such that the resulting string has length 10 (mod 16) octets:

   00 01 02 03 04 … 05 06 07 08 09 || 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00

2. Form the padded message *M'* by right-concatenating to the resulting string the 32-bit string that is equal to the binary representation of the integer l:

   00 01 02 03 04 … 05 06 07 08 09 || 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 || 00 01 00 50

3. Concatenate a 16-bit string of zeros for the padding normally used by the first padding method described in clause B.6.

   00 01 02 03 04 … 05 06 07 08 09 || 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 || 00 01 00 50 || 00 00

4. Parse the padded message M' as M_1, where each message block M_i is a 16-octet string.

5. The hash value $Hash_1$ is computed as follows using 16-byte hash block operations:

i	$Hash_i$	M_i
0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	-
1	7A CB 0D DA B8 D3 EA 7B 97 9E 4C 6D 1A EB AC 8D	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
...	...	...
i - 1	CC C1 F8 A3 D5 6A 93 20 41 08 10 2B 46 25 0D A7	00 01 02 03 04 05 06 07 08 09 80 00 00 00 00 00
i	BC 98 28 D5 9B 2A A3 23 DA F2 0B E5 F2 E6 65 11	00 00 00 00 00 00 00 00 00 00 00 01 00 50 00 00

C.6 Keyed Hash Function for Message Authentication

This annex provides sample test vectors for the keyed hash function for message authentication as specified in clause B.1.4.

C.6.1 Test Vector Set 1

Input: The input to the keyed hash function is as follows:

1. The key *Key* of size *keylen*=128 bits to be used:

   Key = 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F

2. The bit string *M* of length *l*=8 bits to be used:

   M=C0

Actions: The keyed hash value shall be derived as follows:

1. Create the 16-octet string *ipad* (inner pad) as follows:

   ipad = 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36

2. Form the inner key Key_1 by XOR-ing the bit string *Key* and the octet string *ipad*:

   $Key_1 = Key \oplus ipad$ = 76 77 74 75 72 73 70 71 7E 7F 7C 7D 7A 7B 78 79

3. Form the padded message M_1 by right-concatenating the bit string Key_1 with the bit string *M*:

   $M_1 = Key_1 \| M$ = 76 77 74 75 72 73 70 71 7E 7F 7C 7D 7A 7B 78 79 || C0

4. Compute the hash value $Hash_1$ of the bit string M_1:

   $Hash_1$ = 3C 3D 53 75 29 A7 A9 A0 3F 66 9D CD 88 6C B5 2C

5. Create the 16-octet string *opad* (outer pad) as follows:

   opad = 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C

6. Form the outer key Key_2 by XOR-ing the bit string *Key* and the octet string *opad*:

   $Key_2 = Key \oplus opad$ = 1C 1D 1E 1F 18 19 1A 1B 14 15 16 17 10 11 12 13

7. Form the padded message M_2 by right-concatenating the bit string Key_2 with the bit string $Hash_1$:

   $M_2 = Key_2 \| Hash_1$ = 1C 1D 1E 1F 18 19 1A 1B 14 15 16 17 10 11 12 13 ||
   3C 3D 53 75 29 A7 A9 A0 3F 66 9D CD 88 6C B5 2C

8. Compute the hash value $Hash_2$ of the bit string M_2:

   $Hash_2$ = 45 12 80 7B F9 4C B3 40 0F 0E 2C 25 FB 76 E9 99

Output: the 16-octet string $HMAC = Hash_2$ = 45 12 80 7B F9 4C B3 40 0F 0E 2C 25 FB 76 E9 99

C.6.2 Test Vector Set 2

Input: The input to the keyed hash function is as follows:

1. The key *Key* of size *keylen*=256 bits to be used:

   Key = 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F ||
   50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F

2. The bit string *M* of length *l*=128 bits to be used:

   M = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF

Actions: The keyed hash value shall be derived as follows:

1. Compute the hash value Key_0 of the bit string *Key*:

   Key_0 = 22 F4 0C BE 15 66 AC CF EB 77 77 E1 C4 A9 BB 43

2. Create the 16-octet string *ipad* (inner pad) as follows:

   ipad = 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36

3. Form the inner key Key_1 by XOR-ing the bit key Key_0 and the octet string *ipad*:

   $Key_1 = Key_0 \oplus ipad$ = 14 C2 3A 88 23 50 9A F9 DD 41 41 D7 F2 9F 8D 75

4. Form the padded message M_1 by right-concatenating the bit string Key_1 with the bit string *M*:

   $M_1 = Key_1 \| M$ = 14 C2 3A 88 23 50 9A F9 DD 41 41 D7 F2 9F 8D 75 ||
   C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF

5. Compute the hash value $Hash_1$ of the bit string M_1:

   $Hash_1$ = 42 65 BE 29 74 55 8C A2 7B 77 85 AC 73 F2 22 10

6. Create the 16-octet string *opad* (outer pad) as follows:

   opad = 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C 5C

7. Form the outer key Key_2 by XOR-ing the bit string Key_0 and the octet string *opad*:

   $Key_2 = Key_0 \oplus opad$ = 7E A8 50 E2 49 3A F0 93 B7 2B 2B BD 98 F5 E7 1F

8. Form the padded message M_2 by right-concatenating the bit string Key_2 with the bit string $Hash_1$:

   $M_2 = Key_2 \| Hash_1$ = 7E A8 50 E2 49 3A F0 93 B7 2B 2B BD 98 F5 E7 1F ||
   42 65 BE 29 74 55 8C A2 7B 77 85 AC 73 F2 22 10

9. Compute the hash value $Hash_2$ of the bit string M_2:

$Hash_2$ = A3 B0 07 99 84 BF 15 57 F7 4A 0D 63 87 E0 A1 1A

Output: the 16-octet string $HMAC = Hash_2$ = A3 B0 07 99 84 BF 15 57 F7 4A 0D 63 87 E0 A1 1A

C.6.3 Specialized Keyed Hash Function for Message Authentication

This annex provides sample test vectors for the specialized keyed hash function for message authentication as specified in clause B.1.4.

For test vectors, see clause C.6.

10.

This page intentionally left blank.

ANNEX D MAC AND PHY SUB-LAYER CLARIFICATIONS

D.1 Introduction

D.1.1 Scope

This annex applies to the IEEE 802.15.4 2003 Medium Access Control sub-layer (MAC) and Physical Layer (PHY) specification when used in conjunction with higher layers defined by the ZigBee specification. Nothing is implied about the usage under other circumstances.

D.1.2 Purpose

The current ZigBee specification assumes the use of the MAC and PHY sub-layers defined in the IEEE 802.15.4 2003 specification. However, as developers have put the MAC and PHY sub-layers into use, they have uncovered problems that may or may not have been anticipated by the authors of the specification, or are not covered in the IEEE 802.15.4 2003 specification. This document is intended to provide solutions to such problems, for use by the ZigBee Alliance.

D.2 Stack Size Issues

Both MAC and ZigBee stack developers have discovered that implementation of a full-blown MAC is a major undertaking and requires a great deal of code space. Even with the optional GTS and MAC security features eliminated, it is not surprising to find the MAC taking up more than 24K of code space on a processor with 64K of available space.

The ZigBee Alliance has adopted a compensating policy to declare MAC features that are not required to support a particular stack profile optional with respect to that stack profile. In particular, any MAC feature that will not be exploited as a result of platform compliance testing for a particular stack profile need not be present in order for an implementation to be declared platform compliant. For example, since the ZigBee Pro stack profile relies on a beaconless network, the platform compliance testing for the stack profile does not employ beaconing. The code to support regular beaconing, beacon track, and so on, may therefore be absent from the code base of the device under test without the knowledge of the testers, without presenting a problem with respect to platform compliance certification.

The exact list of MAC features that must be supported in a platform is described in the PICS document used for MAC conformance testing.

D.3 MAC Association

At association time, according to the IEEE 802.15.4 specification, a number of frames are sent, including an association request command, an associate response command and a data request. There is some ambiguity in the specification regarding the addressing fields in the headers for these frames. Table D.1 to Table D.3 outline the allowable options that shall be recognized by devices implementing the ZigBee specification. In each case, the first option given is the preferred option and should be used.

Table D.1 Associate Request Header Fields

DstPANId	DstAddr	SrcPANId	SrcAddr
The PANId of the destination device.	The 16-bit short address of the destination device.	0xffff	The 64-bit extended address of the source device.
		PANId omitted because the IntraPAN sub-field in the frame control field is set to one.	
		The PANId of the destination device.	
Not present if the destination device is the PAN coordinator.	Not present if the destination device is the PAN coordinator.		

Note that in this case and the case below, the source of the command is the device requesting association.

Table D.2 Data Request Header Fields

DstPANId	DstAddr	SrcPANId	SrcAddr
The PANId of the destination device.	The 16-bit short address of the destination device.	0xffff	The 64-bit extended address of the source device.
		PANId omitted because the IntraPAN sub-field in the frame control field is set to one.	
		The PANId of the destination device.	
Not present if the destination device is the PAN coordinator.	Not present if the destination device is the PAN coordinator.		

Table D.3 Association Response Header Fields

DstPANId	DstAddr	SrcPANId	SrcAddr
The PANId of the destination device.	The 64-bit extended address of the destination device.	PANId omitted because the IntraPAN sub-field in the frame control field is set to one.	The 64-bit extended address of the source device.
		The PANId of the source device.	
0xffff			

D.4 aMaxMACFrameSize

The IEEE 802.15.4 MAC specification [B1] has two constants that define the minimum and maximum values for the MAC data packet payload size. These are the *aMaxMACPayloadSize* (118 bytes) and the *aMaxMACSafePayloadSize* (102 bytes). Since the overhead imposed by the MAC header is variable, the actual limit of the MAC data payload size is in between these values and may vary by implementation.

When used in a ZigBee platform, the MAC implementation must support transmission and reception of unsecured MAC data packet payloads of up to (*aMaxPHYPacketSize* - *nwkcMinHeaderOverhead*) bytes. The value of *nwkcMinHeaderOverhead* parameter takes into account the fact that ZigBee uses short addressing modes and intra-PAN communications.

D.5 Frame Version Value

The MAC specification requires that any unsecured MAC data packet with payload size greater than aMaxMACSafePayloadSize (102bytes) must have the Frame Version field set to one (see section 6.3.1 of [B1]). When used in a ZigBee platform, the MAC implementation must always set the Frame Version field in unsecured MAC data packets to zero. The reason for this is to ensure backwards compatibility with existing deployed ZigBee devices that cannot receive packets correctly if these bits are set to a non-zero value. Note that this deviation is only on the transmit side, the receive side processing is unchanged. That is, the MAC implementation must be able to receive and process MAC data packets with the Frame Version field set to any non-reserved value, as specified in section 5.6.1.2 of [B1].

The MAC specification allows the coordinator realignment command to be sent with either Frame Version of zero or one. The format of the command is different in each case (see section 5.3.8.1 of [B1]). When used in a ZigBee implementation, the MAC implementation must always set the Frame Version field in the coordinator realignment command to zero.

D.6 CSMA Backoff Timing

The IEEE 802.15.4 2006 specification provides an increase in macMaxBE to 8 from 5. This higher value is allowed within ZigBee and it is recommended as the default. The default value of macMinBE should be 5 instead of 3. This provides better joining performance in dense networks where many devices may be responding to a beacon request.

Note the time a device listens for beacons is set by IEEE 802.15.4 to aBaseSuperframeDuration*(2n+1) symbols where n is the value of the *ScanDuration* parameter. For ZigBee implementations the value of n should be set to ensure the duration of the listening window is similar to the length of time the beacon responses are expected.

D.7 MAC Interface Changes

The IEEE-802-15-4 specification has no notification when a MAC data poll is received by a coordinator (FFD) or any ability for the ZigBee layers to dictate the response to the MAC data poll. Therefore the following interfaces are defined for a MAC used by ZigBee network layers.

D.7.1 Additional Primitives accessed through the MLME-SAP

Those primitives marked with a diamond (◊) are optional for an RFD.

Name	Request	Indication	Response	Confirm
MLME-Poll	(Already specified in reference B1)	D.7.2 ◊	-	(Already specified in reference B1)

D.7.2 MLME-POLL.indication

The MLME-Poll.indication primitive notifies the next higher level that a request for data has been received.

D.7.2.1 Semantics of the service primitive

The semantics of the MLME-Poll.indication primitive is as follows.

