ZIGBEE SPECIFICATION

ZigBee Document 053474r13	
December 1, 2006 3:10 pm	
Sponsored by: ZigBee Alliance	
Accepted by	This document has not been accepted by the Alliance BOD.
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

December 1, 2006

NOTICE OF USE AND DISCLOSURE

ZigBee Alliance, Inc.
2400 Camino Ramon, Suite 375
San Ramon, CA 94583

DOCUMENT HISTORY

ZigBee Specification History

Version Number	Date	Comments
	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, 053474r07
r07	April 28, 2006	Changes made per Editorial comments on spreadsheet,
r13	December 1, 2006 3:08 pm	ZigBee-2006 Specification (see letter ballot comments and resolution in ZigBee document 064112)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

ZIGBEE PROTOCOL OVERVIEW

1.1 Protocol Description

The ZigBee Alliance is developing 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 ZigBee stack architecture, depicted in Figure 1.1, is based on the standard Open Systems Interconnection (OSI) seven-layer model (see [B14]) but defines only those layers relevant to achieving functionality in the intended market space. The IEEE 802.15.4-2003 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 is comprised of the application support sub-layer (APS), the ZigBee device objects (ZDO) and the manufacturer-defined application objects.

IEEE 802.15.4-2003 has two PHY layers that operate 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 IEEE 802.15.4-2003 PHY layers can be found in [B1].

The IEEE 802.15.4-2003 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 802.15.4-2003 MAC sub-layer can be found in [B1].

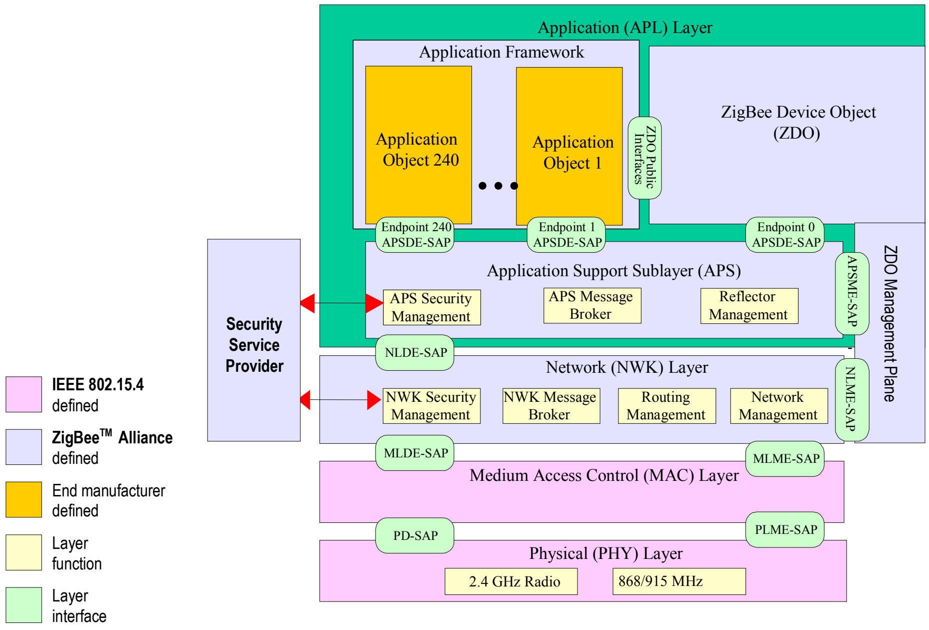


Figure 1.1 Outline of the ZigBee Stack Architecture

The responsibilities of the ZigBee NWK layer shall include mechanisms used to:

- Join and leave a network
- Apply security to frames
- Route frames to their intended destinations
- Discover and maintain routes between devices
- Discover one-hop neighbors
- Store of pertinent neighbor information

The NWK layer of a ZigBee coordinator (see "Network Topology") is responsible for starting a new network, when appropriate, and assigning addresses to newly associated devices.

The ZigBee application layer consists of the APS, the Application Framework (AF), the ZDO and the manufacturer-defined application objects.

The responsibilities of the APS sub-layer include:

- Maintaining tables for binding, defined as the ability to match two devices together based on their services and their needs
- Forwarding messages between bound devices

The responsibilities of the ZDO include:

- Defining the role of the device within the network (e.g., ZigBee coordinator or end device)
- Initiating and/or responding to binding requests
- Establishing a secure relationship between network devices.

The ZDO is also responsible for discovering devices on the network and determining which application services they provide.

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, and 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 802.15.4-2003 specification. Mesh networks shall allow full peer-to-peer communication. ZigBee routers in mesh networks shall not emit regular IEEE 802.15.4-2003 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 Conventions

1.2.1.1 Symbols and Notation

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

1.2.1.2 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 ANSI X9.63-2001 [B7]. All octets shall be represented as binary strings in most-significant-bit first order.

1.2.1.3 Entities

Throughout this specification, each entity shall be a DEV and shall be uniquely identified by its 64-bit IEEE device address [B1]. The parameter *entlen* shall have the integer value 64.

1.2.1.4 Transmission Order

Unless otherwise indicated, the transmission order of all frames in this specification follow the conventions used in IEEE Std. 802.15.4-2003 [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 the order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.

1.2.1.5 Strings and String Operations

A string is a sequence of symbols over a specific set (e.g., 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 rightmost 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:

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
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*	Enhanced counter with CBC-MAC mode of operation
CSMA-CA	Carrier sense multiple access – collision avoidance
FFD	Full function device
GTS	Guaranteed time slot
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
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
RREP	Route reply
RREQ	Route request
RN	Routing node
SAP	Service access point
SKG	Secret key generation
SKKE	Symmetric-key key establishment
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 the IEEE Style Manual [B12].

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)*.

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 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 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. Products that implement future revisions of this specification may set these fields as defined by the specification.

ZigBee Protocol Stack: The name of the ZigBee protocol revision 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. Frames defined in earlier revision of the ZigBee Protocol Stack specification published in March 2005 employ a value of 0x01. Frames defined in subsequent versions of the specification may set the protocol version sub-field of the frame control field to a value other than 0x01 or 0x02. A ZigBee device that conforms to this version of the specification may elect to provide backward compatibility with the 2005 revision of the specification. It shall do so by supporting all frame formats and features as specified in that version of the specification and specify this support by using 0x01 as the protocol version when doing so. In addition, the device conforming to this specification may support frame formats and features consistent with this version of the specification and shall, when doing so, specify this support by using 0x02 as the protocol 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 a multiple ZigBee protocol version. The protocol version of the network to join shall be determined by a backwardly compatible device via examination of 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 ZigBee v1.1 beacon payload support. A ZigBee device shall discard all frames that carry a protocol version sub-field value other than 0x0 or 0x02, and shall process only protocol versions of 0x01 or 0x02, consistent with the protocol version of the network that the device participates within.

Strings and string operations: A string is a sequence of symbols over a specific set (e.g., 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 rightmost *n* symbols. An octet is a symbol string of length 8. In our context, all octets are strings over the binary alphabet.

1.4.1.2 ZigBee Definitions

For the purposes of this standard, the following terms and definitions apply. Terms not defined in this clause can be found in IEEE P802.15.4 §3 [B1] or in ANSI X9.63-2001 §2.1 [B7].

Access control list: 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 (e.g., cryptographic keys, frame counts, key counts, security level information) used for securely communicating with other devices.

Active network key: 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: a key available for use in lieu of the active Network key to process incoming NWK frames.

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

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

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

APS command frame: an APS frame that contains neither source nor endpoints.

Association: the service provided by the IEEE 802.15.4-2003 MAC sub-layer that is used to establish membership in a network.

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

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

Binding: 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: the transmission of a message to every device within a network.

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

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

Broadcast transaction table: collection of broadcast transaction records.

Cluster: is a container for one or more attributes in a command structure that employs attributes or is synonymous with a message in a command structure that does not employ 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: 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.

Component: a component consists of a physical object (e.g., switch, controller, etc.) and its corresponding application profile(s).

Coordinator: an IEEE 802.15.4-2003 device responsible for associating and disassociating devices into its PAN. A coordinator must be a full function device (FFD).

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

Data key: a key shared between two devices for peer-to-peer data communications.

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

Device application: a special application that is responsible for Device operation. The Device Application resides on Endpoint 0 by convention and contains logic to manage the Devices networking and general maintenance features.

Device description: 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: a mode of addressing in which the destination of a frame is completely specified in the frame itself.

Direct binding: the procedure through which the uppers layers of a device which maintains a binding table in the APS can create or remove a binding link in that binding table.

Direct transmission: frame transmission using direct addressing.

Disassociation: the service provided by the IEEE 802.15.4-2003 MAC sub-layer that is used to discontinue the membership of a device in a network.

End application: applications that reside on Endpoints 1 through 240 on a Device. The End Applications implement features that are non-networking and ZigBee protocol related.

End device binding: 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: a particular component within a unit. Each ZigBee device may support up to 240 such components.

Endpoint address: 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: 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.

First hop indirect frame: a period in the transit of a frame that has been indirectly addressed when only the source address appears in the frame.

Indirect addressing: the ability for resource limited devices to communicate without having to know the address of the desired destination. Indirect transmissions shall include only the source endpoint-addressing field along with the Indirect Addressing bit set in the APDU and are directed to the ZigBee Coordinator by the source. The ZigBee Coordinator is expected to lookup the source address/endpoint/cluster ID within its binding table and re-issues the message to each corresponding destination address/endpoint.

Information base: 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: A mechanism that involves the execution of a protocol by two devices to derive a mutually shared secret key.

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

Key-transport key: A key used to protect key transport messages.

Key update: 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 coordinator: A ZigBee Coordinator or ZigBee Router, which is the IEEE 802.15.4 coordinator, which processed the association request for a specific End Device.

Local device: A ZigBee Coordinator, ZigBee Router or ZigBee End Device which issues a ZigBee Device Object request message. The Local device is the client in the message exchange and the message processing description within the ZigBee Device Profile refers to client processing from the perspective of the "Local device".

Link key: A key that is shared between two devices within a PAN.

Master key: A shared key used during the execution of a symmetric-key key establishment protocol. The master key is the basis for long-term security between the two devices, and may be used to generate link keys.

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

Multihop network: 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: a personal area network that does not contain any devices that transmit beacon frames at a regular interval.

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

Network address: 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: time duration required by a broadcast transaction to reach every device of a given network.

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

Network service data unit: Information that is delivered as a unit through a network service access point.

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

Normal operating state: 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 mean 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.

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

Orphaned device: a device that has lost communication contact with or information about the ZigBee device through which it has its PAN membership.

PAN coordinator: The principal controller of an IEEE 802.15.4-2003-based network that is responsible for network formation and maintenance. The PAN coordinator must be a full function device (FFD).

PAN information base: A collection of variables in the IEEE 802.15.4-2003 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 which are accessible to ZigBee device objects only and unavailable to the end applications.

Profile: 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.

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

Proxy binding: the procedure through which a device can create or remove a binding link on the ZigBee coordinator between two devices (none of which may be the device itself).

Public method: attributes which are accessible to End Applications.

Radio: the IEEE 802.15.4-2003 radio that is part of every ZigBee device.

Remote device: A ZigBee Coordinator, ZigBee Router or ZigBee End Device which receives a ZigBee Device Object request message. The Remote device is the server in the message exchange and the message processing description within the ZigBee Device Profile refers to server processing from the perspective of the "Remote device".

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.

Symmetric-key key establishment: a mechanism by which two parties establish a shared secret, based on a pre-shared secret (a so-called master key).

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.

Unit: a component or collection of components that share a single ZigBee radio. Each unit has a unique 64-bit IEEE address and a 16-bit network address.

ZigBee coordinator: an IEEE 802.15.4-2003 PAN coordinator.

ZigBee device object: the portion of the application layer responsible for defining the role of the device within the network (e.g., 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 802.15.4-2003 RFD or FFD participating in a ZigBee network, which is neither the ZigBee coordinator nor a ZigBee router.

ZigBee router: an IEEE 802.15.4-2003 FFD participating in a ZigBee network, which is not the ZigBee coordinator but may act as an IEEE 802.15.4-2003 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 sub-clause.

1.5.1 ZigBee/IEEE References

[B1] Institute of Electrical and Electronics Engineers, Inc., IEEE Std. 802.15.4-2003, IEEE Standard for Information Technology — telecommunications and Information Exchange between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs). New York: IEEE Press. 2003.

[B2] IEEE 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, IEEE, 1985.

[B3] Document 03285r0: Suggestions for the Improvement of the IEEE 802.15.4 Standard, July 2003.

[B4] Document 02055r4: Network Requirements Definition, August 2003.

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/.

1.5.3 Informative References

[B12] 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/.

[B13] IEEE Standards Style Manual, published and distributed in May 2000 and revised on September 20, 2001. Available from http://standards.ieee.org/guides/style/.

[B14] ISO/IEC 7498-1:1994 Information technology — Open systems interconnection — Basic reference model: The basic model.

[B15] ISO/IEC 10731:1994, Information technology — Open Systems Interconnection — Conventions for the definition of OSI services.

[B16] ISO/IEC 9646-1:1991, Information technology — Open systems Interconnection — Conformance testing methodology and framework — Part 1: General concepts.

[B17] ISO/IEC 9646-7:1995, Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 7. Implementation conformance statements.

[B18] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, Boca Raton: CRC Press, 1997.

[B19] 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/.

[B20] 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/.

[B21] 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.

[B22] 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/.