MLME-Poll.indication (

AddrMode

DeviceAddress

)

Name	Type	Valid Range	Description
AddrMode	Integer	0x02 – 0x03	This value can take one of the following values: 2=16 bit short address. 3=64 bit extended address.
DeviceAddress	Integer	As specified by Ad-	The address of the device requesting pending

		drMode parameter.	data.

D.7.2.2 When Generated

The MLME-POLL.indication primitive indicates the reception of a Data request command frame by the MAC sub-layer and issued to the local SSCS (service specific convergence sublayer).

D.7.2.3 Effect on Receipt

The effect on receipt of the MLME-Poll.indication primitive is that the next higher layer is notified that a device is requesting to see if there is a pending MAC data frame. If an indirect frame is queued by the higher layer during the processing of an MLME-POLL.indication it shall affect the pending bit in the ACK frame corresponding to the data request frame that caused the MLME-POLL.indication to be issued.

This page intentionally left blank.

ANNEX E OPERATING NETWORK MANAGER AS NETWORK CHANNEL MANAGER FOR INTERFERENCE REPORTING AND RESOLUTION

Prerequisites: Devices shall limit their operations to channels within their current PHY (i.e. 868/915 MHz or 2450 MHz). Commands including channels outside the band shall be ignored.

A single device can become the Network Channel Manager. This device acts as the central mechanism for reception of network interference reports and changing the channel of the network if interference is detected. The default address of the network manager is the coordinator, however this can be updated by sending a Mgmt_NWK_Update_req command with a different short address for the network channel manager. The device that is the Network Channel Manager shall set the network manager bit in the server mask in the node descriptor and shall respond to System_Server_Discovery_req commands.

Each router or coordinator is responsible for tracking transmit failures using the TransmitFailure field in the neighbor table and also keeping a NIB counter for total transmissions attempted. A device that detects a significant number of transmission failures may take action to determine if interference is a cause. The following steps are an example of that procedure[1]:

1. Conduct an energy scan on all channels within the current PHY. If this energy scan does not indicate higher energy on the current channel then other channels, no action is taken. The device should continue to operate as normal and the message counters are not reset. However, repeated energy scans are not desirable as the device is off the network during these scans and therefore implementations should limit how often a device with failures conducts energy scans.
2. If the energy scan does indicate increased energy on the channel in use, a Mgmt_NWK_Update_notify should be sent to the Network Manager to indicate interference is present. This report is sent as an APS Unicast with acknowledgement and once the acknowledgment is received the total transmit and transmit failure counters are reset to zero.
3. To avoid a device with communication problems from constantly sending reports to the network manager, the device should not send a Mgmt_NWK_Update_notify more than 4 times per hour.

Upon receipt of an unsolicited Mgmt_NWK_Update_notify, the network manager must evaluate if a channel change is required in the network. The specific mechanisms the network manager uses to decide upon a channel change are left to the implementers. It is expected that implementers will apply different methods to best determine when a channel change is required and how to select the most appropriate channel. The following is offered as guidance for implementation.

[1] CCB 1493

The network manager may do the following:

1. Wait and evaluate if other reports from other devices are received. This may be appropriate if there are no other failures reported. In this case the network manager should add the reporting device to a list of devices that have reported interference. The number of devices on such a list would depend on the size of the network. The network manager can age devices out of this list.
2. Request other interference reports using the Mgmt_NWK_Update_req command. This may be done if other failures have been reported or the network manager device itself has failures and a channel change may be desired. The network manager may request data from the list of devices that have reported interference plus other randomly selected routers in the network. The network manager should not request an update from the device that has just reported interference since this data is fresh already.
3. Upon receipt of the Mgmt_NWK_Update_notify, the network manager shall determine if a channel change is required using whatever implementation specific mechanisms are considered appropriate. The network manager device with just one channel allowed in the *apsChannelMask* parameter must not issue the Mgmt_Nwk_Update_Req command to request other devices to change the current channel. However, the network manager may report channel quality issues to the application.
4. If the above data indicate a channel change should be considered, the network manager completed the following:
    a. Select a single channel based on the Mgmt_NWK_Update_notify based on the lowest energy. This is the proposed new channel. If this new channel does not have an energy level below an acceptable threshold, a channel change should not be done. Additionally, a new channel shall not belong to a PHY different from the one on which a network manager is operating now.
5. Prior to changing channels, the network manager should store the energy scan value as the last energy scan value and the failure rate from the existing channel as the last failure rate. These values are useful to allow comparison of the failure rate and energy level on the previous channel to evaluate if the network is causing its own interference.
6. The network manager should broadcast a Mgmt_NWK_Update_req notifying devices of the new channel. The broadcast shall be to all devices with RxOnWhenIdle equal to TRUE. The network manager is responsible for incrementing the *nwkUpdateId* parameter from the NIB and including it in the Mgmt_NWK_Update_req. The network manager shall set a timer based on the value of *apsChannelTimer* upon issue of a Mgmt_NWK_Update_req that changes channels and shall not issue another such command until this timer expires. However, during this period, the network manager can complete the above analysis. However, instead of changing channels, the network manager would report to the local application using Mgmt_NWK_Update_notify and the application can force a channel change using the Mgmt_NWK_Update_req.

Upon receipt of a Mgmt_NWK_Update_req with a change of channels, the local network manager shall set a timer equal to the *nwkNetworkBroadcastDeliveryTime* and shall switch channels upon expiration of this timer. Each node shall also increment the *nwkUpdateId* parameter and also reset the total transmit count and the transmit failure counters.

For devices with RxOnWhenIdle equals FALSE, any network channel change will not be received. On these devices or routers that have lost the network, an active scan shall be conducted on the *apsChannelMask* list in the APS IB using the extended PANID to find the network. If the extended PANID is found on different channels, the device should select the channel with the higher value in the *nwkUpdateId* parameter. If the extended PANID is not found using the *apsChannelMask* list, a scan should be completed using all channels within the current PHY.

ANNEX F USAGE OF MULTIPLE FREQUENCY BANDS

F.1 Introduction

F.1.1 Scope

This annex clarifies uncertainties arising with ZigBee compliant devices that support several frequency bands.

F.1.2 Purpose

The ZigBee specification is based on the IEEE 802.15.4 ([B1]) standard that defines multiple PHYs. A compliant device shall support at least one of the following options: O-QPSK PHY at 2.4 GHz frequency band or the BPSK PHY at both 868 MHz and 915 MHz bands. Each of the frequency bands incorporates its own set of channels through a combination of channel numbers and channel pages. A ZigBee device shall use channel page zero which consists of the following channel numbers: channel 0 for the 868 MHz band, channels from 1 to 10 for the 915 MHz band and channels from 11 to 26 for the 2450 MHz band. Additionally the following apply:

- A Zigbee compliant device declaring support of a frequency band shall support all the channels listed in channel page zero within that frequency band.
- A Zigbee compliant device declaring support of the 868/915 MHz PHY shall support both 868 MHz and 915 MHz frequency bands within this PHY

F.2 Channels and Channel Masks Management General Guideline

F.2.1 Channel Selection During Network Establishment

When there is a set of devices intended to be a part of the same ZigBee network, with devices of that set, potentially, supporting different frequency bands, the coordinator, during network establishment, may choose a channel from a frequency band that is not supported by some of the other devices.

Since, before a network is established, there is no mechanism for the coordinator to dynamically collect information about frequency bands supported on each and every device in the network, this issue may be categorized as a network commissioning issue and has to be resolved in the layers above the ZigBee stack's core.

F.2.2 The Frequency Agility Feature Related Points

How a network manager or a device shall behave, considering the ability to support different frequency bands, is described in Annex E and in section 2.4.3.3.9.2. Implementers of the frequency agility feature should take into account that it is prohibited for a network manager device to move a network from one PHY to another. This limitation is introduced in order to avoid the situations when a part of devices in the network cannot physically migrate to a channel from another PHY and therefore got lost. At the same time moving a network from one frequency band to another within 868/915 MHz PHY is allowed since support of both bands is mandatory in accordance with IEEE P802.15.4 (§C.7.2.3 [B1]). The application layer must meet regional regulatory requirements by setting an appropriate value to the *apsChannelMask* parameter.

F.2.3 Network Management Services and Client Services Affected by Multiple Frequency Bands Support

The following Network Management Client Services and Network Management Services use the *ScanChannels* parameter and, therefore, have to be mentioned in regard of multiple frequency bands support: Mgmt_NWK_Disc_req, Mgmt_NWK_Update_req and NLME-JOIN.request. In case the *ScanChannels* bitmask includes a channel(s) from unsupported frequency band the INVALID_PARAMETER (see [B1]) error status is supposed to be raised from the MAC layer to the NWK layer. If the destination addressing mode in the Mgmt_NWK_Disc_req and Mgmt_NWK_Update_req commands was unicast then the Remote Device shall incorporate the error status into the status field of the correspondent Mgmt_NWK_Disc_rsp and Mgmt_NWK_Update_rsp commands. The same error status shall be reported in NLME-JOIN.confirm primitive sent in response to an NLME-JOIN.request primitive if the latter contains unsupported channels.

In case the NLME-JOIN.request primitive is used by the application layer to request a device to switch to a new channel (the *RejoinNetwork* parameter is equal to 0x03) then the application layer, by implementation-specific means, has to ensure that the chosen channel is supported by all other devices in the network, to avoid the situation when some of the devices might be lost from the network due to inability to switch to an unsupported channel.

F.3 Timing Issues

Different frequency bands declared in the IEEE 802.15.4 2003 standard provide different bit rates. Therefore the ZigBee stack's time-related parameters have to be adjusted accordingly to achieve the stable operation on each of the supported frequency bands. The ZigBee stack's time-related parameters can be divided in two groups in regard of multiple frequency bands support: the first group includes time-related parameters that have a direct impact on the ZigBee stack's core's functioning and that ensure that the core's functioning is correct; the second group consists of the time-related parameters that have to be configured by an application. The ZigBee specification controls the first group of parameters and declares them in a way that makes them dependent on the currently used frequency band. These parameters are presented in Table F.1 and their values must be updated automatically each time a device migrates from one frequency band to another.

Table F.1 Internal Time-related Parameters

Parameter	Reference
:Config_NWK_Time_btwn_Scans	Section 2.5.6.1, Table 2.149
nwkcWaitBeforeValidation	Section 3.5.1, Table 3.43

nwkcRouteDiscoveryTime	Section 3.5.1, Table 3.43
nwkcMaxBroadcastJitter	Section 3.5.1, Table 3.43
nwkcRREQRetryInterval	Section 3.5.1, Table 3.43
nwkcMinRREQJitter	Section 3.5.1, Table 3.43
nwkcMaxRREQJitter	Section 3.5.1, Table 3.43
nwkPassiveAckTimeout	Section 3.5.2, Table 3.44
nwkNetworkBroadcastDeliveryTime	Section 3.5.2, Table 3.44
apsSecurityTimeOutPeriod	Section 4.4.10 Table 4.38

This page intentionally left blank.

ANNEX G INTER-PAN COMMUNICATIONS

G.1 Scope and Purpose

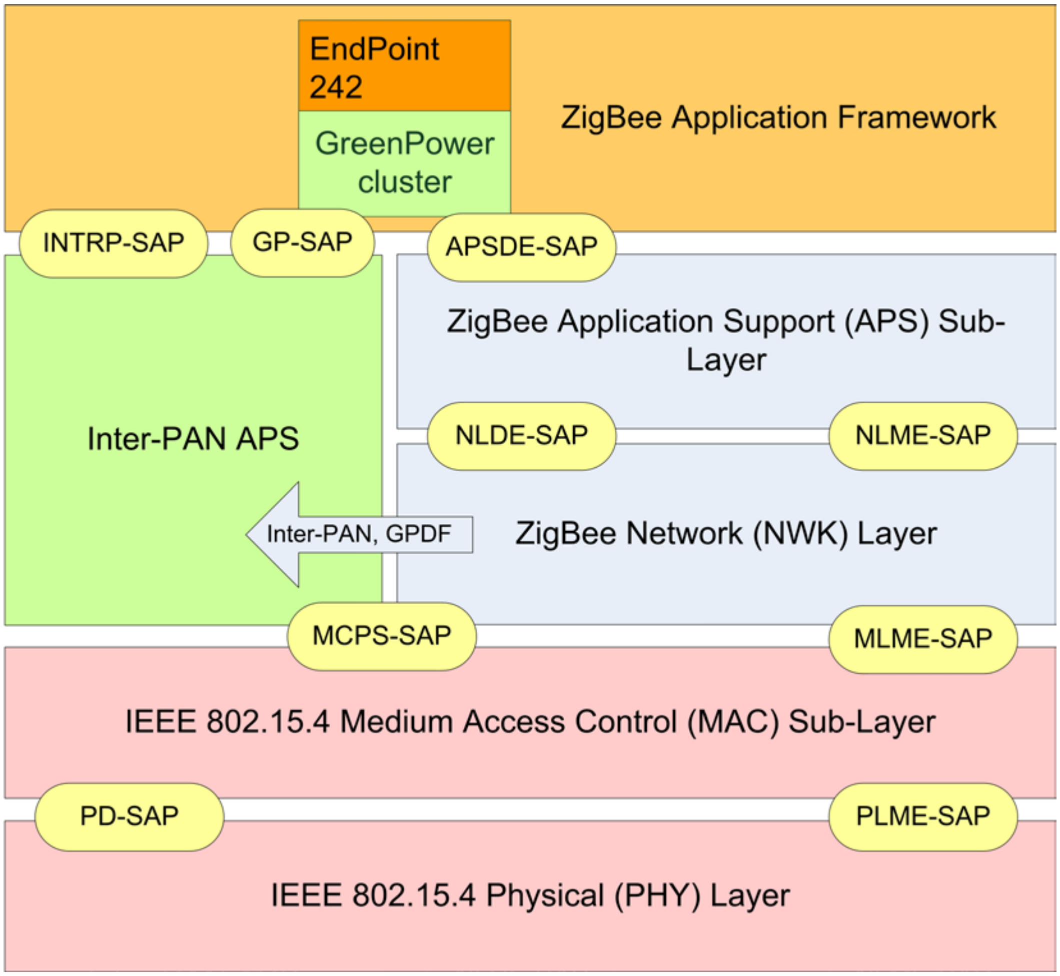
This annex defines a mechanism whereby ZigBee devices can perform exchanges of information with devices in their local area without having to form or join the same ZigBee network. This capability is used in a number of ZigBee functions from extending Smart Energy networks to simple low cost devices, for Green Power end devices, or for Touchlink commissioning.

G.2 General Description

G.2.1 What Inter-PAN APS Does

A schematic view of the ZigBee stack enabling Inter-PAN data and Green Power Device Frame exchange is shown in Figure G.1.

Figure G.1 ZigBee Stack with Inter-PAN APS



Inter-PAN data exchanges and Green Power Device Frame (GPDF) exchanges are handled by a special "stub" of the Application Support Sub-Layer, which is accessible through a special Service Access Point (SAP), the INTRP-SAP, parallel to the APSDE-SAP. The Inter-PAN data exchange architecture is used by several different mechanisms within ZigBee devices.

The same Inter-PAN APS is intended to be used for these different services even if how they use it varies slightly. In case of Inter-PAN data exchanges, the Inter-PAN APS performs just enough processing to pass application data frames to the MAC for transmission and to pass Inter-PAN application frames from the MAC to the application on receipt. In case of Green Power Device Frame exchanges, the Inter-PAN APS also performs security processing of incoming and outgoing GPDF (see section G.5), as well as buffering of outgoing GPDF (see section G.4.3). The incoming GPDF are delivered to the application on endpoint 242 and handled by that; see the specification of the Green Power cluster residing on endpoint 242 [B4].

The use of Inter-PAN frames and Green Power Device Frames is indicated by the sub-fields of the network Frame Control field, as described in section G.3.2.

G.2.2 Service Specification

The INTRP-SAP is a data service comprising eight primitives.

- INTRP-DATA.request - Provides a mechanism for a sending device to request transmission of an Inter-PAN message.
- GP-DATA.request – Provides a mechanism for a sending device to request transmission of a Green Power Device Frame.
- INTRP-DATA.confirm - Provides a mechanism for a sending device to understand the status of a previous request to send an Inter-PAN message.
- GP-DATA.confirm - Provides a mechanism for a sending device to understand the status of a previous request to send a Green Power Device Frame.
- INTRP-DATA.indication - Provides a mechanism for identifying and conveying an Inter-PAN message received from a sending device.
- GP-DATA.indication - Provides a mechanism for identifying and conveying a Green Power Device Frame message received from a sending device.
- GP-SEC.request – provides a mechanism for the Green Power Device Frame processing part of Inter-PAN APS to request security data from the Green Power application.
- GP-SEC.response – provides a mechanism for the Green Power application to provide security data into the Green Power Device Frame processing part of the Inter-PAN APS.

G.2.3 The INTRP-DATA.request Primitive

The INTRP-DATA.request primitive allows an application entity to request data transmission via the Inter-PAN APS.

G.2.3.1 Semantics of the Service Primitive

```
INTRP-DATA.request          {
                            SrcAddrMode
                            DstAddrMode
                            DstPANId
                            DstAddress
                            ProfileId
                            ClusterId
                            ASDULength
                            ASDU
                            ASDUHandle
                            }
```

Table G.1 specifies the parameters of the INTRP-DATA.request primitive.

Table G.1 Semantics of the INTRP-DATA.request Primitive

Name	Type	Valid Range	Description

Name	Type	Valid Range	Description
SrcAddrMode	Integer	0x00 – 0x03	The source addressing mode for the MPDU to be sent. This value can take one of the following values: 0 x 00 = no address (SrcPANId and SrcAddress omitted). 0 x 01 = reserved. 0 x 02 = 16 bit short address. 0 x 03 = 64 bit extended address.
DstAddrMode	Integer	0x01 – 0x03	The addressing mode for the destination address used in this primitive. This parameter can take one of the values from the following list: 0x01 = 16-bit group address 0x02 = 16-bit NWK address, usually the broadcast address 0xffff 0x03 = 64-bit extended address
DstPANID	16-bit PAN ID	0x0000 – 0xFFFF	The 16-bit PAN identifier of the entity or entities to which the ASDU is being transferred or the broadcast PANId 0xffff.
DstAddress	16-bit or 64-bit address	As specified by the AddrMode parameter	The address of the entity or entities to which the ASDU is being transferred.
ProfileId	Integer	0x0000 – 0xffff	The identifier of the application profile for which this frame is intended.
ClusterId	Integer	0x0000 – 0xffff	The identifier of the cluster, within the profile specified by the ProfileId parameter, which defines the application semantics of the ASDU.
ASDULength	Integer	0x00 – (*aMaxMACFrameSiz e* - 9)	The number of octets in the ASDU to be transmitted.
ASDU	Set of octets	-	The set of octets forming the ASDU to be transmitted.
ASDUHandle	Integer	0x00 – 0xff	An integer handle associated with the ASDU to be transmitted.

G.2.3.2 When Generated

This primitive is generated by the local application entity when it wishes to address a frame to one or more peer application entities residing on neighboring devices using Inter-PAN data.

G.2.3.3 Effect on Receipt

On receipt of the INTRP-DATA.request primitive by the Inter-PAN APS, the Inter-PAN APS will construct and transmit an Inter-PAN frame. This frame shall have a Protocol Version sub-field and the Frame Type sub-field of the NWK Frame Control field set to the values as specified in section G.3.2.1. The frame shall contain the given ASDU and set the parameters using the MCPS-DATA.request primitive of the MAC sub-layer, as described in section G.3.1.1. Once the corresponding MCPS-DATA.confirm primitive is received, the stack shall generate the INTRP-DATA.confirm primitive with a status value reflecting the status value returned by the MAC.

G.2.4 The GP-DATA.request Primitive

The GP-DATA.request primitive allows an application entity to request a Green Power data transmission via the Inter-PAN APS.

G.2.4.1 Semantics of the Service Primitive

The primitive interface is as follows:

GP-DATA.request	{
	Actions
	TxOptions
	ApplicationID
	SrcID
	GPD IEEE Address
	GPD CommandID
	ASDULength
	ASDU
	ASDUHandle
	gpTxQueueEntryLifetime
	}

Table G.2 specifies the parameters of the GP-DATA.request primitive.

Table G.2 Parameters of the GP-DATA.request primitive

Name	Type	Valid Range	Description

Name	**Type**	**Valid Range**	**Description**
Actions	Boolean	TRUE/FALSE	TRUE: add Green Power Device Frame into gpTxQueue FALSE: remove Green Power Device Frame from gpTxQueue
TxOptions	8-bit bitmap	Any valid	This provides transmission options for a Green Power Device Frame. There are a bitwise OR of one or more of the following: b0 = Use gpTxQueue b1 = use CSMA/CA b2 = use MAC ACK b3-b4 – Frame type for Tx (see values in Table G.10) b5 – b7 - reserved
ApplicationID	8 bit enumeration	0x00, 0x02	ApplicationID of the Green Power Device to which the frame will be sent. ApplicationID 0x00 indicates the usage of the 32 bit SrcID and ApplicationID 0x02 indicates the usage of the GPD IEEE address.
SrcID	Unsigned 32-bit Integer	0x00000000 – 0xffffffff	The identifier of the GPD entity to which the ASDU will be sent if ApplicationID = 0x00.
GPD IEEE address	IEEE Address	Any Valid	The identifier of the GPD entity to which the ASDU will be sent if ApplicationID = 0x02.
Green Power CommandID	Integer	0x00 – 0xff	The identifier of the command from the Green Power specification [B4], section A.4, which defines the application semantics of the ASDU.

Name	Type	Valid Range	Description
ASDULength	Integer	0x00 – (*aMax-MACFrameSiz e* - 9)	The number of octets in the ASDU to be transmitted.
ASDU	Set of octets	-	The set of octets forming the ASDU to be transmit-ted.
ASDUHandle	Integer	0x00 – 0xff	An integer handle associated with the ASDU to be transmitted.
gpTxQueueEntry-Lifetime	Unsigned 16-bit integer	0x0000 – 0xffff	The lifetime of this packet in the gpTxQueue, in milliseconds. For GPD Commissioning Reply command, initialize to Commissioning Window. 0x0000 indicates immediate transmission. 0xFFFF indicates infinity.

G.2.4.2 When Generated

This primitive is generated by the local application entity (GPEP) when it wishes to address a Green Power Device Frame to the GPD identified by the GPD SrcID/GPD IEEE address parameter.

G.2.4.3 Effect on Receipt

On receipt of the GP-DATA.request primitive by the Inter-PAN APS, the Inter-PAN APS will construct a Green Power Device Frame formatted as specified in section G.3.2.2, with Protocol Version sub-field and Frame Type sub-field of the Network Frame control field set as specified in section G.3.2, containing the given ASDU and protect it, as specified in section G.5. The stub queues the GPDF in the gpTxQueue, as defined in section G.4.3, and later transmits the GPDF, as specified in section G.4.4, using the MCPS-DATA.request primitive of the MAC sub-layer.

G.2.5 The INTRP-DATA.confirm Primitive

The INTRP-DATA.confirm primitive allows the Inter-PAN APS to inform the application entity about the status of a data request.

G.2.5.1 Semantics of the Service Primitive

The primitive interface is as follows:

```
INTRP-DATA.confirm          {
                            ASDUHandle
                            Status
                            }
```

Table G.3 defines the parameters of the INTRP-DATA.confirm primitive.

Table G.3 Parameters of the INTRP-DATA.confirm

Name	Type	Valid Range	Description
ASDUHandle	Integer	0x00 – 0xFF	An integer handle associated with the transmitted data frame.
Status	Enumeration	Any status value returned by the MAC.	The status of the ASDU transmission corresponding to ASDUHandle returned by the MAC.

G.2.5.2 When Generated

This primitive is generated by the Inter-PAN APS on a ZigBee device and passed to the application in response to the receipt of a MCPS-DATA.confirm primitive that is a confirmation of a previous MCPS-DATA.request issued by the Inter-PAN APS.

G.2.5.3 Effect on Receipt

As a result of the receipt of this primitive, the application is informed of the results of an attempt to send a frame via the Inter-PAN APS.

G.2.6 The GP-DATA.confirm Primitive

The GP-DATA.confirm primitive allows the Inter-PAN APS to inform the application entity about the status of a Green Power data request.

G.2.6.1 Semantics of the Service Primitive

The primitive interface is as follows:

```
GP-DATA.confirm                 {
                                ASDUHandle
                                Status
                                }
```

Table G.4 defines the parameters of the GP-DATA.confirm primitive.