[B23] P. Rogaway, D. Wagner, A Critique of CCM, IACR ePrint Archive 2003-070, April 13, 2003.

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 2003 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.

The responsibilities of the APS sub-layer include:

- Maintaining tables for binding, that is, the ability to match two devices together based on their services and their needs
- Forwarding messages between bound devices
- Group address definition, removal and filtering of group addressed messages
- Address mapping from 64 bit IEEE addresses to and from 16 bit NWK addresses
- Fragmentation, reassembly and reliable data transport

The responsibilities of the ZDO include:

- Defining the role of the device within the network (e.g., ZigBee coordinator or end device)
- Discovering devices on the network and determining which application services they provide

- Initiating and/or responding to binding requests
- Establishing a secure relationship between network devices

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 for the transport of application PDUs between two or more devices located on the same network including filtering of group addressed messages. The APSDE also support fragmentation and reassembly of packets larger than the supported data payload of the APSDU and reliable data transport.

The APSME provides security services, binding of devices, establishment and removal of group addresses and also maintains a database of managed objects, known as the APS information base (AIB). The AIB supports addressing mapping between 64 bit IEEE addresses and 16 bit NWK addresses.

2.1.2 Application Framework

The application framework in ZigBee is the environment in which application objects are hosted on ZigBee devices. Inside the application framework, the application objects send and receive data through the APSDE-SAP. The application objects perform the following functions through the ZDO public interfaces (see clause 2.5):

- Control and management of the protocol layers in the ZigBee device
- Initiation of standard network functions

The data service, provided by APSDE-SAP, includes request, confirm, response and indication primitives for data transfer. The request primitive supports data transfers between peer application object entities. The confirm primitive reports the results of a request primitive call. The indication primitive is used to indicate the transfer of data from the APS to the destination application object entity.

Up to 240 distinct application objects can be defined, each interfacing on an endpoint indexed from 1 to 240. 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 reserved for future use.

2.1.3 Addressing

2.1.3.1 Node Addressing

In Figure 2.1, there are two nodes, each containing a single radio. One node contains two switches and the other contains four lamps.

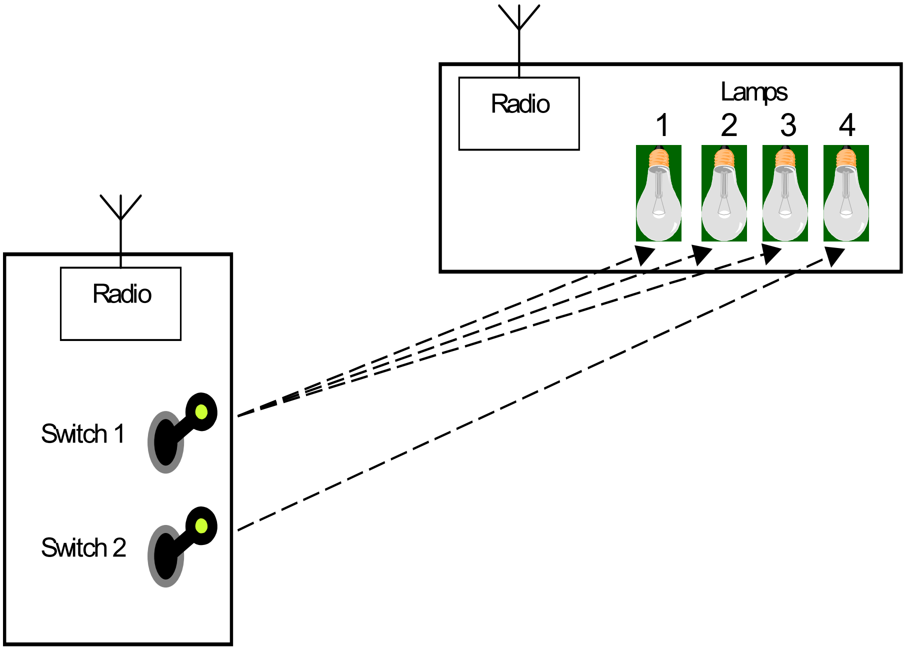


Figure 2.1 Multiple Subunits in a Single Node

A node contains one or more device descriptions and has a single IEEE 802.15.4 radio. In Figure 2.1, the individual parts of the nodes (the switches and lamps) are subunits containing one device description in each subunit. Each node is given an address when it joins the ZigBee network.

2.1.3.2 Endpoint Addressing

In Figure 2.1, it is required that switch 1 should control lamps 1, 2 and 3, while switch 2 should control only lamp 4. However, as the only addressable component is the radio, it is not possible to identify or address the individual subunits, so it would not be possible for switch 2 to turn on lamp 4 only.

ZigBee provides another level of sub-addressing, which is used in conjunction with the mechanisms of IEEE802.15.4. An endpoint number can be used to identify individual switches and lamps. For instance, in the example above, switch 1 could use endpoint 3, while switch 2 could use endpoint 21. Similarly, the lamps will each have their own endpoints. Endpoint 0 is reserved for device management and is used to address the descriptors in the node. Each identifiable subunit in a node (such as the switches and lamps) is assigned its own specific endpoint in the range 1-240.

Physical devices are described in terms of the data attributes that they contain. For instance, a thermostat might contain an output attribute "temperature" which represents the current temperature of a room. A furnace controller may take this attribute as an input and control the furnace according to the temperature value received from the thermostat. These two physical devices, including their attributes, would be described in the relevant device descriptions for those devices.

The simple room thermostat described has temperature-sensing circuitry, which can be queried by the external furnace controller. It advertises its service on an endpoint and the service is described in the simple description implemented on that endpoint.

A more complex version of the thermostat may also have an optional "heartbeat" report timer, which causes the device to report current room temperature after a set period. In this example, the ReportTime attribute specifies when reports are to be sent and writing a suitable time value to this attribute sets the frequency of these temperature reports. This implementation would advertise its services (in a list of cluster identifiers) on a different endpoint.

In order to allow product differentiation in the marketplace, manufacturers may add clusters containing extra attributes of their own in the context of one or more private profiles. These manufacturer-specific clusters do not form part of this or any other ZigBee specification and interoperability is not guaranteed for these clusters. Such services would be advertised on different endpoints from those described above.

2.1.4 Application Communication Fundamentals

2.1.4.1 Application Profiles

Application profiles are agreements for messages, message formats and processing actions that enable applications to create an interoperable, distributed application between applications that reside on separate devices. These application profiles enable applications to send commands, request data and process commands and requests. For instance, a thermostat on one node can communicate with a furnace on another node. Together, they cooperatively form a heating application profile. ZigBee vendors develop application profiles to provide solutions to specific technology needs.

Application profiles are simultaneously a means of unifying interoperable technical solutions within the ZigBee standard, as well as focusing usability efforts within a given marketing area. For example, it is expected that vendors of lighting equipment will want to provide ZigBee profiles that interoperate with several varieties of lighting types or controller types. Additional information on profiles is provided in clause 2.2 of this document.

2.1.4.2 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 profile. Binding decisions are taken by matching an output cluster identifier to an input cluster identifier, assuming both are within the same profile. In the thermostat example above, binding takes place on temperature, between a device with a temperature cluster identifier as output and a device with a temperature cluster identifier as input. The binding table contains the identifier for temperature along with the address of the source and destination devices.

2.1.5 Discovery

2.1.5.1 Device Discovery

Device discovery is the process whereby a ZigBee device can discover other ZigBee devices by initiating queries that are broadcast (of any broadcast address type) or unicast addressed. There are two forms of device discover requests: IEEE address requests and NWK address requests. The IEEE address request is unicast and assumes the NWK address is known. The NWK address request is broadcast to all RxOnWhenIdle devices and carries the known IEEE address as data payload.

Responses to the broadcast or unicast device discovery messages vary by logical device type as follows:

- ZigBee end device: responds to the device discovery query by sending its IEEE or NWK address (depending on the request).
- ZigBee coordinator device: responds to the query by sending its IEEE or NWK addresses and the NWK addresses of all devices that are associated with the ZigBee coordinator (depending on the request).

- ZigBee router device: responds to the query by sending its IEEE or NWK addresses and the NWK addresses of all devices that are associated with the ZigBee router (depending on the request).

A description of the procedure details, primitive calls, and applicable parameters is given in clause 2.4.

Discovery information may also be cached within the devices in the network designated as the Primary Discovery Cache device. The protocol for identifying these devices, requesting storage on them, and uploading discovery information is supplied in clause 1.5. The mechanics of this type of proxy discovery are as follows:

1. Discovery requests utilizing broadcast messaging are processed by discovery cache devices or the target devices themselves and those devices respond either on behalf of device information held in the cache or from their local descriptor information.
2. For unicast discovery requests, the request is directed either to the destination network address of the discovery cache or the device of interest and the network address of interest field is filled in to reflect the target of the discovery request.
3. A discovery command is also provided to enable a device to locate where discovery information can be obtained for a specified network address. The location of the discovery information can either be the device itself or a discovery cache device.

2.1.5.2 Service Discovery

Service discovery is the process whereby services available on endpoints at the receiving device are discovered by external 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). Service discovery utilizes the complex, user, node or power descriptors plus the simple descriptor further addressed by the endpoint (for the connected application object).

The service discovery process in ZigBee is key to interfacing devices within the network. Through specific requests for descriptors on specified nodes, broadcast requests for service matching and the ability to ask a device which endpoints support application objects, a range of options are available for commissioning tools and applications. See clause 2.4 for details on service discovery.

Service discovery information may also be cached in the network within the Primary discovery cache device. The protocol for identifying these devices, requesting storage on them, and uploading discovery information is supplied in Sub-clause 2.5.2.1. The mechanics of this type of proxy discovery are the same as those described in Sub-clause 2.5.2.1 for device discovery.

2.1.6 Binding

In ZigBee, there is an application level concept using cluster identifiers (and the attributes contained in them) on the individual endpoints or group addresses in different nodes. This is referred to as binding – the creation of logical links between complementary application devices and endpoints or group addresses In the example of sub-clause 2.1.3.2, a binding could be made between thermostat and the furnace controller. In Figure 2.2, switch 1 is bound with lamps 1-3, while switch 2 is bound with lamp 4 only.

The information about which cluster is bound between nodes is stored in a binding table. This is described fully in sub-clause 2.1.6 and is illustrated in Figure 2.2.

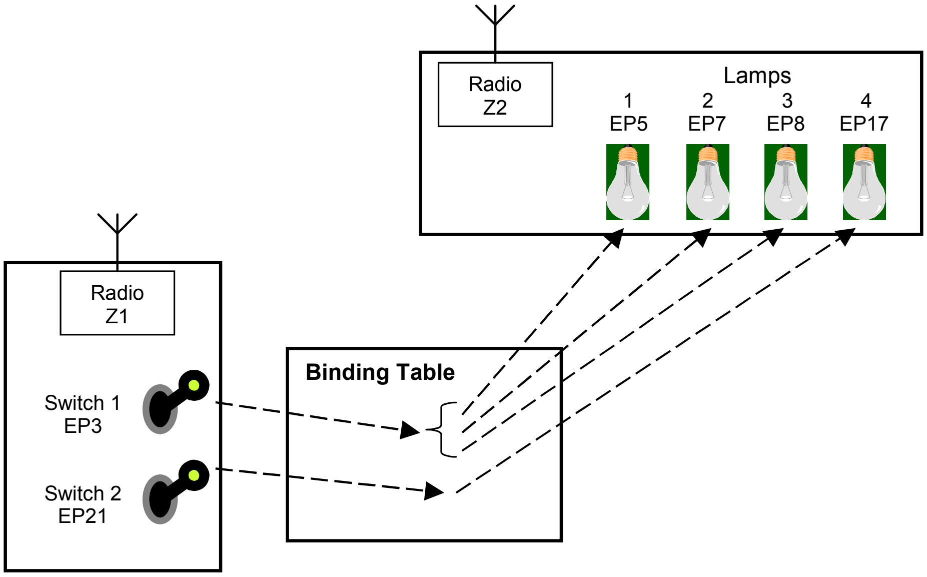


Figure 2.2 ZigBee Binding and the Binding Table

The use of a list of three entries in the binding table for switch 1 allows it to control three lamps, which could also be in separate nodes (with their own ZigBee radios). It is also possible for one lamp to be controlled by several switches: in this case there would be entries for each switch, all linked to the same lamp.

As an alternative, the endpoints describing the three lamps could have been assigned to a single group address in which case a single binding entry could be used. The choice of creating separate binding entries for individual destination application devices and endpoints or grouping the destination devices and endpoints into a single group address is left to the application.

Binding is always performed after a communications link has been established. Once a link has been established, the implementation decides whether a new node should be part of the network. This will depend on the security in operation for the application and how it is implemented. Binding is only allowed if the implemented security on all devices allows this (see Chapter 4).

The binding table is implemented in the source device of the binding pair or within a device designated as the binding table cache. The message reflection feature employs the binding table and can be deployed either on the source device or in the ZigBee coordinator. Some applications may need a duplicate of the binding table to be available in the event that the device storing the table fails. Backup of the binding table, and any other key ZigBee coordinator data, is the responsibility of application software, using ZDO primitives.

The details of the creation of binding links is covered in the ZigBee device profile (see clause 2.4).

2.1.7 Messaging

2.1.7.1 Direct Addressing

Once devices have been associated, commands can be sent from one device to another. A command is sent to an application object at the destination address (radio address plus its endpoint). Details of the commands can be found in 2.1.3. Note that binding is not a pre-requisite for using direct addressing.

Direct addressing assumes device discovery and service discovery have identified a particular device and endpoint, which supply a complementary service to the requestor. Specifically, direct addressing defines a means of directing messages to the device by including its full address and endpoint information.

2.1.7.2 Indirect Addressing

Use of direct addressing requires the controlling device to have knowledge of the following attributes of the target device it will communicate with:

- Address
- Endpoint
- Cluster identifier

Information about these attributes must be committed to a binding table on the ZigBee coordinator for message reflection, prior to the creation of an indirectly addressed message between the device pair.

A full IEEE 802.15.4 address amounts to 10 octets (PAN identifier plus 64-bit IEEE address) and a further octet is required for the endpoint. Extremely simple devices, such as battery-powered switches, may not want the overhead of storing this information, nor the software for acquiring this information. For these devices, indirect addressing will be more appropriate.

When a source device wishes to send a command to a destination using indirect addressing, instead of including the address of the destination device (which it does not know and has not stored), it omits the address and specifies indirect addressing via the APSDE-SAP. The included source address, source endpoint and cluster identifier in the indirect addressed message are translated via the binding table to those of the destination device(s) and the messages are relayed to each indicated destination.

Where a cluster contains several attributes, the cluster identifier is used for addressing and the attribute identifier is used in the command itself to identify a particular attribute within the cluster. The applications, however, can parse and utilize the attributes as defined within their profile.

2.1.7.3 Broadcast Addressing

An application may broadcast messages to all endpoints on a given destination device. This form of broadcast addressing is called application broadcast. The destination address shall be one of the 16-bit network broadcast addresses and the broadcast flag shall be set in the APS frame control field. The source shall include the cluster identifier, profile identifier and source endpoint fields in the APS frame (see sub-clause 2.2.5).

2.1.7.4 Group Addressing

An application may designate a collection of devices and specific endpoints on those devices to a single group address. The group address may then be used to direct outgoing clusters (and the attributes contained in them) to each of the devices and endpoints assigned to the group. The destination address shall be a 16-bit group address and the group address flag shall be set in the APS frame control field. The source shall include the cluster identifier, profile identifier and source endpoint fields in the APS frame.

2.1.8 ZigBee Device Objects

The ZigBee device objects (ZDO), represents 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), 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 to the lower portions of the ZigBee protocol stack, on endpoint 0, through the APSDE-SAP for data and through the APSME-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. These services are described in the following sub-clauses. The ZDO is fully described in clause 2.5.

2.1.8.1 Discovery Management

Discovery management is provided to the application objects whereby, when queried, the IEEE address or Network Address of the requested device shall be returned (if the device is a ZigBee end device), along with the device addresses of all associated devices (if the device is a ZigBee coordinator or router). This is referred to as device discovery, and is used for the discovery of ZigBee devices.

In addition to device discovery, service discovery is also provided to determine what services are offered on each endpoint, defined in a device, by the respective application objects. A device can discover active endpoints on individual devices or all devices and a device can discover specific services that match a given criteria (profile identifiers and cluster identifiers).

2.1.8.2 Binding Management

Binding management is provided to the application objects in order to bind application objects on ZigBee devices to each other for clear and concise connections through all layers of the protocol stack and though the various connections provided by the ZigBee network nodes. Binding tables are constructed and populated according to the binding calls and results. End device bind, bind and unbind commands between devices is supported via the ZigBee device profile.

2.1.8.3 Security Management

Security management is provided to the application objects for enabling or disabling the security portion of the system. If enabled, key management is performed for master keys, network keys, and the means to establish a link key. Primitives are defined in Chapter 4 to permit key establishment, key transport and authentication.

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 the ZDO and the 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:** This is the ability to match two devices together based on their services and their needs. Once two devices are bound, the APSDE shall be able to transfer a message received from one bound device over to the second device.
- **Group Address Filtering:** This provides the ability to filter group addressed messages based on whether the device is a member of the group or not.
- **Reliable transport:** This 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.

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:

- **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:** This provides 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 2.3 depicts the components and interfaces of the APS sub-layer.

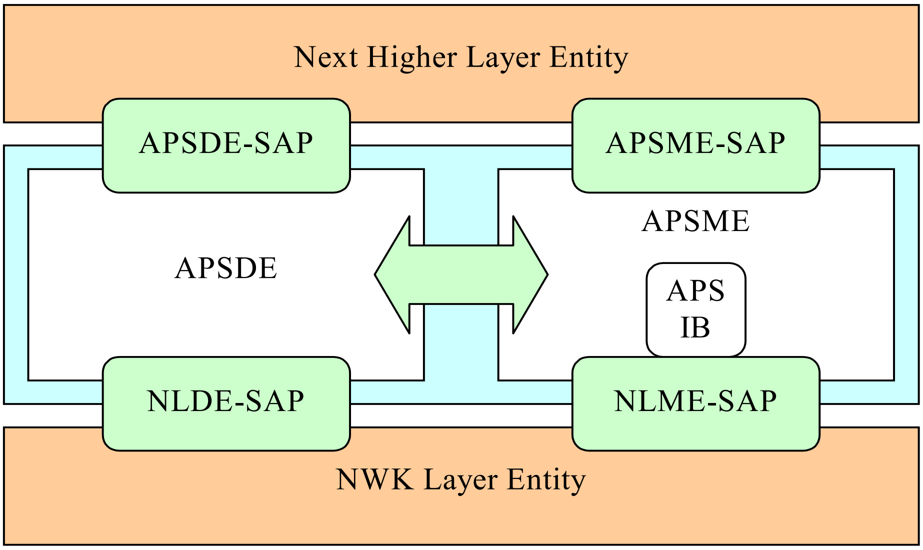


Figure 2.3 The APS Sub-Layer Reference Model

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 though 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 NLME-SAP interfaces (see sub-clause 3.2). 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 subclauses.

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 a single peer NHLE entity.

2.2.4.1.1.1 Semantics of the Service Primitive

This semantics of this primitive are as follows:

```
APSDE-DATA.request                  {
                                    DstAddrMode,
                                    DstAddress,
                                    DstEndpoint,
                                    ProfileId,
                                    ClusterId,
                                    SrcEndpoint,
                                    asduLength,
                                    asdu,
                                    TxOptions,
                                    RadiusCounter
                                    }
```

Table 2.2 specifies the parameters for the APSDE-DATA.request primitive.

Table 2.2 APSDE-DATA.request Parameters

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x00 – 0xff	The addressing mode for the source 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 and 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 DstAddrMode 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 either the number of the individual endpoint or the broadcast endpoint (0xff) of the entity to which the ASDU is being transferred
ProfileId	Integer	0x0000 – 0xffff	The identifier of the profile for which this frame is intended
ClusterId	Integer	0x0000 – 0xffff	The identifier of the object to use in the binding operation if the frame is to be sent using indirect addressing; If indirect addressing is not being used, this parameter is ignored
SrcEndpoint	Integer	0x00 – 0xfe	The individual endpoint of the entity from which the ASDU is being transferred
asduLength	Integer		The number of octets comprising the ASDU to be transferred

Table 2.2 APSDE-DATA.request Parameters

Name	Type	Valid Range	Description
Asdu	Set of octets	-	The set of octets comprising the ASDU to be transferred
TxOptions	Bitmap	0000 0xxx (Where x can be 0 or 1)	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
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 a peer NHLE.

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, the DstAddress and DstEndpoint parameters are ignored and the value of the DstEndpoint parameter is not placed in the resulting APDU; this option allows indirect addressing to be used. If the DstAddrMode parameter is set to 0x02, 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 *apsAddressMap* attribute of the APS IB (see Table 2.23). 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. 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 subfield 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 using the RadiusCounter parameter. If the RadiusCounter parameter is set to 0x00, the network layer will transmit the frame with no restriction in the network. If the RadiusCounter parameter is not set to zero, the network layer will allow the frame transmission to exist in the network for at most that many hops.

If the APDU to be transmitted using direct addressing, the APSDE transmits the constructed frame by issuing the NLDE-DATA.request primitive to the NWK layer. On receipt of the NLDE-DATA.confirm primitive, the APSDE issues the APSDE-DATA.confirm primitive (see sub-clause 2.2.4.1.2) with a status equal to that received from the NWK layer.

If the APDU to be transmitted using indirect addressing (the use of indirect addressing is specified in the delivery mode sub-field) 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, the APSDE constructs the APDU with the endpoint information from the binding table entry, if present, and transmits the frame by issuing the NLDE-DATA.request primitive to the NWK layer using the address information from the binding table entry. On receipt of the corresponding NLDE-DATA.confirm, the APSDE constructs and transmits the APDU for the next binding table entry, as described above; until no more binding table entries remain. On receipt of the initial request, the APSDE issues the APSDE-DATA.confirm primitive with a status of SUCCESS to the originator indicating that the message will be reflected to each binding table entry corresponding to the address of this device and the specified endpoint and cluster identifiers.

If the APDU to be transmitted using indirect addressing and this primitive was received by the APSDE of a device that does not support a binding table, the APSDE constructs the APDU, without a destination endpoint field, and issues the NLDE-DATA.request primitive to the NWK layer using the address of the ZigBee coordinator. On receipt of the NLDE-DATA.confirm primitive, the APSDE issues the APSDE-DATA.confirm primitive with a status equal to that received from the NWK layer.

If the DstAddrMode parameter has a value of 0x01, indicating group addressing, then the frame will be transmitted as a broadcast. A value of 0xffff, that is, the broadcast to all devices address, will be supplied for the DstAddr parameter of the NLDE-DATA.request that is used to transmit the frame.

If the TxOptions parameter specifies that secured transmission is required, the APS sub-layer shall use the security service provider (see sub-clause 4.2.4) to secure the ASDU. If the security processing fails, the APSDE shall issue the APSDE-DATA.confirm primitive with a status of SECURITY_FAIL.

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 then the ASDU is not transmitted and the APSDE shall issue the APSDE-DATA.confirm primitive with a status of INVALID_REQUEST.

2.2.4.1.2 APSDE-DATA.confirm

This primitive reports the results of a request to transfer a data PDU (ASDU) from a local NHLE to a single peer NHLE.