Table G.4 Parameters of the GP-DATA.confirm

Name	Type	Valid Range	Description
ASDUHandle	Integer	0x00 – 0xFF	An integer handle associated with the transmitted data frame.
Status	Enumeration	Any status value returned by the MAC.	The status of the ASDU transmission corresponding to ASDUHandle as returned by the MAC. In addition to the values returned by the MAC layer, it can have the following values: TX_QUEUE_FULL ENTRY_REPLACED ENTRY_ADDED ENTRY_EXPIRED ENTRY_REMOVED FRAME_SENDING_FINALIZED

G.2.6.2 When Generated

This primitive is generated by the Inter-PAN APS on a ZigBee device and passed to the application (GPEP) in response to the receipt of a GP-DATA.request and MCPS-DATA.confirm primitive that is a confirmation of a previous MCPS-DATA.request issued by the Inter-PAN APS. The reasons for the various Status codes are described in section G.4.3.

G.2.6.3 Effect on Receipt

As a result of the receipt of this primitive, the application is informed of the results of an attempt to send a Green Power Device Frame via the Inter-PAN APS.

G.2.7 The GP-SEC.request Primitive

G.2.7.1 Semantics of the GP-SEC.request primitive

The primitive interface is as follows:


```
GP-SEC.request    {
                  ApplicationID
                  SrcID
                  GPD IEEE Address
                  GPDFSecurityLevel
                  GPDFKeyType
                  GPDSecurityFrameCounter
                  Stub Handle
                  }
```

Table G.5 defines the parameters of the GP-SEC.request primitive.

Table G.5 Parameters of the GP-SEC.request

Name	Type	Valid Range	Description
ApplicationID	8 bit enumeration	0x00, 0x02	ApplicationID of the Green Power Device from which the Green Power Device Frame was received. ApplicationID 0x00 indicates the usage of the 32 bit SrcID and ApplicationID 0x02 indicates the usage of the GPD IEEE address.
SrcID	Unsigned 32-bit integer	0x00000001 – 0xfffffffe	The identifier of the GPD entity from which the Green Power Device Frame was received if ApplicationID = 0x00.
GPD IEEE Address	64-bit address	Any valid	The identifier of the GPD entity from which the Green Power Device Frame was received if ApplicationID = 0x02.
GPDFSecurityLevel	8-bit enumeration	0x00 – 0x03	The security level of the received frame.

Name	Type	Valid Range	Description
GPDFKeyType	8-bit enumeration	0x00 – 0x07	The security key type of the received frame.
GPD security frame counter	Unsigned 8-bit or 32-bit Integer	As specified by the GPDFSecurityLevel parameter	The security frame counter value corresponding to the received frame.
Stub Handle	Unsigned 8-bit Integer	0x00 – 0xff	The handle used between the Inter-PAN APS and the higher layers, to match the request with the response.

G.2.7.2 When Generated

This primitive is generated by the Inter-PAN APS and passed up on reception of a Green Power Device Frame.

G.2.7.3 Effect on Receipt

Upon receipt of this primitive the device is informed of reception of a Green Power Device Frame. The device then can retrieve security material for handling the frame.

G.2.8 The GP-SEC.response Primitive

G.2.8.1 Semantics of the GP-SEC.response primitive

The primitive interface is as follows:

GP-SEC.response	{
	Status
	Stub Handle
	ApplicationID
	SrcID
	GPD IEEE Address GPDFSecurityLevel

GPDFKeyType
GPDKey
GPDSecurityFrameCounter
SecurityWindow
}

Table G.6 defines the parameters of the GP-SEC.response primitive.

Table G.6 Parameters of the GP-SEC.response Primitive

Name	Type	Valid Range	Description
Status	8-bit enumeration	Any valid	The status code as returned by the end point. The following values are supported: MATCH DROP_FRAME PASS_UNPROCESSED
Stub Handle	Unsigned 8-bit Integer	0x00 – 0xff	The handle used between the Inter-PAN APS and the higher layers, to match the request with the response.
ApplicationID	8 bit enumeration	0x00, 0x02	ApplicationID of the Green Power Device from which the Green Power Device Frame was received. ApplicationID 0x00 indicates the usage of the 32 bit SrcID and ApplicationID 0x02 indicates the usage of the GPD IEEE Address.
SrcID	Unsigned 32-bit integer	0x00000001 – 0xfffffffe	The identifier of the GPD entity from which the Green Power Device Frame was received if ApplicationID = 0x00.

Name	Type	Valid Range	Description
GPD IEEE Address	64-bit address	Any valid	The identifier of the GPD entity from which the Green Power Device Frame was received if ApplicationID = 0x02.
GPDFSecurityLevel	8-bit enumeration	0x00 – 0x03	The security level to be used for security processing.
GPDFKeyType	8-bit enumeration	0x00 – 0x07	The security key type to be used for security processing.
GPD Key	Security Key	Any valid	The security key to be used for GPDF security processing.
GPD security frame counter	Unsigned 32-bit Integer	Any valid	The security frame counter value to be used for security processing.
SecurityWindow	Unsigned 8-bit Integer	0x00 – 0xff	The SecurityWindow value to be used for security processing of this incoming frame.

13385

G.2.8.2 When Generated

This primitive is generated by the Green Power endpoint (GPEP) and passed to the Inter-PAN APS on reception of a GP-SEC.request. The GPEP responds with appropriate status, based on the GPEP client/server functionality, the operational/commissioning mode the GPEP is in and the content of Proxy/Sink Table.

G.2.8.3 Effect on Receipt

Upon receipt of this primitive the Inter-PAN APS checks the value of the *Status* field. If the Status is MATCH, the Inter-PAN APS triggers security processing of the GPDF, with the supplied parameters. If the *Status* is DROP_FRAME, it silently drops the frame. If the *Status* is PASS_UNPROCESSED, it generates GP-DATA.indication with the unprocessed fields GPD CommandID, GPD Command Payload and MIC copied from the received GPDF, and passes it to the Green Power application endpoint.

G.2.9 The INTRP-DATA.indication Primitive

The INTRP-DATA.indication primitive allows the Inter-PAN APS to inform the next higher layer that it has received a frame that was transmitted via the Inter-PAN APS on another device.

G.2.9.1 Semantics of the Service Primitive

The primitive interface is as follows:

INTRP-DATA.indication	{
	SrcAddrMode
	SrcPANId
	SrcAddress
	DstAddrMode
	DstPANId
	DstAddress
	ProfileId
	ClusterId
	ASDULength
	ASDU
	LinkQuality
	}

Table G.7 defines the parameters of the INTRP-DATA.indication primitive.

Table G.7 Parameters of the INTRP-DATA.indication Primitive

Name	Type	Valid Range	Description

Name	Type	Valid Range	Description
SrcAddrMode	Integer	0x00- 0x03	The source addressing mode for the MPDU to be sent. The following values are allowed: 0x00 – no address (SrcPANId and SrcAddress omitted) 0x01 = reserved 0x02 = 16 bit short address 0x03 = 64 bit extended address
SrcPANId	16-bit PAN Id	0x0000 – 0xffff	The 16-bit PAN identifier of the entity from which the ASDU is being transferred.
SrcAddress	64-bit address	As specified by the SrcAddrMode parameter	The device address of the entity from which the ASDU is being transferred.
DstAddrMode	Integer	0x00 – 0x03	The addressing mode for the destination address used in this primitive. This parameter can take one of the values from the following list: 0x00 = no address (DstPANId and DstAddr omitted) 0x01 = 16-bit group address 0x02 = 16-bit NWK address, usually the broadcast address 0xffff 0x03 = 64-bit extended address
DstPANID	16-bit PAN Id	0x0000 – 0xffff	The 16-bit PAN identifier of the entity or entities to which the ASDU is being transferred or the broadcast PAN ID 0xffff.
DstAddress	16-bit or 64-bit address	As specified by the DstAddrMode parameter	The address of the entity or entities to which the ASDU is being transferred.

Name	Type	Valid Range	Description
ProfileId	Integer	0x0000 – 0xffff	The identifier of the application profile for which this frame is intended.
ClusterId	Integer	0x0000 – 0xffff	The identifier of the cluster, within the profile specified by the ProfileId parameter, which defines the application semantics of the ASDU.
ASDULength	Integer	0x00 – (*aMaxMACFrameSize* - 9)	The number of octets in the ASDU to be transmitted.
ASDU	Set of octets	-	The set of octets forming the ASDU to be transmitted.
LinkQuality	Integer	0x00 – 0xff	The link quality observed during the reception of the ASDU.

G.2.9.2 When Generated

This primitive is generated and passed to the application in the event of the receipt, by the Inter-PAN APS, of a MCPS-DATA.indication primitive from the MAC sub-layer, containing a frame that was generated by the Inter-PAN APS of a peer ZigBee device, and that was intended for the receiving device.

G.2.9.3 Effect on Receipt

Upon receipt of this primitive the application is informed of the receipt of an application frame transmitted, via the Inter-PAN APS, by a peer device and intended for the receiving device. The values of the INTRP-DATA.indication shall be copied into the matching field names of the APSME-DATA.indication. Additionally these fields shall be set as follows:

1. The DstAddrMode shall be set to 0x04.
2. The DstAddress shall be set to the DstAddress of the INTRP-DATA.indication primitive.
3. The SrcAddrMode shall be set to 0x04.
4. The SrcAddress shall be set to the SrcAddress of the INTRP-DATA.indication primitive.
5. The SecurityStatus field enumeration shall be set to UNSECURED.
6. The Inter-PAN field shall be set to TRUE.

G.2.10 The GP-DATA.indication Primitive

The GP-DATA.indication primitive allows the Inter-PAN APS to inform the next higher layer that it has received a Green Power Device Frame.

G.2.10.1 Semantics of the Service Primitive

The primitive interface is as follows:

GP-DATA.indication	{
	Status
	LinkQuality
	SeqNumber
	SrcAddrMode
	SrcPANId
	SrcAddress
	ApplicationID
	GPDFSecurityLevel
	GPDFKeyType
	AutoCommissioning
	RxAfterTx
	SrcID
	GPD Security Frame Counter
	CommandID
	ASDULength
	ASDU
	MIC
	}

Table G.8 defines the parameters of the GP-DATA.indication primitive.

13464 **Table G.8 Parameters of the GP-DATA.indication Primitive**

Name	Type	Valid Range	Description
Status	8-bit enumeration	Any Valid	Status Code returned by Green Power. It can have the following values: SECURITY_SUCCESS NO_SECURITY COUNTER_FAILURE AUTH_FAILURE UNPROCESSED
LinkQuality	Integer	0x00 – 0xff	The link quality observed during the reception of the ASDU.
SeqNumber	Unsigned 8-bit integer	0x00 – 0xff	The sequence number from the MAC header of the received frame.
SrcAddrMode	Integer	0x00- 0x03	The source addressing mode for the MPDU to be sent. The following values are allowed: 0x00 – no address (SrcPANId and SrcAddress omitted) 0x01 = reserved 0x02 = 16 bit short address 0x03 = 64 bit extended address
SrcPANId	16-bit PAN Id	0x0000 – 0xffff	The 16-bit PAN identifier of the entity from which the ASDU is being transferred.
SrcAddress	64-bit address	As specified by the SrcAddrMode parameter	The device address of the entity from which the ASDU is being transferred.

Name	Type	Valid Range	Description
ApplicationID	8-bit enumeration	0x00, 0x02	The ApplicationID, corresponding to the received frame. ApplicationID 0x00 indicates the use of a SrcID; ApplicationID 0x02 indicates the usage of the GDP IEEE address.
GPDFSecurityLevel	8-bit enumeration	0x00- 0x03	The security level of the received frame.
GPDFKeyType	8-bit enumeration	0x00 – 0x07	The security key type, which was successfully used for security processing the received frame.
Auto-Commissioning	Boolean	TRUE/FALSE	The Auto-commissioning sub-field, copied from the received frame.
RxAfterTx	Boolean	TRUE/FALSE	The RxAfterTx sub-field, copied from the received frame.
SrcID	Unsigned 32-bit Integer	0x00000000 – 0xffffffff	The identifier of the GPD entity from which the frame was received if ApplicationID = 0x00.

Name	Type	Valid Range	Description
GPD security frame counter	Unsigned 32-bit Integer	As specified by the GPDFSecurityLevel parameter	The security frame counter value used on transmission by the GPD entity from which the frame was received.
GPD Command ID	Unsigned 8-bit Integer	0x00 – 0xff	The identifier of the command, within the Green Power specification [B4] section A.4 which defines the application semantics of the frame.
ASDULength	Integer	0x00 – (*aMaxPHYPacketSize*)	The number of octets in the received ASDU.
ASDU	Set of octets	-	The set of octets in the received ASDU.
MIC	Unsigned 16-bit or 32-bit Integer	As specified by the GPDFSecurityLevel parameter	The set of octets forming the MIC for the received frame.