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
                            }
```

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 source 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 and 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 DstAddrMode 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
Status	Enumeration	SUCCESS, NO_SHORT_ADDRESS , NO_BOUND_DEVICE, SECURITY_FAIL, NO_ACK or any status values returned from the NLDE-DATA.confirm primitive	The status of the corresponding request

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 or SECURITY_FAIL or any status values returned from the NLDE-DATA.confirm primitive. The reasons for these status values are fully described in 2.2.4.1.2.

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

This semantics of this primitive are as follows:

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

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 and DstEndpoint not present 0x02 = 16-bit address for DstAddress and DstEndpoint present 0x03 – 0xff = reserved
DstAddress	Address	As specified by the DstAddrMode parameter	The individual device address or group address to which the ASDU is directed
DstEndpoint	Integer	0x00 – 0xff	The target endpoint on the local entity from which the ASDU has been received
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 = SrcAddress and SrcEndpoint not present 0x01 = reserved 0x02 = 16-bit short address for SrcAddress and SrcEndpoint present 0x03 = 64-bit extended address for SrcAddress and SrcEndpoint present 0x04 – 0xff = reserved
SrcAddress	Address	As specified by the SrcAddrMode parameter	The individual device address or group address of the entity from which the ASDU has been received
SrcEndpoint	Integer	0x00 – 0xfe	This parameter shall be present if, and only if, the SrcAddrMode parameter has a value of 0x02 or 0x03 and, if present, shall be 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

Table 2.4 APSDE-DATA.indication Parameters

Name	Type	Valid Range	Description
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
WasBroadcast	Boolean	TRUE or FALSE	TRUE if the transmission was a broadcast, FALSE otherwise
SecurityStatus	Enumeration	UNSECURED, SECURED_NWK_KEY, SECURED_LINK_KEY	UNSECURED if the ASDU was received without any security 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

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. If the frame control field of the ASDU header indicates that the frame is secured, security processing shall be done as specified in sub-clause 4.2.4.

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 sub-clause 4.2.4.

If the received frame is received using indirect addressing and the APDU does not contain a source endpoint, the APSDE issues this primitive with the SrcAddrMode parameter set to 0x00. If the received frame is not received using indirect addressing, the received source address must be mapped to its corresponding extended 64-bit IEEE address by using the *apsAddressMap* attribute of the APS ID (see Table 2.23.) If a corresponding 64-bit IEEE address could be found, the APSDE issues this primitive with the SrcAddrMode parameter set to 0x02 and the SrcAddress parameter set to the corresponding 64-bit IEEE address. If a corresponding 64-bit IEEE address could not be found, the APSDE issues this primitive with the SrcAddrMode parameter set to 0x01, 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 sub-clauses 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-GET	2.2.4.4.1		2.2.4.4.2	
APSME-SET	2.2.4.4.3		2.2.4.4.4	
APSME-UNBIND	2.2.4.3.3		2.2.4.3.4	
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 the ZigBee coordinator, a device supporting a binding table cache or a device that wishes to store source bindings, may process these primitives. If any other device receives these primitives from their next higher layer, the primitives should be ignored.

2.2.4.3.1 APSME-BIND.request

This primitive allows the next higher layer to request to bind two devices together 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 – 0xff	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 source 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, 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 directly or by proxy.

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_DEVICE, 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 – 0xff	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 source 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

2.2.4.3.2.2 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 ILLEGAL_DEVICE, 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 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 – 0xff	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

Table 2.8 APSME-UNBIND.request Parameters

Name	Type	Valid Range	Description
DstAddrMode	Integer	0x00 – 0xff	The addressing mode for the source 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, 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 sub-clause 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. If the devices do not exist on the network, the APSME issues the APSME-BIND.confirm primitive with the Status parameter set to ILLEGAL_DEVICE.

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 directly or by proxy.

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	ValidRange	Description
Status	Enumeration	SUCCESS, ILLEGAL_DEVICE, 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 – 0xff	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

Table 2.9 APSME-UNBIND.confirm Parameters

Name	Type	ValidRange	Description
DstAddrMode	Integer	0x00 – 0xff	The addressing mode for the source 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 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_DEVICE, 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 2.23	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 2.23	The identifier of the AIB attribute that was read
AIBAttributeLength	Integer	0x0000 - 0xffff	The length, in octets, of the attribute value being returned
AIBAttributeValue	Various	Attribute Specific (see Table 2.23)	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 sub-clause 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 2.23.	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 2.23).	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 out of 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 2.23.	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 fully described in sub-clause 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 group 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 - 0xfff7	The 16-bit address of the group being added
Endpoint	Integer	0x01 - 0xf0	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 out of 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 of 0x00 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, the APSME will issue 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 - 0xfff7	The 16-bit address of the group being added
Endpoint	Integer	0x01 - 0xf0	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 APSME-ADD-GROUP.request primitive had an invalid value then the Status value will be set to INVALID_PARAMETER. If the APSME 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 - 0xfff7	The 16-bit address of the group being removed
Endpoint	Integer	0x01 - 0xf0	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 out of 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 of 0x00 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 and the APSME will issue 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 SUCCESS.

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 or INVALID_PARAMETER	The status of the request to remove a group.
GroupAddress	16-bit group address	0x0000 - 0xfff7	The 16-bit address of the group being removed
Endpoint	Integer	0x01 - 0xf0	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 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 - 0xf0	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 of 0x00 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 from the group table, related to this endpoint, and will issue 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, INVALID_PARAMETER or TABLE_FULL	The status of the request to remove all group
Endpoint	Integer	0x01 - 0xf0	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 subclause 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 subclause are depicted in the order in which they are transmitted by the NWK layer, from left to right, where the left-most bit is transmitted first in time. Bits within each field are numbered from 0 (left-most 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 the order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.

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 2.4.

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

Figure 2.4 General APS Frame Format

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 2.5.

Bits: 0-1	**2-3**	**4**	**5**	**6**	**7**
Frame type	Delivery mode	Indirect address mode	Security	Ack. request	Reserved

Figure 2.5 Format of the Frame Control Field

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	Reserved

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	Indirect addressing
10	Broadcast
11	Group addressing

If the value is 0b01, indirect addressing is in use and either the destination or source endpoint shall be omitted, depending on the value of the indirect address mode sub-field. If the value is 0b10, the message is a broadcast. In this case the message will go to all devices and all endpoints defined for the selected broadcast address in use as defined in Section 3.7.5[1]. 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.

1. CCB #584

2.2.5.1.1.3 Indirect Address Mode Sub-field

The indirect address mode sub-field is one bit in length and specifies whether the source or destination endpoint fields are present in the frame when the delivery mode sub-field is set to indicate indirect addressing. If this sub-field is set to 1, the destination endpoint field shall be omitted from the frame, indicating an indirect transmission to the ZigBee coordinator. If this sub-field is set to 0, the source endpoint field shall be omitted from the frame, indicating an indirect transmission from the ZigBee coordinator. If the delivery mode sub-field of the frame control field does not indicate indirect addressing, the indirect address mode sub-field shall be ignored.

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 recipient 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 after determining that the frame is valid.

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 field shall be included in the frame only if the delivery mode sub-field of the frame control field is set to 0b00 (normal unicast delivery) or if the delivery mode sub-field is set to 0b01 (indirect addressing) and the indirect address mode sub-field of the frame control field is set to 0.

A destination endpoint value of 0x00 addresses the frame to the ZigBee device object (ZDO), resident in each device. A destination endpoint value of 0x01-0xf0 addresses the frame to an application operating on that endpoint. A destination endpoint value of 0xff addresses the frame to all active endpoints. All other endpoints (0xf1-0xfe) are reserved.

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 subfield 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.

2.2.5.1.4 Cluster Identifier Field

The cluster identifier field is 16-bits in length and specifies the identifier of the cluster that is to be used in the binding operation on the ZigBee coordinator or on the device indicated by the SrcAddr of the request. The frame type sub-field of the frame control field specifies whether the cluster identifier field is present or not. It will be for data frames, but not for command frames.

2.2.5.1.5 Profile Identifier Field

The profile identifier is 2 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 8-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 0x01-0xf0 indicates that the frame originated from an application operating on that endpoint. All other endpoints (0xf1-0xfe) are reserved.

If the delivery mode sub-field of the frame control field indicates a delivery mode of indirect addressing and the indirect address mode sub-field is set to 0, this field shall not be included in the frame .

2.2.5.1.7 APS Counter

This field is 8 bits in length and is used as described in sub-clause 2.2.8.3.2 to prevent the reception of duplicate frames. This value shall be incremented by one for each new transmission.

2.2.5.1.8 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 subclauses.

2.2.5.2.1 Data Frame Format

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

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

Figure 2.6 Data Frame Format

The order of the fields of the data frame shall conform to the order of the general APS frame as illustrated in Figure 2.4.

2.2.5.2.1.1 Data Frame APS Header Field

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 field shall be included in a data frame according to the value of the delivery mode sub-field 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. The source endpoint present sub-field shall be set to 1. 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 the sequence of octets that has been received by the APS data service and that is to be reflected to the destination devices or delivered to the next higher layer if the coordinator is one of the destinations.

2.2.5.2.2 APS Command Frame Format

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

Octets: 1	**0/2**	**1**	**1**	**Variable**
Frame control	Group Address	APS counter	APS command identifier	APS command payload
APS header			APS payload	

Figure 2.7 APS Command Frame Format.

The order of the fields of the APS command frame shall conform to the order of the general APS frame as illustrated in Figure 2.7.

2.2.5.2.2.1 APS Command Frame APS Header Field

The APS header field for an APS command frame shall contain the frame control and APS counter fields. The group address field shall be included in the frame if the delivery mode sub-field of the frame control field is set to indicate group addressing.

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.

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

Figure 2.8 Acknowledgement Frame Format

The order of the fields of the acknowledgement frame shall conform to the order of the general APS frame as illustrated in Figure 2.8.

2.2.5.2.3.1 Acknowledgement Frame APS Header Field

The APS header field for an acknowledgement frame shall contain the frame control, cluster identifier, profile identifier and APS counter fields. If the delivery mode indicates direct addressing both the source and destination endpoint fields shall be included in an acknowledgement frame. If the delivery mode indicates indirect addressing the source and destination endpoint fields shall be included in an acknowledgement frame according to the value of the indirect address mode 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. All other sub-fields shall be set appropriately according to the intended use of the acknowledgement frame.

Where present, the source endpoint field shall reflect the value in the destination endpoint field of the frame that is being acknowledged. Similarly, where present, the destination endpoint field shall reflect the value in the source endpoint field of the frame that is being acknowledged.

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

2.2.6 Command Frames

This specification defines no command frames. Refer to sub-clause 4.5.8 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 2.22.

Table 2.22 APS Sub-layer Constants

Constant	Description	Value
apscMaxAddrMapEntries	The maximum number of Address Map entries	1 (minimum value) Implementation-specific (maximum value)
apscMaxDescriptorSize	The maximum number of octets contained in a non-complex descriptor	64
apscMaxDiscoverySize	The maximum number of octets that can be returned through the discovery process	64
apscMaxFrameRetries	The maximum number of retries allowed after a transmission failure.	3
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

2.2.7.2 APS Information Base

The APS information base comprises the attributes required to manage the APS layer of a device. The autarkies of the AIB are listed in Table 2.23. The AIB also comprises of some additional attributes that are required to manage the security service provider. These attributes are listed in sub-clause 4.5.9.

Table 2.23 APS IB Attributes

Attribute	Identifier	Type	Range	Description	Default
apsAddressMap	0xc0	Set	Variable	The current set of 64 bit IEEE to 16 bit NWK address maps (see Table 2.24)	Null set
apsBindingTable	0xc1	Set	Variable	The current set of binding table entries in the device (see sub-clause 2.2.8.1.1)	Null set
apsGroupTable	0x0c2	Set	Variable	The current set of group table entries (see Sub-clause 2.2.8.2)	Null set

Table 2.24 Address Map

Entry Number	64 bit IEEE address	16 bit NWK Address
0x00 - *apscMaxAddrMapEntries*	A valid 64-bit IEEE address	0x0000 – 0xffff

2.2.8 Functional Description

2.2.8.1 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. This table is employed by the indirect addressing mechanism.

2.2.8.1.1 Binding Table Implementation

A ZigBee coordinator or 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 ith destination address or destination group address associated with the binding link
e_{di}	= the ith optional destination endpoint identifier associated with the binding link. Note that e_{di} will only be present when a_{di} is a device address

2.2.8.1.2 Binding

The APSME-BIND.request or APSME-UNBIND.request primitive initiates the procedure for creating or removing a binding link. Only the ZigBee coordinator, a device supporting a binding table cache, or a device that wishes to store source bindings, shall initiate this procedure. This is achieved by issuing 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. The 64-bit IEEE address shall be mapped to the 16-bit NWK address using the *apsAddressMap* attribute of the APS IB (see Table 2.23.) 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. This is achieved by issuing 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 was 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 was 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 direct binding attempt and the success of the procedure. This is achieved by issuing the APSME-BIND.confirm primitive with the binding results and the status parameter set to SUCCESS.

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

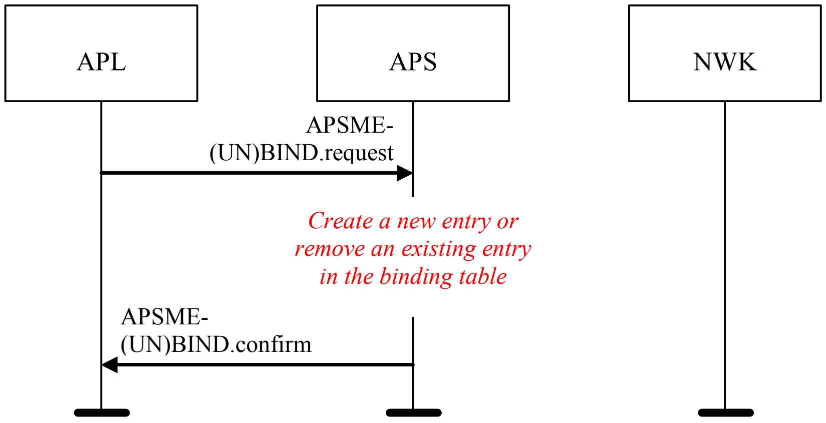

APL
APS
NWK
APSME-(UN)BIND.request
Create a new entry or remove an existing entry in the binding table
APSME-(UN)BIND.confirm

Figure 2.9 Direct Binding on a Device Supporting a Binding Table

2.2.8.2 Group Addressing

The APS 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.

2.2.8.2.1 The Group Table

The group table shall be viewed as a set of associations between groups and endpoints as follows:

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

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 to them as long as it represents the associations just described.

2.2.8.3 Transmission, Reception and Acknowledgement

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

2.2.8.3.1 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 sub-clause 2.2.5.1 and transmitted using the NWK layer data service

Transmissions may be direct, indirect or group addressed. Direct transmissions shall include both destination and source endpoint fields. In this case, the delivery mode sub-field of the frame control field shall be set to either 0b00 (Normal Unicast) or 0b10 (Broadcast). 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.

The APS layer of the device originating an indirect transmission where the binding table entry is stored at the ZigBee coordinator shall direct the transmission to the ZigBee coordinator, which shall handle the task of message reflection.

Indirect transmissions (i.e., those in which the delivery mode sub-field is set to 0b01) shall include only either the source endpoint field or the destination endpoint field, depending on the direction of transmission with respect to the ZigBee coordinator. If the indirect transmission is directed to the ZigBee coordinator, the indirect address mode sub-field shall be set to 1 and the destination endpoint field shall be omitted from the frame. Conversely, if the indirect transmission is directed from the ZigBee coordinator after message reflection, the indirect address mode sub-field shall be set to 0 and the source endpoint field shall be omitted from the frame.

For all devices where the binding table is stored on the source device, the APS layer of the source device originating the transmission shall employ direct transmission to the destination addresses indicated by the corresponding binding table entries in the case where the binding table entry contains a unicast destination device address. In this case, 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, the delivery mode subfield of the frame control field shall have a value of 0b11 and the destination group address shall be placed in the APS header in place of the destination endpoint. The frame shall then be broadcast using the NLDE-DATA.request primitive and employing a broadcast address of 0xffff.

The destination endpoint field, if present, shall contain the endpoint of the intended recipient of the APDU. The source endpoint field, if present, shall contain the endpoint of the originator of the APDU.

If security is required, the frame shall be processed as described in clause 4.5.

When the frame is constructed and ready for transmission, it shall be passed to the NWK data service with a suitable destination and source address. 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.

2.2.8.3.2 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 of a device receives a transmission, it shall pass it directly to the NHLE, unless it needs to be reflected, it is part of an incomplete fragmented transmission, reserved bits are set in the APDSDU frame[2], or it is determined to have been a duplicate of a frame that has been passed up previously. The APSDE shall maintain a duplicate rejection table to include at least the source address, APS counter values 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. In exceptional cases this may result in messages incorrectly being rejected as duplicates.

If reserved bits are set in the APSDU frame, the APS sub-layer shall drop the frame.[3]

If the APSDE receives a secured frame, it shall process the frame as described in clause 4.5 to remove the security.

If the APSDE receives a frame containing both destination and source endpoints, it shall be assumed to be a direct transmission. In this case, the APSDE shall pass it directly to the NHLE at the destination endpoint supplied. If the destination endpoint is set to the broadcast endpoint (0xff), the APSDE shall present the frame to all non-reserved endpoints (0x01-0xf0) supported by the NHLE.[4]

If the APSDE of the ZigBee coordinator receives a frame containing only the source endpoint with the delivery mode sub-field of the frame control field set to

2. CCB #613
3. CCB #613
4. CCB #548

the indirect addressing value (0x01) it shall be assumed to be an indirect transmission. If the device is not a ZigBee coordinator, the frame shall be discarded if the destination endpoint is not present and the delivery mode sub-field of the frame control field is set to the indirect addressing value (0x01).

If the APSDE of a ZigBee coordinator receives an indirect transmission, it shall search its binding table for an entry that matches the source address communicated from the NWK layer, the cluster identifier included in the received frame and the source endpoint included in the frame. If a match is not found, the frame shall be discarded. If a match is found, the APSDE shall build an APDU for each destination endpoint contained in the matching entry in the binding table. The APSDE shall then transmit each frame using the NWK data service.

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 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.request 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 DstAddress parameter.[5] In either case, it shall search the group table and, for each endpoint associated with the given group address it shall issue the APSDE-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 APSDE-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.

2.2.8.3.3 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 shall be sent with its acknowledgement request sub-field set to 0.

2.2.8.3.3.1 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

5. CCB #624

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.

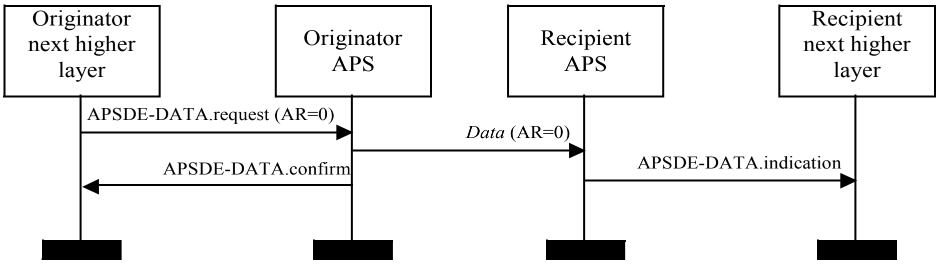

Originator next higher layer
Originator APS
Recipient APS
Recipient next higher layer
APSDE-DATA.request (AR=0)
Data (AR=0)
APSDE-DATA.confirm
APSDE-DATA.indication

Figure 2.10 Successful Data Transmission Without an Acknowledgement

2.2.8.3.3.2 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.

If the original transmission used indirect addressing, the ZigBee coordinator shall send an acknowledgement to the originator then, for each reflected message, shall indicate to the recipient that it requires an acknowledgement by transmitting the data frame with the acknowledgement request sub-field of the frame control field set to 1.

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.

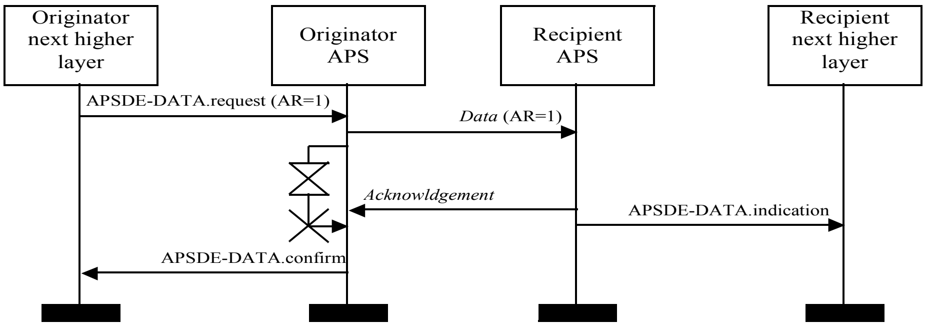


Figure 2.11 Successful Data Transmission With an Acknowledgement

2.2.8.3.4 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 at most *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 the same frame counter as the original frame.

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: private 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 narrow 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 contain 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.

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) and endpoints 0xf1-0xfe are reserved. Consequently, a single physical ZigBee radio can support up to 240 applications on endpoints 0x01-0xf0.

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. 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 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 created with a single endpoint application written for a standard, published ZigBee profile identifier “XX”. If a manufacturer wanted to deploy a ZigBee device supporting the standard profile “XX” and also provide vendor specific extensions, these extensions would be advertised on a separate endpoint. Devices that support the standard profile identifier “XX” but not manufactured with the vendor extensions, would only advertise support for the single profile identifier “XX” and could not respond to or create messages using the vendor extensions.

2.3.1.6 Enabling Backward Compatibility

In the previous example, a device is created using a standard, published ZigBee profile identifier “XX” which contains the initial version of the standard profile. If the ZigBee Alliance were to update this standard profile to create new features and additions, the revisions would be incorporated into a new standard profile with a new profile identifier (say “XY”). Devices manufactured with just profile identifier “XX” would be backward 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, 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.

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 2.25.

Table 2.25 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 (Chapter 1).

The complex descriptor shall be formatted and transmitted as illustrated in Figure 2.12.

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

Figure 2.12 Format of the Complex Descriptor

Each field included in the complex descriptor shall be formatted as illustrated in Figure 2.13.

Octets: 1	Variable
Compressed XML tag	Field data

Figure 2.13 Format of an Individual Complex Descriptor Field

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 descriptor, each formatted as illustrated in Figure .

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 2.37.

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 sub-clause 2.1.5. 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 Composite Devices

A ZigBee node may contain a number of separate subunits, each of which has its own simple descriptor. The mechanism for discovering this is described in ZigBee device profile service discovery section.

2.3.2.4 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 2.26 in their order of transmission.

Table 2.26 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 transfer size	16
Server Mask	16

2.3.2.4.1 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 2.27.

Table 2.27 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

2.3.2.4.2 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.

2.3.2.4.3 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.

2.3.2.4.4 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.

2.3.2.4.5 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 2.28, shall be set to 1. All other bits shall be set to 0.

Table 2.28 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

2.3.2.4.6 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 MAC sub-layer [B1]. The MAC capability flags field shall be formatted as illustrated in Figure 2.14.

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

Figure 2.14 Format of the MAC Capability Flags Field

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 sub-clause 2.3.2.5.)

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 always be set to 1.

2.3.2.4.7 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.

2.3.2.4.8 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, and specifies the maximum size, in octets, of the application support sub-layer data unit (ASDU) 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 (fragmentation is not currently supported).

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

2.3.2.4.9 Maximum Transfer Size Field

The maximum transfer size field of the node descriptor is sixteen bits in length, with a valid range of 0x0000-0x7fff, and specifies the maximum size, in octets, that can be transferred to or from this node in one single message transfer. This value can exceed the value of the node maximum buffer size field (see sub-clause 2.3.2.4.8).

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

2.3.2.4.10 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 inTable 2.29.

Table 2.29 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 – 15	Reserved

2.3.2.5 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 2.30 in the order of their transmission.

Table 2.30 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

2.3.2.5.1 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 2.31.

Table 2.31 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 sub-field of the node descriptor
0001	Receiver comes on periodically as defined by the node power descriptor
0010	Receiver comes on when stimulated, e.g. by a user pressing a button
0011-1111	Reserved

2.3.2.5.2 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 2.32, shall be set to 1. All other bits shall be set to 0.

Table 2.32 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

2.3.2.5.3 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 2.33, shall be set to 1. All other bits shall be set to 0.

Table 2.33 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

2.3.2.5.4 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 2.34.

Table 2.34 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

2.3.2.6 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 2.35 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 *maxCommandSize*.

Table 2.35 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)

2.3.2.6.1 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 1-240.

2.3.2.6.2 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.

2.3.2.6.3 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.

2.3.2.6.4 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 2.36.

Table 2.36 Values of the Application Device Version Field

Application Device Version Value $b_3b_2b_1b_0$	Description
0000	Version 1.0
0001–1111	Reserved

2.3.2.6.5 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.

2.3.2.6.6 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, and specifies the list of input clusters supported on this endpoint, used during the binding procedure.

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

2.3.2.6.7 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.

2.3.2.6.8 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, and specifies the list of output clusters supported on this endpoint, used during the binding procedure.

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

2.3.2.7 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 2.37, 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 *maxCommandSize*.

Table 2.37 Fields of the Complex Descriptor

Field Name	XML Tag	Compressed XML Tag Value $b_3b_2b_1b_0$	Data Type
Reserved	-	0000	-
Language and character set	<languageChar>	0001	See sub-clause 2.3.2.7.1
Manufacturer name	<manufacturerName>	0010	Character string
Model name	<modelName>	0011	Character string
Serial number	<serialNumber>	0100	Character string
Device URL	<deviceURL>	0101	Character string
Icon	<icon>	0110	Octet string
Icon URL	<outliner>	0111	Character string
Reserved	-	1000 – 1111	-

2.3.2.7.1 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 2.15.

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

Figure 2.15 Format of the Language and Character Set Field

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 2.38.

Table 2.38 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.

2.3.2.7.2 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.

2.3.2.7.3 Model Name Field

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

2.3.2.7.4 Serial Number Field

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

2.3.2.7.5 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.

2.3.2.7.6 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.

2.3.2.7.7 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.

2.3.2.8 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,[6] and shall contain a maximum of 16 characters.

The fields of the user descriptor are shown in Table 2.39 in the order of their transmission.

Table 2.39 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, the application framework shall determine whether the specified profile identifier matches the identifier of the profile implemented on the specified endpoint. If the profile identifier does not match, the application framework shall reject the frame. Otherwise, the application framework shall pass the payload of the received frame to the application implemented on the specified endpoint.