G.2.10.2 When Generated

This primitive is generated and passed to the application in the event of the receipt, by the Inter-PAN APS, of a MCPS-DATA.indication primitive from the MAC sub-layer, containing a Green Power Device Frame.

G.2.10.3 Effect on Receipt

Upon receipt of this primitive the Green Power endpoint (GPEP) is informed of the receipt of a Green Power Device Frame.

G.2.11 Qualifying and Testing of Inter-PAN Messages

Support for Inter-PAN messages and Green Power is optional. If a device claims Inter-PAN communication support then certification and application level testing shall ensure both the sending and receiving devices correctly react and understand the INTRP-DATA.request and INTRP-DATA.indication primitives. Green Power certification and application level testing shall also ensure the GP-DATA.request, GP-DATA.indication, GP-SEC.request, and GP-SEC.response primitives are supported as mandated by the Green Power Specification [B4].

G.3 Frame Formats

The overall view of a ZigBee frame is as shown in Figure G.2

Figure G.2 - ZigBee Frame Format Overview

802.15.4 MAC Header	ZigBee NWK Header	ZigBee APS Header	ZigBee Payload

Briefly, the frame contains the familiar headers controlling the operation of the MAC sub-layer, the NWK layer and the APS. Following these, there is a payload, formatted as specified in [B1].

Since most of the information contained in the NWK header is not relevant for Inter-PAN transmission, the Inter-PAN frame, shown in Figure G.3, contains only a stub of the NWK header. A Inter-PAN APS header is also used and is described in section G.3.3.

Figure G.3 Inter-PAN Frame Format

Octets: 2	**1**	**variable**	**2**
Frame Control	Sequence Number	Addressing Fields	NWK Frame Control
802.15.4 MAC Header			NWK Header

For Green Power Device Frames there is a different set of MAC and NWK headers as shown in Figure G.4.

Figure G.4 Green Power Device Frame Format

Octets: 2	**1**	**4/10/12/ Variable**	**1**	**0/1**	**0/4**	**0/4**	**Variable**	**0/2/4**
Frame Control	Sequence Number	Addressing Fields	NWK Frame Control	Extended NWK Frame Control	GPD SrcID	Security Frame Counter	GPDF Application Payload	MIC
802.15.4 MAC Header			GPDF NWK Header				GPDF Application Payload	GPDF NWK Trailer

G.3.1 MAC Header

The 802.15.4 MAC header has several options depending on how the frame is being used. The MAC header fields are shown in Table G.9 with notes on their use.

Table G.9 MAC Header Fields for Inter-PAN APS Frames

Field Name	**Octets**	**Usage**
Frame Control	2	Varies by Inter-PAN APS frame
Sequence Number	1	Normally used as MAC sequence number, increasing for each frame sent. Green Power usage discussed in section G.3.2.2.1.
Destination PAN ID	0/2	May be set as the PANID of the destination or 0xffff.
Destination Address	2/8	Normally either broadcast short address or a 64 bit long address of the destination. Green Power usage discussed in section G.3.1.2
Source PAN ID	0/2	Used in Inter-PAN messaging but not in Green Power Device Frames
Source Address	2/8	Normally set to the 64 bit address of the source device. Green Power usage discussed in section G.3.1.2

The MAC header usage varies by application using the Inter-PAN messaging.

G.3.1.1 MAC Header usage for Inter-PAN messaging

Because Inter-PAN messaging is used for devices not on the ZigBee network, short addressing is not normally used unless it is the broadcast short address such that any device within range can respond. Otherwise the 64 bit long addresses are used for source and destination addressing. Source and Destination PANID's may be used or may be omitted.

G.3.1.2 MAC Header usage for Green Power Device Frames

The Green Power Device Frame originating from the GPD can be sent with MAC Dest PANID and MAC Dest Address set to 0xffff.

If the IEEE address of the GPD is used for unique identification, the Green Power Device Frame shall include the Extended NWK Frame Control field and its ApplicationID sub-field shall be set to 0b010. Then, for the frame transmitted by the GPD, the GPD's IEEE address shall be transmitted in the MAC Src Address field, and the Intra-PAN sub-field and the Source Addressing Mode sub-field of the MAC Frame Control field shall be set accordingly. For the frames transmitted to the GPD, the GPD's IEEE address shall be transmitted in the MAC Dest Address field, and the Intra-PAN sub-field and the Destination Addressing Mode sub-field of the MAC Frame Control field shall be set accordingly; see also section G.3.2.

G.3.2 Network Header

G.3.2.1 Stub NWK Header for Inter-PAN Messages

The stub NWK Header for Inter-PAN messages is shown below in Figure G.5.

Figure G.5 Stub NWK Header for Inter-PAN messages

Octets:2
NWK Frame Control

The NWK header Frame control field for the Inter-PAN messages is formatted exactly as the NWK header used by other ZigBee frames, see section 3.3.1.1 of the current specification.

For Inter-PAN messages, the frame type 0b11 is used with the protocol version of the ZigBee stack. All other sub-fields shall have a value of 0.

G.3.2.2 Stub NWK Header for Green Power Device Frames

G.3.2.2.1 Stub NWK for Green Power Device Frames

The format of the stub NWK Header for GPDF is formatted as shown in Figure G.6.

Figure G.6 NWK Header Frame Control for Green Power Device Frames

Octets: 1	0/1	0/4	0/4	Variable	0/2/4
NWK frame control	Extended NWK Frame Control	GPD SrcID	Security frame counter	GPDF appli-cation payload	MIC
GPDF NWK Header				GPDF appli-cation payload	GPDF NWK Trailer

The NWK Frame Control of the stub NWK Header for GPDF is formatted as shown in **Error! Reference source not found.**.

Figure G.7 NWK Header Frame Control for Green Power Device Frames

NWK Frame Control				Extended NWK Frame Control	
Bits: 0-1	**2-5**	**6**	**7**	**Bits 0-2**	**3-7**
Frame type	Protocol version	Auto-Commissioning	NWK Frame Control Extension	Application ID	Defined for specific ApplicationID

The sub-fields of the NWK frame control field are as follows:

For the Green Power Device Frames, the ZigBee Protocol Version sub-field shall carry the value of 0x3.

The frame type sub-field as used in combination with the ZigBee Protocol Version = 0x3, can take the values specified in Table G.10.

Table G.10 Values for Frame Type for GPDF

Value	Description
0b00	Data Frame
0b01	Maintenance Frame
0b10	Reserved
0b11	Reserved

If the Frame Type 0b01 (Maintenance frame) is used then the Green Power SrcID and the security fields (security frame counter and MIC) shall not be present. Green Power Devices should omit the extended NWK frame control and it may also be omitted when sending to Green Power Devices. The NWK Frame Control extension sub-filed shall be set accordingly.

If the Frame Type 0b00 (Data frame) is used the Green Power frame format is as follows:

The Auto Commissioning sub-field indicates if the device implements the GPD Commissioning command (see reference [B4], section A.4). If set to 0b1 the device does not implement the GPD Commissioning command. If set to 0b0 the device does implement the GPD Commissioning command.

The NWK Frame Control Extension, if set to 0b1, indicates the Extended NWK Frame Control field is present. The Extended NWK Frame Control extension shall be present if the ApplicationID is different than 0b000.

The ApplicationID allows for re-defining the frame format. The current specification defines the frame format for ApplicationID 0b000 and 0b010 (Green Power). Default value to be used on reception, if the Extended NWK Frame Control field is not present is 0b000.

For ApplicationID 0b000 and 0b010 and 0b001, the bits 3-7 are defined in Figure G.8. For ApplicationID 0b000 the Extended NWK Frame Control field shall be present if the frame is protected, if RxAfterTx is set, or if the frame is sent to the Green Power Device.

Figure G.8 Format of Extended NWK Frame Control field for GPDF with Application ID 0b000 and 0b010

Bits 3-4	5	6	7
Security Level	Security Key	RxAfterTx	Direction

The SecurityLevel sub-field indicates if the frame is protected.

If ApplicationID is set to 0b000 and 0b010, the Security Level sub-field can have values as defined in Table G.10. Default value to be used on reception, if the Extended NWK Frame Control field is not present, is 0b00.

If the SecurityLevel is set to 0b00, the SecurityKey sub-field is ignored on reception, and the fields Security frame counter and MIC are not present. The MAC sequence number field carries the random or the incremental sequence number, according to the capabilities of this GPD.

If the SecurityLevel is set to 0b01, the Security Frame counter field is not present, the MAC sequence number field carries the 1LSB of the frame counter, and the MIC field is present, has the length of 2B, and carries the 2LSB of the Message Integrity Code (see section G.5.4 of the current document).

If the SecurityLevel is set to 0b10 or 0b11, the Security Frame counter field is present, has the length of 4B, and carries the full 4B security frame counter, the MIC field is present, has the length of 4B, and carries the full 4B Message Integrity Code (see section G.5.4 of the current document). The MAC sequence number field carries the random or the incremental sequence number, according to the capabilities of this GPD; it shall not be used for security, but only for duplicate filtering at MAC level.

Table G.11 Values of gpSecurityLevel

Value	Description
0b00	No security
0b01	1LSB of frame counter and short (2B) MIC only
0b10	Full (4B) frame counter and full (4B) MIC only
0b11	Encryption & full (4B) frame counter and full (4B) MIC

The SecurityKey sub-field indicates the type of the key used for frame protection by this GPD. The Security Key sub-field, if set to 0b1, indicates an individual key (KeyType 0b100 or 0b111). If set to 0b0, it indicates a shared key (KeyType 0b011, 0b010 or 0b001) or no key.

The RxAfterTx sub-field is a Boolean flag. If the value of this sub-field is 0b1, then it indicates that the GPD will enter the receive mode after gpdRxOffset, for a device-specific duration, but not shorter than gpdMinRxWindow. If the value of this sub-field is 0b0, then the GPD will not enter the receive mode after sending this particular GPDF frame. Default value to be used on reception, if the Extended NWK Frame Control field is not present, is 0b0.

The Direction sub-field shall be set to 0b0, if the GPDF is transmitted by the GPD, and to 0b1, if the GPDF is transmitted by GPP. Default value to be used on reception, if the Extended NWK Frame Control field is not present, is 0b0.

G.3.2.2.2 Remaining Fields of the Stub NWK Header for GPDF

The GPDSrcID field is present if the FrameType sub-field is set to 0b00 and the ApplicationID sub-field of the Extended NWK Frame Control field is set to 0b000 (or not present). It is also present if the FrameType sub-field is set to 0b01, the NWK Frame control Extension sub-field is set to 0b1, and the ApplicationID sub-field of the Extended NWK Frame Control field is set to 0b000. The GPDSrcID field carries the unique identifier of the GPD, to/by which this GPDF is sent. The value of 0x00000000 indicates unspecified. The value of 0xffffffff indicates all. The values 0xfffffff9 – 0xfffffffe are reserved. The GPDSrcID field is not present if the FrameType sub-field is set to 0b01 and the Extended NWK Frame control sub-field is set to 0b0. Unique identification of the GPD by an address is not required then. The GPDSrcID field is not present if the ApplicationID sub-field of the Extended NWK Frame Control field is set to 0b010. The GPD is then identified by its IEEE address, which is then carried in the corresponding MAC address field, source or destination for the GPDF sent by or to the GPD, respectively.

The presence and length of the Security frame counter field is dependent on the value of ApplicationID and SecurityLevels sub-field, as described above.

The MIC field carries the Message Integrity Code for this message, calculated as specified in section G.5.4 of the current specification. Its presence and length is dependent on the value of ApplicationID and SecurityLevel sub-fields, as described above.

The application payload of the GPDF is defined in [B4], section A.1.4.1.6.

G.3.3 Inter-PAN APS Header

The format of the Inter-PAN APS header is shown in Figure G.9. This is used in normal Inter-PAN messages and Touchlink messages but not in Green Power Device Frames.

Figure G.9 Inter-PAN APS Header Format

Octets: 1	**0/2**	**2**	**2**
APS frame control	Group address	Cluster identifier	Profile identifier
	Addressing fields		

The Inter-PAN APS header contains only 4 fields totaling a maximum of 7 octets in length.

The APS frame control field shall be 1 octet in length and is identical in format to the frame control field of the general APDU frame in [B3] (see Figure G.10).