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

6. CCB #604

implemented within ZigBee Device Objects. The ZigBee Device Profile operates like any ZigBee profile by defining Device Descriptions and Clusters. Unlike application specific profiles, the Device Descriptions and 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

2.4.2.1 Device and Service Discovery Overview

Device Discovery is a distributed operation where individual devices or designated discovery cache devices respond to discovery requests. The “device address of interest” field enables response from either the device itself or a discovery cache device. In selected cases where the discovery cache device *and* the “device address of interest” device also responds, the response from the “device address of interest” shall be used.

Service Discovery is a distributed operation 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 cases, where *both* the discovery cache device *and* the “device address of interest” devices 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 match 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 Devices shall respond only with its address. A ZigBee Coordinator or Router shall reply with its own address and the addresses of each associated child devices. Inclusion of the associated child devices permit 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 match 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 may be broadcast or 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 match the supplied criteria. This command may be broadcast to all RxOnWhenIdle devices 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 "simple 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.
  - 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 map 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 bind tables:

- Registering devices that implement source binding (where source binding is defined to be the ability for a device that is a client and, supplying requests, holds a record of the server destination that are the desired targets of the recipients) :
  - 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 bind table:
  - Provides the ability to replace one device for another, by replacing all instances of its address in the binding table
- Backing up a bind 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 (which contains the addresses of any source device registered to support its own binding table):
  - 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 (which contains the addresses of any source device registered to support its own binding table):
  - 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:

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
- Provides the ability to set management information controls including:
  - Network disassociate

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

In neither the client or server roles of the Device Profile shall indirect messaging be used. Specifically, due to the omission of a Simple Descriptor for the Device Profile and due to the role of the Device Profile in discovery itself, the use of binding and indirect messaging on endpoint 0 is prohibited for client or server operations.

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 2.16.

Octets: 1	Variable
Transaction sequence number	Transaction data

Figure 2.16 Format of the ZDP Frame

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 8-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 re-start 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 sub-clauses 2.4.3 and 2.4.4.

2.4.3 Client Services

The Device Profile Client Services supports 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 2.40 lists the commands supported by Device Profile, Device and Service Discovery Client Services. Each of these commands will be discussed in the following subclauses.

Table 2.40 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	O	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
Discovery_Cache_req	O	M
End_Device_annce	O	O
User_Desc_set	O	O
System_Server_Discover _req	O	M
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	M

2.4.3.1.1 NWK_addr_req

The NWK_addr_req command (ClusterID=0x0000) shall be formatted as illustrated in Figure 2.17 .

Octets: 8	1	1
IEEEAddress	RequestType	StartIndex

Figure 2.17 Format of the NWK_addr_req Command

Table 2.41 specifies the fields of the NWK_addr_req Command Frame.

Table 2.41 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 broadcast to all RxOnWhenIdle devices.

For future upgrade ability, the destination addressing may be permitted to be unicast. This would permit directing the message to a well-known destination that supports centralized or agent-based device discovery.

2.4.3.1.1.2 Effect on Receipt

Upon receipt, a Remote Device shall compare the IEEEAddr to its local IEEE address or any IEEE address held in its local discovery cache. If there is no match, the request shall be discarded and no response generated. If a match is detected between the contained IEEEAddr and the Remote Device's IEEE address or one held in the discovery cache, 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 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 of the request and the NumAssocDev field set to 0. If the RequestType is single device response or extended response, the Remote Device shall create a unicast message to the Local Device and include the Remote Device's 16-bit NWK address as the source address and the discovered NWKAddrRemoteDev address along with the matched IEEE Address in the response payload. If the RequestType was Single device response, the response message shall be sent with a SUCCESS status. Otherwise, if the RequestType was Extended response and the Remote Device is either the ZigBee coordinator or router with associated devices, the Remote Device shall first include the matched IEEE Address and NWK Address in the message payload and shall also supply a list of all 16 bit NWK addresses, starting with the entry StartIndex and continuing with whole entries until the maximum APS packet length is reached, for its associated devices along with a status of SUCCESS.

2.4.3.1.2 IEEE_addr_req

The IEEE_addr_req command (ClusterID=0x0001) shall be formatted as illustrated in Figure 2.18.

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

Figure 2.18 Format of the IEEE_addr_req_Command Frame

Table 2.42 specifies the fields of the IEEE_addr_req command frame.

Table 2.42 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 of this command, a remote device shall create a unicast IEEE_addr_rsp command to the source as indicated by the IEEE_addr_req command according to the supplied RequestType. If the RequestType is one of the reserved values, an IEEE_addr_resp command shall be generated 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 field set to 0. Additionally, if the RequestType indicates an Extended Response and the Remote Device is the ZigBee coordinator or router with associated devices, the Remote Device shall first include its own 64-bit IEEE address and shall also supply a list of all 16 bit network addresses for its associated devices, starting with the entry StartIndex and continuing with whole entries until the maximum APS packet length is reached, with a status of SUCCESS.

2.4.3.1.3 Node_Desc_req

The Node_Desc_req_command (ClusterID=0x0002) shall be formatted as illustrated in Figure 2.19.

Octets: 2
NWKAddrOfInterest

Figure 2.19 Format of the Node_Desc_req Command Frame

Table 2.43 specifies the fields for the Node_Desc_req command frame .

Table 2.43 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 2.43. 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 sub-clause 2.4.4.1.3.1.

2.4.3.1.4 Power_Desc_req

The Power_Desc_req command (ClusterID=0x0003) shall be formatted as illustrated in Figure 2.20.

Octets: 2
NWKAddrOfInterest

Figure 2.20 Format of the Power_Desc_req Command Frame

Table 2.44 specifies the fields of the Power_Desc_req command frame.

Table 2.44 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 2.44. 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 sub-clause 2.4.4.1.4.1.

2.4.3.1.5 Simple_Desc_req

The Simple_Desc_req command (ClusterID=0x0004) shall be formatted as illustrated in Figure 2.21.

Octets: 2	1
NWKAddrOfInterest	EndPoint

Figure 2.21 Format of the Simple_Desc_req Command Frame

Table 2.45 specifies the fields of the Simple_Desc_req command frame.

Table 2.45 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	1-240	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 2.45. 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 sub-clause 2.4.4.1.5.1.

2.4.3.1.6 Active_EP_req

The Active_EP_req command (ClusterID=0x0005) shall be formatted as illustrated in Figure 2.22.

Octets: 2
NWKAddrOfInterest

Figure 2.22 Format of the Active_EP_req Command Frame

Table 2.46 specifies the fields of the Active_EP_req command frame.

Table 2.46 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 2.46. 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 sub-clause 2.4.4.1.6.1.

2.4.3.1.7 Match_Desc_req

The Match_Desc_req command (ClusterID=0x0006) shall be formatted as illustrated in Figure 2.23.

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

Figure 2.23 Format of the Match_Desc_req Command Frame

Table 2.47 specifies the fields of the Match_Desc_req command frame.

Table 2.47 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 RxOnWhenIdle devices 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 2.47. The NWKAddrOfInterest field shall contain the broadcast to all RxOnWhenIdle devices network address (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.

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 sub-clause 2.4.4.1.7.1.

2.4.3.1.8 Complex_Desc_req

The Complex_Desc_req command (ClusterID=0x0010) shall be formatted as illustrated in Figure 2.24.

Octets: 2
NWKAddrOfInterest

Figure 2.24 Format of the Complex_Desc_req Command Frame

Table 2.48 specifies the fields of the Complex_Desc_req command frame.

Table 2.48 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 2.48. 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 sub-clause 2.4.4.1.8.1.

2.4.3.1.9 User_Desc_req

The User_Desc_req (ClusterID=0x0011) command shall be formatted as illustrated in Figure 2.25.

Octets: 2
NWKAddrOfInterest

Figure 2.25 Format of the User_Desc_req Command Frame

Table 2.49 specifies the fields of the User_Desc_req command frame.

Table 2.49 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 2.49. 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 sub-clause 2.4.4.1.9.1.

2.4.3.1.10 Discovery_Cache_req

The Discovery_Cache_req command (ClusterID=0x0012) shall be formatted as illustrated in Figure 2.26.

Octets: 2	8
NWKAddr	IEEEAddr

Figure 2.26 Format of the Discovery_Cache_req Command Frame

Table 2.50 specifies the parameters for the Discovery_Cache_req command frame.

Table 2.50 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 RxOnWhenIdle devices.

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 End_Device_annce

The End_Device_annce command (ClusterID=0x0013) shall be formatted as illustrated in Figure 2.27.

Octets: 2	8	1
NWKAddr	IEEEAddr	Capability

Figure 2.27 Format of the End_Device_annce Command Frame

Table 2.51 specifies the fields of the End_Device_annce command frame.

Table 2.51 Fields of the End_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 2.14	Capability of the local device

2.4.3.1.11.1 When Generated

The End_Device_annce is provided to enable ZigBee end devices on the network to notify other ZigBee devices that the end device has joined or re-joined the network, identifying the end devices 64-bit IEEE address and new 16-bit NWKaddress and informing the Remote Devices of the capability of the ZigBee EndDevice. The destination addressing on this primitive is broadcast to all devices.

2.4.3.1.11.2 Effect on Receipt

Upon receipt, the Remote Device (a ZigBee coordinator or source device for the binding operation) shall use the IEEEAddr in the message to find a match with any binding table entries held in the Remote Device. If a match is detected, the Remote Device shall update its APS Information Block Address Map with the updated NWKAddr corresponding to the IEEEAddr received.

2.4.3.1.12 User_Desc_set

The User_Desc_set command (ClusterID=0x0014) shall be formatted as illustrated in Figure 2.28.

Octets: 2	1	Various
NWKAddrOfInterest	Length	UserDescriptor

Figure 2.28 Format of the User_Desc_set Command Frame

Table 2.52 specifies the fields of the User_Desc_set command frame.

Table 2.52 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

2.4.3.1.12.1 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 2.52. 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.

2.4.3.1.12.2 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 sub-clause 2.4.4.1.11.1.

2.4.3.1.13 System_Server_Discovery_req

The System_Server_Discovery_req command (ClusterID=0x0015) shall be formatted as illustrated in Figure 2.29.

Octets: 2
ServerMask

Figure 2.29 Format of the System_Server_Discovery_req Command Frame

Table 2.53 specifies the fields of the System_Server_Discovery_req command frame.

Table 2.53 Fields of the System_Server_Discovery_req Command

Name	Type	Valid Range	Description
ServerMask	Bitmap	16-bits	See Table 2.29 for bit assignments

2.4.3.1.13.1 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 RxOnWhenIdle devices'.

2.4.3.1.13.2 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.

2.4.3.1.14 Discovery_store_req

The Discovery_Store_req command (ClusterID=0x0016) shall be formatted as illustrated in Figure 2.30.

Octets: 2	8	1	1	1	1	Variable
NWKAddr	IEEEAddr	NodeDesc -Size	PowerDesc-Size	ActiveEPSize	SimpleDesc-Count	SimpleDesc-SizeList

Figure 2.30 Format of the Discovery_Store_req Command Frame

Table 2.54 specifies the fields of the Discovery_store_req command frame.

Table 2.54 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
NodeDescSize	Integer	0x00-oxff	Size in bytes of the Node Descriptor for the Local Device

Table 2.54 Fields of the Discovery_store_req Command (Continued)

Name	Type	Valid Range	Description (Continued)
PowerDescSize	Integer	0x00 - 0xff	Size in bytes of the Power Descriptor for the Local Device
ActiveIPSize	Integer	0x00 - 0xff	Size in bytes of the ActiveEPCount and ActiveIPList 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

2.4.3.1.14.1 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.

2.4.3.1.14.2 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, however, 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.

2.4.3.1.15 Node_Desc_store_req

The Node_Desc_store_req command (ClusterID=0x0017) shall be formatted as illustrated in Figure 2.31.

Octets: 2	8	Variable
NWKAddr	IEEEAddr	NodeDescriptor

Figure 2.31 Format of the Node_Desc_store_req Command Frame

Table 2.55 specifies the fields of the Node_Desc_store_req command frame. .

Table 2.55 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 sub-clause 2.3.2.4

2.4.3.1.15.1 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.

2.4.3.1.15.2 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 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.

2.4.3.1.16 Power_Desc_store_req

The Power_Desc_store_req command (ClusterID=0x0018) shall be formatted as illustrated in Figure 2.32.

Octets: 2	8	Variable
NWKAddr	IEEEAddr	PowerDescriptor

Figure 2.32 Format of the Power_Desc_store_req Command Frame

Table 2.56 specifies the fields of the Power_Desc_store_req command frame.

Table 2.56 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
PowerDescriptor	Power Descriptor		See the Power Descriptor format in sub-clause 2.3.2.5; This field shall only be included in the frame if the status field is equal to SUCCESS

2.4.3.1.16.1 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.

2.4.3.1.16.2 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 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.

2.4.3.1.17 Active_EP_store_req

The Active_EP_store_req command (ClusterID=0x0019) shall be formatted as illustrated in Figure 2.33 .

Octets: 2	8	1	Variable
NWKAddr	IEEEAddr	ActiveEPCount	ActiveEPList

Figure 2.33 Format of the Active_EP_store_req Command Frame

Table 2.57 specifies the fields of the Active_EP_store_req command frame

Table 2.57 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
ActiveEPList			List of bytes, each of which represents an 8-bit endpoint number

2.4.3.1.17.1 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.

2.4.3.1.17.2 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 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.

2.4.3.1.18 Simple_Desc_store_req

The Simple_Desc_store_req command (ClusterID=0x001a) shall be formatted as illustrated in Figure 2.34.

Octets: 2	8	1	Variable
NWKAddr	IEEEAddr	Length	SimpleDescriptor

Figure 2.34 Format of the Simple_Desc_store_req Command Frame

Table 2.58 specifies the fields of the Simple_Desc_store_req command frame .

Table 2.58 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
SimpleDescriptor	Simple Descriptor		See the Simple Descriptor format in sub-clause 2.3.2.6

2.4.3.1.18.1 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.

2.4.3.1.18.2 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 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.

2.4.3.1.19 Remove_node_cache_req

The Remove_node_cache_req command (ClusterID=0x001b) shall be formatted as illustrated in Figure 2.35.