Figure G.10 Format of the APS Frame Control Field for Inter-PAN Messages

Bits: 0-1	**2-3**	**4**	**5**	**6**	**7**
Frame type	Delivery Mode	Reserved	Security	ACK request	Extended Header Present

The fields of the frame control field have the following values:

- The frame type sub-field shall have a value of 0b11, which is the Inter-PAN APS frame type.
- The delivery mode sub-field may have a value of 0b00, indicating unicast, 0b10, indicating broadcast or 0b11 indicating group addressing.
- Security is never enabled for Inter-PAN transmissions. This sub-field shall be a value of 0.
- The ACK request sub-field shall have a value of 0, indicating no ACK request. No APS ACKs are to be used with Inter-PAN transmissions.
- The extended header present sub-field shall always have a value of 0, indicating no extended header.

The optional group address shall be present if and only if the delivery mode field has a value of 0x0b11. If present it shall contain the 16-bit identifier of the group to which the frame is addressed.

The cluster identifier field is 2 octets in length and specifies the identifier of the cluster to which the frame relates and which shall be made available for filtering and interpretation of messages at each device that takes delivery of the frame. For touchlink this has a value of 0x1000.

The profile identifier is two octets in length and specifies the ZigBee profile identifier for which the frame is intended and shall be used during the filtering of messages at each device that takes delivery of the frame. For touchlink this has the value of 0xc05e.

G.4 Frame Processing

Assuming the INTRP-SAP described above, frames transmitted using the Inter-PAN APS are processed as described here.

G.4.1 Inter-PAN Transmission (non Green Power Device Frames)

On receipt of the INTRP-DATA.request primitive, the Inter-PAN APS shall construct a Inter-PAN APS frame. The header of the Inter-PAN APS frame shall contain a NWK and an APS frame control field as described in section G.3, a cluster identifier field equal to the value of the ClusterId parameter of the INTRP-DATA.request and a profile identifier field equal to the value of the ProfileId parameter. If the DstAddrMode parameter of the INTRP-DATA.request has a value of 0x01, indicating group addressing, then the APS header shall also contain a group address field with a value corresponding to the value of the DstAddress parameter. The payload of the Inter-PAN APS frame shall contain the data payload to be transmitted.

The Inter-PAN APS frame will then be transmitted using the MCPS-DATA.request primitive of the MAC sub-layer with key primitive parameters set as follows:

- The value of the SrcAddrMode parameter of the MCPS-DATA.request shall always be set to a value of three, indicating the use of the 64-bit extended address.
- The SrcPANId parameter shall be equal to the value of the macPANID attribute of the MAC PIB.
- The SrcAddr parameter shall always be equal to the value of the MAC sub- layer constant aExtendedAddress.
- If the DstAddrMode parameter of the INTRP-DATA.request primitive has a value of 0x01, then the DstADdrMode parameter of the MCPS-DATA.request shall have a value of 0x02. Otherwise, the DstAddrMode parameter of the MCPS-DATA.request shall reflect the value of the DstAddrMode parameter of the INTRP-DATA.request.
- The DstPANId parameter shall have the value given by the DstPANID parameter of the INTRP-DATA.request primitive.
- If the DstAddrMode parameter of the INTRP-DATA.request has a value of 0x01, indicating group addressing, then the value of the DstAddr parameter of the MCPS-DATA.request shall be the broadcast address 0xffff. Otherwise, value of the DstAddr parameter shall reflect the value of the DstAddress parameter of the INTRP-DATA.request primitive.
- The MsduLength parameter shall be the length, in octets, of the Inter-PAN APS frame.
- The Msdu parameter shall be the Inter-PAN APS frame itself.
- If the transmission is a unicast, then the value of the TxOptions parameter shall be 0x01, indicating a request for acknowledgement. Otherwise, the TxOptions parameter shall have a value of 0x00, indicating no options.

On receipt of the MCPS-DATA.confirm primitive from the MAC sub-layer, the Inter-PAN APS will invoke the INTRP-DATA.confirm primitive with a status reflecting the status returned by the MAC.

G.4.2 Inter-PAN Reception (non Green Power Device Frames)

On receipt of the MCPS-DATA.indication primitive from the MAC sub-layer, the receiving entity - in case of a ZigBee device this is normally the NWK layer - shall determine whether the frame should be passed to the Inter-PAN APS or processed as specified in [B3]. For a frame that is to be processed by the Inter-PAN APS, the non- varying sub-fields of the NWK frame control field must be set exactly as described in section G.3.2.1and the APS frame control field must be set exactly as described in section G.3.3. Any variation from this format shall trigger the message to be dropped and no further processing shall be done.

If the delivery mode sub-field of the APS frame control field of the Inter-PAN APS header has a value of 0b11, indicating group addressing, then, if the device implements group addressing, the value of the group address field shall be checked against the NWK layer group table, and, if the received value is not present in the table, the frame shall be discarded with no further processing or action.

On receipt of a frame for processing, the Inter-PAN APS shall generate an INTRP- DATA.indication with parameter values as follows:

- The value of the SrcAddrMode parameter of the INTRP-DATA.indication shall always be set to a value of three, indicating the use of the 64-bit extended address
- The value of the SrcPANId parameter shall reflect that of the SrcPANId parameter of the MCPS-DATA.indication.
- The SrcAddress parameter of the INTRP-DATA.indication shall always reflect the value of a 64-bit extended address.
- Values for the DstAddrMode parameter shall be one of:
  - 0x03, if the DstAddrMode parameter of the INTRP-DATA.indication has a value of 0x03.
  - 0x02, if the DstAddrMode parameter of the INTRP-DATA.indication has a value of 0x02
- The value of the DstPANId parameter of the INTRP-DATA.indication shall reflect the value of the DstPANId parameter of the MCPS-DATA.indication.
- If the DstAddrMode parameter of the INTRP-DATA.indication has a value of 0x01, indicating group addressing then the DstAddress parameter of the INTRP-DATA.indication shall reflect the value of the group address field of the Inter-PAN APS header. Otherwise, the value of the DstAddress parameter of the INTRP-DATA.indication shall reflect the value of the DstAddr parameter of the MCPS-DATA.indication.
- The value of the ProfileId parameter shall be the same as the value of the profile identifier field of the Inter-PAN APS header.
- The value of the ClusterId parameter shall be the same as the value of the cluster identifier field of the Inter-PAN APS header.
- The ASDULength field shall contain the number of octets in the Inter-PAN APS frame payload.
- The ASDU shall be the Inter-PAN APS payload itself.
- The value of the LinkQuality parameter shall reflect the value of the mpduLinkQuality parameter of the MCPS-DATA.indication.

G.4.3 Green Power Device Frame Transmission

On receipt of the GP-DATA.request primitive, the Inter-PAN APS shall check the gpTxQueue. If the gpTxQueue already has an entry for the GPD ID (i.e. GPD SrcID/GPD IEEE address) in the GP-DATA.request, the previous GPDF is overwritten and GP-DATA.confirm with the Status ENTRY_REPLACED is provided to the GPEP. If the gpTxQueue has no previous entry for this GPD SrcID/GPD IEEE address and it has empty entries, the GPDF is added to the gpTxQueue and GP-DATA.confirm with the Status ENTRY_ADDED is provided to the GPEP. If the gpTxQueue has no previous entry for this GPD SrcID/GPD IEEE address and it is full, the Inter-PAN APS returns GP-DATA.confirm with the Status set to QUEUE_FULL.

G.4.3.1 gpTxQueue

In gpTxQueue, GPDF are stored for transmission to GPD.

In its gpTxQueue, each GP infrastructure device shall have a maximum of only one pending GPDF frame per GPD ID. Each entry in the gpTxQueue shall have a gpTxQueueEntryLifetime parameter associated, initiated by the value in the GP-DATA.request. When this timeout elapses, the GP-DATA.confirm with the Status ENTRY_EXPIRED is returned to the GPEP, the entry is cleared and can be used for any GPDF for any GPD ID. The gpTxQueue shall have a minimum length of 5 entries.

G.4.3.2 gpTxOffset

The gpTxOffset is the time after which the Inter-PAN APS shall send a GPDF in response to a GPDF with RxAfterTx sub-field set, if any present in the gpTxQueue for this GPD ID. It is measured from the start of the reception of the first GPDF in a given GPFS.

The gpTxOffset has value identical to the gpdRxOffset (see sec. A.1.6.3.1).

G.4.3.3 gpTxDuration

The gpTxDuration is the maximum allowed transmission time for the Inter-PAN APS after gpTxOffset. Thus, depending on the GPDF length, the Inter-PAN APS may send the GPDF more than once, to increase the reliability of communication. It is measured from the start of the transmission of the first GPDF in a given GPFS.

The gpTxDuration has the value of 10ms.

G.4.4 Green Power Device Frame Reception

On receipt of a GPDF, the Inter-PAN APS shall filter out (silently drop) frames with ApplicationID value other than 0b000 and 0b010 frames with Direction sub-field of the Extended NWK Frame Control field set to 0b1, and duplicate frames. For this purpose, the MCPS-DATA.indication shall also include the MAC sequence number parameter.

Frames with ApplicationID 0b000 and 0b010 shall be further processed, as follows.

The Inter-PAN APS shall check the *SecurityLevel*. If the *SecurityLevel* is not supported, the stub shall silently drop the frame. If *SecurityLevel* is supported and has the value of 0b00-0b10, and GPD CommandID has the value from the range 0xf0-0xff, the GPDF is silently dropped. If *SecurityLevel* is supported, the stub then generates GP-SEC.request and waits for GP-SEC.response.

On receipt of GP-SEC.response with *Status* DROP_FRAME, the stub drops the frame. On receipt of GP-SEC.response with Status PASS_UNPROCESSED, the stub generates GP- DATA.indication for the unprocessed frame. On receipt of GP-SEC.response with *Status* MATCH, the stub security-processes the received GPDF, as described in section G.5.

If security processing fails, the stub indicates that with GP-DATA.indication carrying the corresponding *Status* value and stops any further processing of this frame.

If security processing is successful, and the *SecurityLevel* was 0b11, the stub checks the plaintext value of the GPD *CommandID*. If it has the value from the range 0xf0-0xff, the GPDF is silently dropped.

If security processing was successful, and the GPD *CommandID* is not from the 0xf0 – 0xff range, the stub checks if the *RxAfterTx* sub-field of the *Extended NWK Frame Control* field of the received GPDF was set to 0b1. If yes, it searches the gpTxQueue for an entry for this GPD ID. If a suitable GPDF is found, the stub triggers security processing of the to-be-sent GPDF with the same security input parameters as for the received GPDF. If the Data Frame Type is used, the NWK Frame Control Extension sub-field shall be set to 0b1, the Extended NWK Frame Control field shall be present, and the *RxAfterTx* sub-field shall be set to 0b0 and the Direction sub-field shall be set to 0b1.

Then, the Inter-PAN APS constructs the GPDF with the ApplicationID sub-field of the Extended NWK Frame Control field set to 0b000 or 0b010, as supplied in the GP-DATA.request primitive, and the remaining fields as supplied by the GP-DATA.request primitive.

The Inter-PAN APS schedules GPDF transmission to commence after gpTxOffset, by sending MCPS-DATA.request, with UseCSMA parameter set to FALSE and Use MAC ACK copied from the TxOptions parameter as supplied by the GP-DATA.request primitive.

The parameter UseCSMA of the TxOptions is an extension to the MCPS-DATA.request and shall be propagated by the stub to the MAC layer. When UseCSMA is FALSE, CSMA/CA shall be skipped for the transmission of this GPDF. On reception of the MCPS-DATA.confirm, the stub calls GP-DATA.confirm with Status value copied from the MCPS-DATA.confirm.

Subsequently, and if no matching entry is found in the *gpTxQueue*, the stub indicates reception of the GPDF to the next higher layer, by calling GP-DATA.indication. If *SecurityLevel* was 0b00, the stub calls GP-DATA.indication with the Status NO_SECURITY; if *SecurityLevel* was 0b01 – 0b11, the stub calls GP-DATA.indication with the Status SECURITY_SUCCESS.

G.5 Green Power Security Stub Operations

G.5.1 Per GPDF Security Level and Key Selection

The Inter-PAN APS shall:

- For the incoming secured GPDF: use the parameters supplied by the GP-SEC.response.
- For the outgoing secured GPDF: use the same key and protection level as for the triggering GPDF.

G.5.2 Construction of AES Nonce

The AES nonce, defined by the current specification (see section 4.5.2.2) to have the format as depicted in Figure G.11, is used for security operations and shall be constructed in the following way.