Octets: 2	8
NWKAddr	IEEEAddr

Figure 2.35 Format of the Remove_node_cache_req Command Frame

Table 2.59 specifies the fields of the Remove_node_cache_req command frame.

Table 2.59 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

2.4.3.1.19.1 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.

2.4.3.1.19.2 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 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 NOT_PERMITTED. Finally, the Remote Device shall remove all cached discovery information for the device of interest and return SUCCESS to the Local Device.

2.4.3.1.20 Find_node_cache_req

The Find_node_cache_req command (ClusterID=0x001c) shall be formatted as illustrated in Figure 2.36.

Octets: 2	8
NWKAddr	IEEEAddr

Figure 2.36 Format of the Find_node_cache Command Frame

Table 2.60 specifies the fields of the Find_node_cache_req command frame.

Table 2.60 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

2.4.3.1.20.1 When Generated

The Find_node_cache_req is provided to enable ZigBee devices on the network to broadcast to all RxOnWhenIdle devices 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.

2.4.3.1.20.2 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 SUCCESS to the Local Device along with the NWKAddr and IEEEaddr for the device of interest.

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

Table 2.61 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 subclauses.

Table 2.61 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 2.37.

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

Figure 2.37 Format of the End_Device_Bind_req Command Frame

Table 2.62 specifies the fields of the End_Device_Bind_req command frame.

Table 2.62 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.[a]
SrcIEEEAddress	IEEE Address	A valid 64-bit IEEE Address	The IEEE address of the device generating the request
SrcEndpoint	8 bits	1-240	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

Table 2.62 Fields of the End_Device_Bind_req Command (Continued)

Name	Type	Valid Range	Description
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).[b]
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).[c]

a. CCB #587

b. CCB #544

c. CCB #544

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 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.

The ZigBee coordinator shall then determine the 64-bit IEEE address of each local device. If these addresses are not known, the ZigBee coordinator shall determine them by using the IEEE_Addr_req command and corresponding IEEE_Addr_rsp command.

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 2.38.

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

Figure 2.38 Format of the Bind_req Command Frame

Table 2.63 specifies the fields of the Bind_req command frame.

Table 2.63 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	0x01-0xf0	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	0x01-0xf0	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 2.39.

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

Figure 2.39 Format of the Unbind_req Command Frame

Table 2.64 specifies the fields of the Unbind_req command frame.

Table 2.64 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	0x01-0xf0	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.

Table 2.64 Fields of the Unbind_req Command (Continued)

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	0x01-0xf0	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 a 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 SrcAddress is specified and the Binding Manager is unsupported on that remote device, a status of NOT_SUPPORTED shall be returned. If a Binding Table entry for the SrvAddress, 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 2.40.

Octets: 8
NodeAddress

Figure 2.40 Format of the Bind_Register_req Command Frame

Table 2.65 specifies the fields for the Bind_Register_req command frame.

Table 2.65 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 bind 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 2.41.

Octets: 8	1	8	1
OldAddress	OldEndpoint	NewAddress	NewEndpoint

Figure 2.41 Format of the Replace_Device_req Command Frame

Table 2.66 specifies the fields for the Replace_Device_req command frame.

Table 2.66 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	0x01 - 0xf0	The endpoint being replaced
NewAddress	IEEE Address	A valid 64-bit IEEE	The replacement address
NewEndpoint	Integer	0x01 - 0xf0	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 its OldAddress is non-zero and, if so, shall search its binding table for entries of source addresses and source entries, or destination addresses and source addresses, 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 bind 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 2.42.

Figure 2.42 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[a]

a. CCB #601

Table 2.67 specifies the fields of the Store_Bkup_Bind_Entry_req command frame.

Table 2.67 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 - 0xf0	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

Table 2.67 Fields of the Store_Bkup_Bind_Entry_req Command

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[a]
DstAddress	Address	As specified by the DstAddrMode field	The destination address for the binding entry.
DstEndp	Integer	0x01-0xf0	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.

a. CCB #601

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 2.43.

Figure 2.43 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[a]

a. CCB #601

Table 2.67 specifies the fields of the Remove_Bkup_Bind_Entry_req command frame.

Table 2.68 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 - 0xf0	The IEEE address 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[a]
DstAddress	Address	As specified by the DstAddrMode field	The destination address for the binding entry.
DstEndp	Integer	0x01-0xf0	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.

a. CCB #601

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 2.44.

Octets: 2	2	2	Variable
BindingTableEntries	StartIndex	BindingTableListCount	BindingTableList

Figure 2.44 Format of the Backup_Bind_Table_req Command Frame

Table 2.69 specifies the fields of the Backup_Bind_Table_req command frame.

Table 2.69 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
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 2.122 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 when 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 2.45.

Octets: 2
StartIndex

Figure 2.45 Fields of the Recover_Bind_Table_req Command Frame

Table 2.70 specifies the fields of the Recover_Bind_Table_req command frame.

Table 2.70 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 2.46.

Octets: 2	2	2	Variable
SourceTableEntries	StartIndex	SourceTableListCount	SourceTableList

Figure 2.46 Fields of the Backup_Source_Bind_req Command Frame

Table 2.71 specifies the fields of the Backup_Source_Bind_req command frame.

Table 2.71 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 2.47.

Octets: 2
StartIndex

Figure 2.47 Format of the Recover_Source_Bind_req Command Frame

Table 2.72 specifies the fields of the Recover_Source_Bind_req command frame.

Table 2.72 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 bind 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 bind 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 2.73 lists the commands supported by Device Profile: Network Management Client Services. Each of these primitives will be discussed in the following subclauses.

Table 2.73 Network Management Client Services Commands

Network Management Client Services	Client Transmission	Server Processing
Mgmt_NWK_Disc_req	O	O
Mgmt_Lqi_req	O	O

Table 2.73 Network Management Client Services Commands

Mgmt_Rtg_req	O	O
Mgmt_Bind_req	O	O
Mgmt_Leave_req	O	O
Mgmt_Direct_Join_req	O	O
Mgmt_Permit_Joining_req	O	M
Mgmt_Cache_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 2.48.

Octets: 4	1	1
ScanChannels	ScanDuration	StartIndex

Figure 2.48 Format of the Mgmt_NWK_Disc_req Command Frame

Table 2.74 specifies the fields for the Mgmt_NWK_Disc_req command frame.

Table 2.74 Fields of the Mgmt_NWK_Disc_req Command

Name	Type	Valid Range	Description
ScanChannels	Bitmap	32 bit field	See (sub-clause 3.3.2.1) 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 * (2n + 1)) symbols, where n is the value of the ScanDuration parameter. For more information on MAC sub-layer scanning (see [B1].
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 2.49.

Octets: 1
StartIndex

Figure 2.49 Format of the Mgmt_Lqi_req Command Frame

Table 2.75 specifies the fields for the Mgmt_NWK_Disc_req command frame.

Table 2.75 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 and the destination address must be that of a ZigBee Coordinator or ZigBee Router.

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.

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 2.50.

Octets: 1
StartIndex

Figure 2.50 Format of the Mgmt_Rtg_req Command Frame

Table 2.76 specifies the fields for the Mgmt_Rtg_req command frame.

Table 2.76 Fields of the Mgmt_Rtg_req Command

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

2.4.3.3.3.1 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 2.51.

Octets: 1
StartIndex

Figure 2.51 Format of the Mgmt_Bind_req Command Frame

Table 2.77 specifies the fields for the Mgmt_Bind_req command frame.

Table 2.77 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 (ZigBee Coordinator or ZigBee Router) 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 2.52.

Bits: 64	**6**	**1**	**1**
Device Address	Reserved	Remove Children	Rejoin

Figure 2.52 Format of the Mgmt_Leave_req Command Frame

Table 2.78 specifies the fields for the Mgmt_Leave_req command frame.

Table 2.78 Fields of the Mgmt_Leave_req Command

Name	Type	Valid Range	Description
DeviceAddress	Device Address	An extended 64 bit, IEEE address	See (sub-clause 3.3.8.1) for details on the DeviceAddress parameter within NLME-LEAVE.request
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 issue the NLME-LEAVE.request primitive using the parameters supplied with the Mgmt_Leave_req command. The results of the leave attempt shall be reported back to the local device via the Mgmt_Leave_rsp command.

If the remote device does not support this optional management request, it shall return a status of NOT_SUPPORTED. 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 2.53.

Octets: 8	1
Device Address	Capability Information

Figure 2.53 Format of the Mgmt_Direct_Join _reqCommand Frame

Table 2.79 specifies the fields for the Mgmt_Direct_Join_req command frame.

Table 2.79 Fields of the Mgmt_Direct_Join_req Command

Name	Type	Valid Range	Description
DeviceAddress	Device Address	An extended 64 bit, IEEE address	See sub-clause 3.3.6.1 for details on the DeviceAddress parameter within NLME-JOIN.request
CapabilityInformation	Bitmap	See Table 3.17	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 2.54.

Octets: 1	**1**
PermitDuration	TC_Significance

Figure 2.54 Format of the Mgmt_Permit_Joining_req Command Frame

Table 2.80 specifies the fields of the Mgmt_Permit_Joining_req command frame.

Table 2.80 Fields of the Mgmt_Permit_Joining_req Command

Name	Type	Valid Range	Description
PermitDuration	Integer	0x00 - 0xff	See sub-clause 3.3.4.1 for details on the PermitDuration parameter within NLME-PERMIT-JOINING.request
TC_Significance	Boolean Integer	0x00 - 0x01	If this is set to 0x01 and the remote device is the trust center, the command affects the trust center authentication policy as described in the sub-clauses below; If this is set to 0x00, there is no effect on the trust center

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 RxOnWhenIdle devices'.

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; if set to 0xFF joining is permitted indefinitely or until another Mgmt_Permit_Joining_req is received. If a second Mgmt_Permit_Joining_req is received whilst the previous one is still counting down it will supersede the previous request. Additionally, if the remote device is the Trust Center and TC_Significance is set to 1, the trust center authentication policy will be affected. In this case, if PermitDuration is set to a non-zero value, trust center authentication is allowed so that when the trust center receives an NLME-JOIN.indication or an APSME-UPDATE-DEVICE.indication indicating that a new device has joined, it will authenticate it and issue a security key as appropriate. Alternatively, if PermitDuration is set to zero, trust center authentication will be disallowed with the effect that if an APSME-UPDATE-DEVICE.indication is received indicating that a new device has joined; The trust center shall then issue an APSME-REMOVE-DEVICE.request to refuse the new device. Note that the TC_Significance flag and the Trust Center action will be subject to the security policy imposed by the stack profile. Particularly the Trust Center may be configured not to act on this flag unless it has been sent by particular trusted devices such as configuration tools which are known to the Trust Center. Similarly, the main command features may be disallowed under some stack profiles unless the sending device can be authenticated as a known configuration device. If the command is not permitted, a response of INVALID_REQUEST shall be sent (unless the command was a broadcast to all RxOnWhenIdle devices in which case no response shall be sent) 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.

2.4.3.3.8 Mgmt_Cache_req

The Mgmt_Cache_req command (ClusterID=0x0037) shall be formatted as illustrated in Figure 2.55.

Octets: 1
StartIndex

Figure 2.55 Fields of the Mgmt_Cache_req Command Frame

Table 2.81 specifies the fields of the Mgmt_Cache_req command frame.

Table 2.81 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.

2.4.4 Server Services

The Device Profile Server Services supports 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.

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.[7]

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

7. CCB #530

status field shall be set to NOT_SUPPORTED. The resulting response shall be unicast to the requesting client.[8]

2.4.4.1 Device and Service Discovery Server

Table 2.82 lists the commands supported by the Device and Service Discovery Server Services device profile. Each of these commands will be discussed in the following subclauses. For receipt of the End_Device_Annce command, the server shall check all internal references to the IEEE address supplied in the request. For all references to the IEEE address in the Local Device, the corresponding NWK address supplied in the End_Device_Annce shall be substituted. The server shall not supply a response to the End_Device_Annce.

Table 2.82 Device and Service Discovery Server Service Primitives

Device and Service Discovery Server Services	Server Processing
NWK_addr_rsp	M
IEEE_addr_rsp	M
Node_Desc_rsp	M
Power_Desc_rsp	M
Simple_Desc_rsp	M
Active_EP_rsp	M
Match_Desc_rsp	M
Complex_Desc_rsp	O
User_Desc_rsp	O
User_Desc_conf	O
System_Server_Discovery_rsp	M
Discovery_store_rsp	O
Node_Desc_store_rsp	O
Power_Desc_store_rsp	O
Active_EP_store_rsp	O
Simple_Desc_store_rsp	O
Remove_node_cache_rsp	O
Find_node_cache_rsp	O

8. CCB #585

2.4.4.1.1 NWK_addr_rsp

The NWK_addr_rsp command (ClusterID=0x8000) shall be formatted as illustrated in Figure 2.56.

Octets: 1	8	2	1	1	Variable
Status	IEEEAddr RemoteDev	NWKAddr RemoteDev	Num AssocDev	StartIndex	NWKAddr AssocDevList

Figure 2.56 Format of the NWK_addr_rsp Command Frame

Table 2.83 specifies the fields of the NWK_addr_rsp command frame.

Table 2.83 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
NumAssocDev	Integer	0x00-0xff	Count of the number of associated devices to the Remote Device and 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 the RequestType in the request is for a Single Device Response, this field shall not be included in the frame

Table 2.83 Fields of the NWK_addr_rsp Command (Continued)

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 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 0x-ffff	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 the RequestType in the request is for a Single Device Response, this field shall not be included in the frame

2.4.4.1.1.1 When Generated

The NWK_addr_rsp is generated from a Remote Device receiving a broadcast NWK_addr_req who detect a match of the IEEEAddr parameter with their own IEEE address or the IEEE address held in a local discovery cache. The destination addressing on this command is unicast.

2.4.4.1.1.2 Effect on Receipt

Upon receipt, a Remote Device shall attempt to match the IEEEAddr in the NWK_addr_req command with the Remote Device's IEEE address. If no match exists, the request shall be discarded and no response message processing performed. If a match is detected, the Remote Device shall create a unicast message to the source indicated by the NWK_addr_req command. Included in the NWK_addr_rsp payload is the IEEE address that matched from the NWK_addr_req and the NWK address of the Remote Device. If the RequestType is one of the reserved values, the Status field shall be set to INV_REQUESTTYPE and the NumAssocDev, StartIndex and NWKAddrAssocDevList fields shall not be included in the frame. If the RequestType was set to single response, the NumAssocDev, StartIndex and NWKAddrAssocDevList fields shall not be included in the frame. If the RequestType was set to extended response and the Remote Device is not the ZigBee coordinator or router, the NumAssocDev field shall be set to 0 and the StartIndex and NWKAddrAssocDevList fields shall not be included in the frame. If the RequestType was set to extended response and the Remote Device is the ZigBee coordinator or router with associated devices, The NumAssocDev field shall contain the number of 16-bit NWK addresses contained in the NWKAddrAssocDevList field, the StartIndex field shall be set to the value contained in the NWK_addr_req command and the NWKAddrAssocDevList shall contain the list of all 16-bit NWK addresses of its associated devices, starting with the entry StartIndex and continuing with whole entries until the maximum APS packet length is reached.