Figure G.11 Format of the AES Nonce for Green Power Device Frames

Octets: 8	**4**	**1**
Source Address	Frame Counter	Security Control

For *ApplicationID* = 0b000, the *Source address* parameter shall take the value:

- for the incoming secured GPDF (i.e. the GPDF sent by the GPD): SourceAddress[63:32] = SrcID, SourceAddress[31:0] = SrcID;

- for the outgoing secured GPDF (i.e. the GPDF sent to the GPD): SourceAddress[63:32] = SrcID, SourceAddress[31:0] = 0;

The SrcID is little Endian (LSB first).

For example, if the SrcID = 0x87654321, the Source address parameter takes the following values:

- for the incoming secured GPDF: 0x8765432187654321 = { 0x21, 0x043, 0x65, 0x87, 0x21, 0x43, 0x65, 0x87 };

- for the outgoing secured GPDF: 0x8765432100000000 = { 0x00, 0x00, 0x00, 0x00, 0x21, 0x43, 0x65, 0x87 }.

For *ApplicationID* = 0b010, the *Source address* parameter shall take the value of the IEEE address of the GPD, for both incoming and outgoing secured GPDF.

Frame counter parameter shall take the value:

- for the incoming secured GPDF: 4B frame counter for this GPD, part or whole of which is being transmitted in the GPDF:
  - o if SecurityLevel was 0b01: the frame counter value is derived as described in A.3.7.2.4
- For the outgoing secured GPDF: the 4B value of frame counter that was last used by this GPD (i.e. the frame counter value from the GPDF received from this GPD with RxAfterTx=TRUE that immediately precedes the sending of this frame to the GPD).

Security control field, defined to be part of the AES nonce by the current specification and formatted as shown in Figure G.12, is never exchanged between the GP-capable devices. Thus, for interoperability, the values used shall be as defined below.

Figure G.12 Format of the Security Control field of the AES Nonce for Green Power Device Frames

Bits: 0-2	**3-4**	**5**	**6-7**
Security level	Key Identifier	Extended Nonce	Reserved

- Security level = 0b101
- Key identifier = 0b00
- Note that this security level and Key identifier are never transmitted and are NOT used for determining the transformation applied to the packet, since those are governed by the Security sub- field of the NWK Frame Control field of the GPDF. The values here are defined for interoperability only.
- Extended nonce = 0b0
- Reserved =
  - o For *ApplicationID* = 0b000 and for incoming secured GPDF (i.e. GPDF sent by GPD): Reserved = 0b00;
  - o For outgoing secured GPDF (i.e. GPDF sent to GPD) with an *ApplicationID* = 0b010: Reserved = 0b11.

The *Nonce* shall be formatted little endian, i.e. LSB first. Also the fields *Source address* and *Frame counter* shall be little endian, i.e. LSB first.

G.5.3 Initialization

If the *SecurityLevel* field of the GPDF has the value 0b01, the following transformation applies.

The definition Payload is applied to the following fields of the GPDF:

Payload = GPD CommandID || GPD Command Payload.

The definition Header is applied to the following fields of the GPDF:

Header = MAC sequence number || MAC addressing fields || NWK Frame Control || Extended NWK Frame Control || SrcID.

The following definitions apply:

- For the MAC sequence number field as part of the Header
  - In case of an incoming frame, the MAC sequence number from the received frame is used.
  - In case of an outgoing frame, 1LSB of the Security Frame Counter is used for security processing. Note: the 1LSB of the Security Frame Counter is independent of the macDSN attribute the MAC layer will use to transmit the frame.
- MAC addressing fields = are as in the received frame / as requested by the application;
- SrcID field = as in the received frame / as requested by the application (i.e. only for ApplicationID = 0b000).

If the *SecurityLevel* field of the GPDF has the value 0b10 or 0b11, the following transformation applies.

The definition *Payload* is applied to the following fields of the GPDF:

Payload = GPD CommandID || GPD Command Payload.

The definition Header is applied to the following fields of the GPDF:

Header = NWK Frame Control || Ext NWK Frame Control || SrcID || Frame counter; whereby the SrcID field is only present if the ApplicationID = 0b000.

G.5.4 Outgoing frame encryption and authentication

Determine the security level, as described in section G.5.1, and perform initialization, as described in section G.5.3.

G.5.4.1 CCM* execution

Execute the CCM* mode encryption and authentication operation, as specified in Annex A. The following parameters are used:

- The parameter M is =4, which means that 4B MIC is calculated (irrespective of *gpdSecurityLevel*).
- Nonce is constructed as described in section G.5.2.
- The bit string Key determined as described in section G.4.4.
- If the frame requires encryption (as indicated by *gpdSecurityLevel* = 0b11),
  - the octet string a shall be the Header, as defined in section G.5.3,
  - and the octet string m shall be the string Payload, as defined in G.5.3,
- Otherwise if the security level, as indicated by the *gpdSecurityLevel* parameter equal to 0b10 or 0b01, does not require encryption,

  - The octet string a shall be the string Header || Payload, as defined in G.5.3,
  - The octet string m shall be a string of length zero.

The output CCM* is the string *c*, which consists of right-concatenation of the encrypted message Cipher text and the encrypted authentication tag *U*.

G.5.4.2 Constructing protected GPDF

For transmission of the protected GPDF:

- If the security level, as indicated by gpdSecurityLevel = 0b01:
  - The fields GPD *CommandID* and GPD Command Payload remain unmodified;
  - 2 LSB of U are inserted into GPDF MIC field.
  - Then, the data unit is passed down using the GP-DATA.request. The MAC layer will fill the MAC Sequence Number field with the value of the macDSN attribute of the MAC PIB. Note: the macDSN attribute is independent of the 1LSB of the security frame counter used to protect the frame.
- Else, if the security level, as indicated by *gpdSecurityLevel* = 0b10:
  - The fields GPD *CommandID* and *GPD Command Payload* remain unmodified;
  - 4 LSB of U are inserted into GPDF MIC field.
  - The *Frame counter* used for frame protection is inserted into GPDF Security frame counter field.
- Else if the security level, as indicated by the *gpdSecurityLevel* = 0b11:
  - The *Ciphertext* is used as Payload, i.e. the *Ciphertext* replaces the fields *GPD CommandID* and *GPD Command payload*;
  - 4 LSB of U are inserted into *GPDF MIC field*;
  - The *Frame counter* used for frame protection is inserted into *GPDF Security frame counter field*.

G.5.5 Incoming frame decryption and authentication check

Determine the security level, as described in section G.5.1, and perform initialization, as described in section G.5.3.

The following parameters are used for CCM* mode encryption and authentication operation, as specified in Annex A:

- The parameter M is 4.
- Nonce is constructed as described in section G.5.2.
- The bit string Key determined as described in section G.4.4.

If decryption is required (*SecurityLevel* 0b11), proceed with CCM* as specified in Annex A.2.3, by using *PlaintextData* = encrypted GPD CommandID || encrypted GPD Command Payload from the received GPDF.

For authentication (for all *SecurityLevel* 0b01 - 0b11), calculate the *U*, as defined in section G.5.4.1, taking the decrypted GPD *CommandID* and GPD Command Payload fields as Payload, and the Header fields as defined in section G.5.3. Subsequently, compare the MIC field of the received GPDF with the corresponding number of LSB of the calculated U.

Subsequently, the results are evaluated as described in section G.5.6.

G.5.6 Reporting to the next higher layer

If the authentication is successful, stub calls GP-DATA.indication with Status SECURITY_SUCCESS and carrying the unprotected GPD CommandID and GPD Command Payload.

If the authentication is not successful, and

- *SecurityLevel*=0b10 or 0b11
- or *SecurityLevel* = 0b01 and *gppSecurityWindow* = 0,

The stub calls GP-DATA.indication with Status AUTH_FAILED and carrying the protected GPD CommandID and GPD Command Payload.

Otherwise, if the authentication is not successful and *SecurityLevel*=0b01 and if *gppSecurityWindow* parameter >0, the gppSecurityWindow is decremented and Frame Counter is modified as follows: the second LSB of the Frame Counter used in the previous run is incremented by 1, and the LSB is over-written with the MAC sequence number field from the received GPDF. Then, the processing as described in section G.5.5 is performed.

G.6 Inter-PAN Best Practices

Network Channel Manager Inter-PAN support is not specified in Annex E of the core stack specification ([B3]). New channel notifications will not be broadcast Inter-PAN. Inter-PAN devices which do not receive the network channel change will need to perform the network discovery procedure described in B.3.4.

It is recommended that devices that use Inter-PAN should implement a whitelist of known accepted commands and constrain the list to only the necessary commands. Inter-PAN commands should carefully screened by the receiving device since they can be sent by devices that do not have network security credentials and are performing an active attack.

This page intentionally left blank.

ANNEX H SECURITY TEST VECTORS FOR GREEN POWER DEVICE FRAMES

H.1 Overview

The parameters marked ***bold and italics*** are dependent on device application and capabilities and thus could have other values.

Note: ‘||’ in this pseudo-code means concatenation.

All test vectors use ApplicationID = 0x00.

H.2 Security Test Vectors for a Shared Key

H.2.1 Common Settings

GP Security Key = [0xC0 , 0xC1 , 0xC2 , 0xC3 , 0xC4 , 0xC5 , 0xC6 , 0xC7 , 0xC8 , 0xC9 , 0xCa , 0xCb , 0xCc , 0xCd , 0xCe , 0xCf] = 0xCFCECDCCCBCAC9C8C7C6C5C4C3C2C1C0

MAC fields:

- Dest PANId = 0xffff
- Dest Addr = 0xffff
- MAC SeqNum = 0x02

NWK fields:

- NWK FC := [Ext NWK Header = 0b1 || ***Auto-Commissioning =0b0*** || ZigBee Protocol 0b0011 || Frame type =0b00] → [0b10001100] 0x8c
- GPD SrcID = 0x87654321
- Security Frame Counter = 0x00000002

Application fields:

- GPD CommandID = 0x20 (OFF)
- No data payload

H.2.2 SecurityLevel = 0b01

H.2.2.1 Transmitted Packet

Transmitted packet = MAC FC || MAC header || GP stub NWK header || Payload || MIC

Transmitted test packet

12 01 08 02 FF FF FF FF 8C 08 21 43 65 87 20 B7 55

Note: even for SecurityLevel = 0b01, 4B MIC (U) is calculated, of which only part is transmitted in the packet.

H.2.2.2 Inputs

NWK fields:

Extended NWK FC = [Direction = 0b0 || RxAfterTx = 0b0 || SecurityKey = 0b0 ||SecurityLevel = 0b01 || ApplicationID = 0b000] = 0b00001000 = 0x08

H.2.2.3 Green Power Security Calculation

Definitions

Nonce N = [0x21, 0x43, 0x65, 0x87, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00, 0x05]

a = header || Payload

Header = MAC sequence number || MAC addressing fields || NWK FC || NWK_EXT FC || SrcID.

Header Construction

1. header = 0x02 || 0xffff || 0xffff || 0x8c || 0x08 || 0x87654321
2. header = [0x02, 0xff, 0xff, 0xff, 0xff, 0x8c, 0x08, 0x21, 0x43, 0x65, 0x87]
3. payload = 0x20
4. a = 0x02 || 0xffff || 0xffff || 0x8c || 0x08 || 0x87654321 || 0x20
5. a = [0x02, 0xff, 0xff, 0xff, 0xff, 0x8c, 0x08, 0x21, 0x43, 0x65, 0x87, 0x20]

Calculation

1. l(a) = 0x0c
2. L(a) = 0x00 0x0c
3. AddAuthData = L(a) || a || padding
4. AddAuthData = [0x00, 0x0c, 0x02, 0xff, 0xff, 0xff, 0xff, 0x8c, 0x08, 0x21, 0x43, 0x65, 0x87, 0x20, 0x00, 0x00]
5. Flags = [Reserved = 0b0 ||Adata = 0b1 || (M-2)/2 = 0b001 || (L-1) = 0b001 → 0x49]
6. B0 = [Flags =0x49|| Nonce N = 0x21 0x43 0x65 0x87 0x21 0x43 0x65 0x87, 0x02, 0x00, 0x00, 0x00, 0x05 || 0x00 0x00]

Result

1. U = 0xD76F55B7
2. MIC = 2LSB of U = 0x55B7 = [0xB7, 0x55]

H.2.3 SecurityLevel = 0b10

H.2.3.1 Transmitted Packet

Transmitted packet = MAC FC || MAC header || GP stub NWK header || Payload || MIC

Transmitted test packet

18 01 08 02 FF FF FF FF 8C 10 21 43 65 87 02 00 00 00 20 CF 78 7E 72

H.2.3.2 Inputs

NWK fields:

NWK FC Extended = [Direction = 0b0 || RxAfterTx = 0b0 || SecurityKey = 0b0 ||SecurityLevel = 0b10 || ApplID = 0b000] → 0b00010000 → 0x10

H.2.3.3 GP Security Calculation

Definitions

Nonce N = [0x21, 0x43, 0x65, 0x87, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00, 0x05]

a = header || Payload

Header = NWK FC || NWK_EXT FC || SrcID || Security Frame Counter.

header = 0x8c || 0x10 || 0x87654321 || 0x00000002

header = [0x8c, 0x10, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00]

payload = 0x20

a = 0x8c || 0x10 || 0x87654321 || 0x00000002 || 0x20

a = [0x8c, 0x10, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00; 0x20]

Calculation

1. l(a) = 0x0b
2. L(a) = 0x00 0x0b
3. AddAuthData = L(a) || a || padding
4. AddAuthData = [0x00, 0x0b, 0x8c, 0x10, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00]
5. Flags = [Reserved = 0b0 ||Adata = 0b1 || (M-2)/2 = 0b001 || (L-1) = 0b001 □ 0x49]
6. B0 = [Flags =0x49|| Nonce N = 0x21 0x43 0x65 0x87 0x21 0x43 0x65 0x87, 0x02, 0x00, 0x00, 0x00, 0x05 || 0x00 0x00]

Result

U = 0x727E78CF

MIC = FULL U = 0x727E78CF = [0xCF, 0x78, 0x7E, 0x72]

H.2.4 SecurityLevel = 0b11

H.2.4.1 Transmitted packet

Transmitted packet = MAC FC || header || Payload || MIC

Transmitted packet

18 01 08 02 FF FF FF FF 8C 18 21 43 65 87 02 00 00 00 83 CA 43 24 DD

H.2.4.2 Inputs

NWK fields:

NWK FC Extended = [Direction = 0b0 || RxAfterTx = 0b0 || SecurityKey = 0b0 ||SecurityLevel = 0b11 || ApplID = 0b000] → 0b00011000 → 0x18

H.2.4.3 GP Security Calculation

Definitions

Nonce N = [0x21, 0x43, 0x65, 0x87, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00, 0x05]

a = Header

m = Payload

Header = NWK FC || NWK_EXT FC || SrcID || Security Frame Counter.

header = 0x8c || 0x18 || 0x87654321 || 0x00000002

header = [0x8c, 0x18, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00]

payload = 0x20

a = 0x8c || 0x18 || 0x87654321 || 0x00000002

a = [0x8c, 0x18, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00]

m = 0x20

Calculation

1. l(a) = 0x0a
2. L(a) = 0x00 0x0a
3. AddAuthData = L(a) || a || padding
4. AddAuthData = [0x00, 0x0a, 0x8c, 0x18, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00]
5. PlaintextData = m || padding
6. PlaintextData = [0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]

7. AuthData = AddAuthData || PlaintextData

8. AuthData = [0x00, 0x0a, 0x8c, 0x18, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]
9. FlagsAuth = [Reserved = 0b0 ||Adata = 0b1 || (M-2)/2 = 0b001 || (L-1) = 0b001 □ 0x49]
10. B0 = [FlagsAuth =0x49|| Nonce N = 0x21 0x43 0x65 0x87 0x21 0x43 0x65 0x87, 0x02, 0x00, 0x00, 0x00, 0x05 || l(m) = 0x00 0x01]
11. B1 = [0x00, 0x0a, 0x8c, 0x18, 0x21, 0x43, 0x65, 0x87, 0x02, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00]
12. B2 = [0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]
13. FlagsEncrypt = [Reserved = 0b0 ||[Reserved = 0b0 || 0b000 || (L-1) = 0b001 □ 0x01]
14. Ai = [FlagsEncrypt = 0x01 || Nonce N = 0x21 0x43 0x65 0x87 0x21 0x43 0x65 0x87, 0x02, 0x00, 0x00, 0x00, 0x05 || Counter = 0x00 0x0i]

Result

U = 0xDD2443CA

MIC = FULL U = 0xDD2443CA = [0xCA, 0x43, 0x24, 0xDD]

Cipher = 0x83

H.3 Security test vectors for an individual key

H.3.1 Common settings

GP Security Key = [0xC0 , 0xC1 , 0xC2 , 0xC3 , 0xC4 , 0xC5 , 0xC6 , 0xC7 , 0xC8 , 0xC9 , 0xCa , 0xCb , 0xCc , 0xCd , 0xCe , 0xCf] = 0xCFCECDCCCBCAC9C8C7C6C5C4C3C2C1C0

Nonce = 21 43 65 87 21 43 65 87 02 00 00 00 05

MAC fields:

- Dest PANId = 0xffff
- Dest Addr = 0xffff
- MAC SeqNum = 0x02

NWK fields:

- NWK FC := [Ext NWK Header = 0b1 || ***Auto-Commissioning*** =0b0|| ZigBee Protocol 0b0011 || Frame type =0b00] → [0b10001100] → 0x8c
- GPD SrcID = 0x87654321
- Security Frame Counter = 0x00000002

Application fields:

- GPD CommandID = ***0x20 (OFF)***
- No data payload

H.3.2 SecurityLevel=0b01

Extended NWK FC = [Direction = 0b0 || RxAfterTx = 0b0 || SecurityKey = 0b1 || SecurityLevel = 0b01 || ApplID = 0b000] → 0x28

Over the air packet:

12 01 08 02 FF FF FF FF 8C 28 21 43 65 87 20 61 02

Note: even for SecurityLevel = 0b01, 4B MIC (U) is calculated, of which only part is transmitted in the packet.

H.3.3 SecurityLevel=0b10

Extended NWK FC = [Direction = 0b0 || RxAfterTx = 0b0 || SecurityKey = 0b1 || SecurityLevel = 0b10 || ApplID = 0b000] → 0x30

Over the air packet:

18 01 08 02 FF FF FF FF 8C 30 21 43 65 87 02 00 00 00 20 AD 69 A9 78

H.3.4 SecurityLevel=0b11

Extended NWK FC = [Direction = 0b0 || RxAfterTx = 0b0 || SecurityKey = 0b1 || SecurityLevel = 0b11 || ApplID = 0b000] → 0x38

Over the air packet:

18 01 08 02 FF FF FF FF 8C 38 21 43 65 87 02 00 00 00 83 5F 1A 30 34

H.4 Security test vectors for bidirectional operation

H.4.1 Common settings

For all frames:

NWK Frame Type sub-field = 0b00

ZigBee Protocol Version sub-field = 0b0011

Auto-commissioning sub-field = 0b0

Extended NWK Frame Control Present sub-field = 0b1

GPD SrcID = 0x87654321

Security Frame Counter = 0x44332211

Security Key = { 0xC0 0xC1 0xC2 0xC3 0xC4 0xC5 0xC6 0xC7 0xC8 0xC9 0xCA 0xCB 0xCC 0xCD 0xCE 0xCF }

For incoming frames (from GPD to GPP / GPS):

RxAfterTx sub-field = 0b1

Direction sub-field = 0b0

MAC Seq Nbr

For SecurityLevel = 0b10 or 0b11: 0x01

For SecurityLevel = 0b01: 0x11 being LSB of Security Frame Counter

GPD CommandID = 0x20 (OFF)

GPD Command payload = (No payload)

For outgoing frames (from GPP/GPS to GPD):

RxAfterTx sub-field = 0b0

Direction sub-field = 0b1

MAC Seq Nbr = 39

GPD CommandID = 0xF3 (Channel Configuration)

GPD Command payload = 0x00 (channel 11)

Note: For SecurityLevel = 0b01: 0x11 (LSB of Security Frame Counter) is used for MIC calculation.

H.4.2 Security test vectors for a shared key

For all test vectors with a shared security key:

Security Key sub-field of Extended NWK Frame Control field = 0b0 (shared key)

H.4.2.1 SecurityLevel = 0b01

Incoming frame (GPD to GPP / GPS):

0x12 0x01 0x08 0x11 0xFF 0xFF 0xFF 0xFF 0x8C 0x48 0x21 0x43 0x65 0x87 0x20 0x16 0x0B

Outgoing frame (GPP/GPS to GPD):

0x13 0x01 0x08 0x11 0xFF 0xFF 0xFF 0xFF 0x8C 0x88 0x21 0x43 0x65 0x87 0xF3 0x00 **0x6C 0xFD**

Full 4B MIC: 0x4782**FD6C**

H.4.2.2 SecurityLevel = 0b10

Incoming frame (GPD to GPP / GPS)

0x18 0x01 0x08 0x01 0xFF 0xFF 0xFF 0xFF 0x8C 0x50 0x21 0x43 0x65 0x87 0x11 0x22 0x33 0x44 0x20 **0xF6 0x36 0x78 0x9E**

Full 4B MIC: **0x9E7836F6**

Outgoing frame (GPP/GPS to GPD)

0x19 0x01 0x08 0x39 0xFF 0xFF 0xFF 0xFF 0x8C 0x90 0x21 0x43 0x65 0x87 0x11 0x22 0x33 0x44 0xF3 0x00 **0xCC 0xA0 0xBB 0x2E**

Full 4B MIC: **0x2EBBA0CC**

H.4.2.3 SecurityLevel = 0b11

Incoming frame (GPD to GPP / GPS)

0x18 0x01 0x08 0x01 0xFF 0xFF 0xFF 0xFF 0x8C 0x58 0x21 0x43 0x65 0x87 0x11 0x22 0x33 0x44 0x2A **0x3D 0x17 0x0A 0xAA**

Encrypted data: 0x2A

Full 4B MIC: **0xAA0A173D**

Outgoing frame (GPP/GPS to GPD)

0x19 0x01 0x08 0x39 0xFF 0xFF 0xFF 0xFF 0x8C 0x98 0x21 0x43 0x65 0x87 0x11 0x22 0x33 0x44 0x9E 0x7E **0x14 0x0F 0xB5 0xDA**

Encrypted data: 0x9E 0x7E

Full 4B MIC: **0xDAB50F14**

H.4.3 Security test vectors for an individual key

For all test vectors with an individual key:

Security Key sub-field in NWK Ext field = 0b1 (individual key)

H.4.3.1 SecurityLevel = 0b01

Incoming frame (GPD to GPP / GPS)

0x12 0x01 0x08 0x11 0xFF 0xFF 0xFF 0xFF 0x8C 0x68 0x21 0x43 0x65 0x87 0x20 0x43 0x82

Outgoing frame (GPP/GPS to GPD)

0x13 0x01 0x08 0x11 0xFF 0xFF 0xFF 0xFF 0x8C 0xA8 0x21 0x43 0x65 0x87 0xF3 0x00 **0x71 0x15**

Full 4B MIC: **0xFA601571**

H.4.3.2 SecurityLevel = 0b10

Incoming frame (GPD to GPP / GPS)

0x18 0x01 0x08 0x01 0xFF 0xFF 0xFF 0xFF 0x8C 0x70 0x21 0x43 0x65 0x87 0x11 0x22 0x33 0x44 0x20 **0x6E 0xA9 0x51 0xBC**

Full 4B MIC: **0xBC51A96E**

Outgoing frame (GPP/GPS to GPD)

0x19 0x01 0x08 0x39 0xFF 0xFF 0xFF 0xFF 0x8C 0xB0 0x21 0x43 0x65 0x87 0x11 0x22 0x33 0x44 0xF3 0x00 **0xF9 0xF1 0x7C 0x8A**

Full 4B MIC: **0x8A7CF1F9**

H.4.3.3 SecurityLevel = 0b11

Incoming frame (GPD to GPP / GPS)

0x18 0x01 0x08 0x01 0xFF 0xFF 0xFF 0xFF 0x8C 0x78 0x21 0x43 0x65 0x87 0x11 0x22 0x33 0x44 0x2A **0xD9 0xF0 0x08 0x6D**

Encrypted data: 0x2A

Full 4B MIC: **0x6D08F0D9**

Outgoing frame (GPP/GPS to GPD)

0x19 0x01 0x08 0x39 0xFF 0xFF 0xFF 0xFF 0x8C 0xB8 0x21 0x43 0x65 0x87 0x11 0x22 0x33 0x44 0x9E 0x7E **0xD6 0x6E 0x60 0x08**

Encrypted data: 0x9E 0x7E

Full 4B MIC: **0x08606ED6**