The DEVICE_NOT_FOUND status shall be treated as reserved for Version 1.0 and is to be used only with future unicast forms of the NWK_addr_req primitive.

2.4.4.1.2 IEEE_addr_rsp

The IEEE_addr_rsp command (ClusterID=0x8001) shall be formatted as illustrated in Figure 2.57.

Octets: 1	8	2	1	1	Variable
Status	IEEEAddr RemoteDev	NWKAddr RemoteDev	Num AssocDev	StartIndex	NWKAddr AssocDevList

Figure 2.57 Format of the IEEE_addr_rs Command Frame

Table 2.84 specifies the fields of the IEEE_addr_rs command frame.

Table 2.84 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

Table 2.84 IEEE_addr_rsp Parameters (Continued)

Name	Type	Valid Range	Description
NumAssocDev	Integer	0x00-0xff	Count of the number of associated devices to the Remote Device and 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 the RequestType 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 the RequestType in the request is for a Single Device Response, this field shall not be included in the frame

Table 2.84 IEEE_addr_rsp Parameters (Continued)

Name	Type	Valid Range	Description
NWKAddrAssocDevList	Device Address List	List of NumAssocDev 16-bit short addresses, each with range 0x0000 0x-ffff	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 the RequestType in the request is for a Single Device Response, this field shall not be included in the frame

2.4.4.1.2.1 When Generated

The IEEE_addr_rsp is generated from Remote Devices in response to the unicast IEEE_addr_req inquiring as to the 64 bit IEEE address of the Remote Device. The destination addressing on this command shall be unicast.

2.4.4.1.2.2 Effect on Receipt

Upon receipt, a Remote Device shall create a unicast message to the source indicated by the IEEE_addr_req command. Included in the IEEE_addr_rsp payload is the IEEE address and the NWK address of the Remote Device. If the RequestType is one of the reserved values, the Status field shall be set to INV_REQUESTTYPE and the NumAssocDev, StartIndex and NWKAddrAssocDevList fields shall not be included in the frame. If the RequestType was set to single response, the NumAssocDev, StartIndex and NWKAddrAssocDevList fields shall not be included in the frame. If the RequestType was set to extended response and the Remote Device is not the ZigBee coordinator or router, the NumAssocDev field shall be set to 0 and the StartIndex and NWKAddrAssocDevList fields shall not be included in the frame. If the RequestType was set to extended response and the Remote Device is the ZigBee coordinator or router with associated devices, The NumAssocDev field shall contain the number of 16-bit NWK addresses contained in the NWKAddrAssocDevList field, the StartIndex field shall be set to the value contained in the IEEE_addr_req command and the NWKAddrAssocDevList shall contain the list of all 16-bit NWK addresses of its associated devices, starting with the entry StartIndex and continuing with whole entries until the maximum APS packet length is reached.

The DEVICE_NOT_FOUND status shall be treated as reserved for Version 1.0 and is to be used only with future forms of the IEEE_addr_req primitive.

2.4.4.1.3 Node_Desc_rsp

The Node_Desc_rsp command (ClusterID=0x8002) shall be formatted as illustrated in Figure 2.58.

Octets: 1	2	See sub-clause 2.3.2.4
Status	NWKAddr OfInterest	Node Descriptor

Figure 2.58 Format of the Node_Desc_rsp Command Frame

Table 2.85 specifies the fields of the Node_Desc_rsp command frame.

Table 2.85 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 sub-clause 2.3.2.4. This field shall only be included in the frame if the status field is equal to SUCCESS

2.4.4.1.3.1 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 2.85. 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 sub-clause 2.3.2.4) 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 is 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 sub-clause 2.3.2.4) of the matching child device in the NodeDescriptor field.

2.4.4.1.3.2 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.

2.4.4.1.4 Power_Desc_rsp

The Power_Desc_rsp command (ClusterID=0x8003) shall be formatted as illustrated in Figure 2.59.

Octet: 1	**2**	**See sub-clause 2.3.2.5**
Status	NWKAddr OfInterest	Power Descriptor

Figure 2.59 Format of the Power_Desc_rsp Command Frame

Table 2.86 specifies the fields of the Power_Desc_rsp command frame.

Table 2.86 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 sub-clause 2.3.2.5. This field shall only be included in the frame if the status field is equal to SUCCESS

2.4.4.1.4.1 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 2.86. 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 sub-clause 2.3.2.5) 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 is 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 sub-clause 2.3.2.5) of the matching child device in the PowerDescriptor field.

2.4.4.1.4.2 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.

2.4.4.1.5 Simple_Desc_rsp

The Simple_Desc_rsp command (ClusterID=0x8004) shall be formatted as illustrated in Figure 2.60.

Octet: 1	2	1	Variable
Status	NWKAddr OfInterest	Length	Simple Descriptor

Figure 2.60 Format of the Simple_Desc_rsp Command Frame

Table 2.87 specifies the fields of the Simple_Desc_rsp command frame.

Table 2.87 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 sub-clause 2.3.2.6. This field shall only be included in the frame if the status field is equal to SUCCESS

2.4.4.1.5.1 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 2.87. 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 2.45, 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 sub-clause 2.3.2.6) 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 is 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 sub-clause 2.3.2.6) for that endpoint of the matching child device in the SimpleDescriptor field.

2.4.4.1.5.2 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.

2.4.4.1.6 Active_EP_rsp

The Active_EP_rsp command (ClusterID=0x8005) shall be formatted as illustrated in Figure 2.61.

Octet: 1	2	1	Variable
Status	NWKAddr OfInterest	ActiveEP Count	ActiveEP List

Figure 2.61 Format of the Active_EP_rsp Command Frame

Table 2.88 specifies the fields of the Active_EP_rsp command frame.

Table 2.88 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

2.4.4.1.6.1 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 2.88. 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.

2.4.4.1.6.2 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.

2.4.4.1.7 Match_Desc_rsp

The Match_Desc_rsp command (ClusterID=0x8006) shall be formatted as illustrated in Figure 2.62.

Octet: 1	2	1	Variable
Status	NWKAddr OfInterest	Match Length	Match List

Figure 2.62 Format of the Match_Desc_rsp Command Frame

Table 2.89 specifies the fields of the Match_Desc_rsp command frame.

Table 2.89 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

2.4.4.1.7.1 When Generated

The Match_Desc_rsp is generated by a remote device in response to a Match_Desc_req either broadcast or directed to the remote device. This command shall be unicast to the originator of the Match_Desc_req command.

The remote device shall generate the Match_Desc_rsp command using the format illustrated in Table 2.89. If the NWKAddrOfInterest field of the original Match_Desc_req was equal to the broadcast network address for all RxOnWhenIdle devices (0xfffd), the remote device shall apply the match criterion, as described below, that was specified in the original Match_Desc_req command to each of its simple descriptors. If the remote device is the coordinator or a router, it shall also apply the match criterion, as described below, to each simple descriptor that it may have obtained from each of its children.

If the NWKAddrOfInterest field of the original Match_Desc_req was not equal to the broadcast network address for all RxOnWhenIdle devices (0xfffd), the remote device shall set the NWKAddrOfInterest field to the same network address that was specified in the original Match_Desc_req command.

If the NWKAddrOfInterest field matches the network address of the remote device, it shall apply the match criterion, as described below, that was specified in the original Match_Desc_req command to each of its simple descriptors.

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 MatchLength field to 0 and not include the MatchList 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 is 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 MatchLength field to 0 and not include the MatchList field.

If the NWKAddrOfInterest matches the network address of one of the children of the remote device, it shall determine whether any simple descriptors for that device are available. If no simple descriptors are available for the child indicated by the NWKAddrOfInterest field, the remote device shall set the Status field to NO_DESCRIPTOR, set the MatchLength field to 0 and not include the MatchList field. If any simple descriptors are available for the child indicated by the NWKAddrOfInterest field, the remote device shall apply the match criterion, as described below, that was specified in the original Match_Desc_req command to each of these simple descriptors.

The remote device shall apply the match criteria to each simple descriptor (see sub-clause 2.3.2.6) as follows. The remote device shall first check that the ProfileID field matches the application profile identifier field of the simple descriptor. If the profile identifiers do not match, the remote device shall assume the match to be unsuccessful and perform no further matching.

If the profile identifiers match, the remote device shall determine whether the match criteria contains a list of input clusters (the NumInClusters field is not equal to 0). If the match criteria contains a list of input clusters, the remote device shall check that at least one of the cluster identifiers listed in the InClusterList field matches one of the cluster identifiers in the application input cluster list field of the simple descriptor. If at least one matching input cluster is found, the remote device shall assume the match to be successful, note the identifier of the endpoint to which this simple descriptor refers and perform no further matching.

If the remote device is unable to find any matching input clusters, it shall determine whether the match criterion contains a list of output clusters (the NumOutClusters field is not equal to 0). If the match criterion contains a list of output clusters, the remote device shall check that at least one of the cluster identifiers listed in the OutClusterList field matches one of the cluster identifiers in the application output cluster list field of the simple descriptor. If at least one matching output cluster is found, the remote device shall assume the match to be successful and note the identifier of the endpoint to which this simple descriptor refers. If the remote device is unable to find any output matching clusters, it shall assume to be unsuccessful.

If the above procedure produces one or more matches, the remote device shall construct a separate Match_Desc_rsp command for each matching device (including itself). For each response, the Status field shall be set to SUCCESS, the NWKAddrOfInterest field shall be set to the address of the appropriate matching device, the MatchLength field shall be set to the number of simple descriptors that matched the criteria for the appropriate matching device and the MatchList field shall contain an ascending list of the endpoints on which a simple descriptor matched the criteria for the appropriate matching device.

2.4.4.1.7.2 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.

2.4.4.1.8 Complex_Desc_rsp

The Complex_Desc_rsp command (ClusterID=0x8010) shall be formatted as illustrated in Figure 2.63.

Octet: 1	2	1	Variable
Status	NWKAddr OfInterest	Length	Complex Descriptor

Figure 2.63 Format of the Complex_Desc_rsp Command Frame

Table 2.90 specifies the fields of the Complex_Desc_rsp command frame.

Table 2.90 Fields of the Complex_Desc_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, 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
ComplexDescriptor	Complex Descriptor		See the Complex Descriptor format in sub-clause 2.3.2.7. This field shall only be included in the frame if the status field is equal to SUCCESS

2.4.4.1.8.1 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 2.90. 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 sub-clause 2.3.2.7) 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 is 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 sub-clause 2.3.2.7) of the matching child device in the ComplexDescriptor field.

2.4.4.1.8.2 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.

2.4.4.1.9 User_Desc_rsp

The User_Desc_rsp command (ClusterID=0x8011) shall be formatted as illustrated in Figure 2.64.

Octet: 1	2	1	Variable
Status	NWKAddr OfInterest	Length	User Descriptor

Figure 2.64 Format of the User_Desc_rsp Command Frame

Table 2.91 specifies the fields of the User_Desc_rsp command frame.

Table 2.91 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 sub-clause 2.3.2.8. This field shall only be included in the frame if the status field is equal to SUCCESS

2.4.4.1.9.1 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 2.91. 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 sub-clause 2.3.2.8) 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 is 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 sub-clause 2.3.2.8) of the matching child device in the UserDescriptor field.

2.4.4.1.9.2 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.

2.4.4.1.10 System_Server_Discovery_rsp

The System_Server_Discovery_rsp command (ClusterID=0x8015) shall be formatted as illustrated in Figure 2.68.

Octet: 1	**2**
Status	ServerMask

Figure 2.65 System_Server_Discovery_rsp Command Frame

Table 2.92 specifies the fields of the Discovery_register_rsp command frame.

Table 2.92 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 2.29 for bit assignments

2.4.4.1.10.1 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.

2.4.4.1.10.2 Effect on Receipt

The requesting device is notified that this device has some of the system server functionality that the requesting device is seeking.

2.4.4.1.11 User_Desc_conf

The User_Desc_conf command (ClusterID=0x8014) shall be formatted as illustrated in Figure 2.66.

Octets: 1	2
Status	NWKAddr OfInterest

Figure 2.66 Format of the User_Desc_conf Command Frame

Table 2.93 specifies the fields of the User_Desc_conf command frame.

Table 2.93 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

2.4.4.1.11.1 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 2.93. 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 is 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.

2.4.4.1.11.2 Effect on Receipt

The local device is notified of the results of its attempt to configure the user descriptor on a remote device.

2.4.4.1.12 Discovery_Cache_rsp

The Discovery_Cache_rsp command (ClusterID=0x8012) shall be formatted as illustrated in Figure 2.67.

Octets: 1
Status

Figure 2.67 Format of the Discovery_Cache_rsp Command Frame

Table 2.94. specifies the fields of the Discovery_Cache_rsp Command Frame.

Table 2.94 Fields of the Discovery_Cache_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS	The status of the Discovery_Cache_req command

2.4.4.1.12.1 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.

2.4.4.1.12.2 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.

2.4.4.1.13 Discovery_store_rsp

The Discovery_store_rsp command (ClusterID=0x8016) shall be formatted as illustrated in Figure 2.68.

Octets: 1
Status

Figure 2.68 Format of the Discovery_store_rsp Command Frame

Table 2.95 specifies the fields of the Discovery_store_rsp command frame

Table 2.95 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

2.4.4.1.13.1 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 whether insufficient space exists.

2.4.4.1.13.2 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.

2.4.4.1.14 Node_Desc_store_rsp

The Node_Desc_store_rsp command (ClusterID=0x8017) shall be formatted as illustrated in Figure 2.69.

Octets: 1
Status

Figure 2.69 Format of the Node_Desc_store_rsp Command Frame

Table 2.96 specifies the fields of the Node_Desc_store_rsp command frame.

Table 2.96 Fields of the Node_Desc_store_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INSUFFICIENT_SPACE or NOT_SUPPORTED	The status of the Node_store_rsp command

2.4.4.1.14.1 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 whether insufficient space exists.

2.4.4.1.14.2 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 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.

2.4.4.1.15 Power_Desc_store_rsp

The Power_Desc_store_rsp command (ClusterID=0x8018) shall be formatted as illustrated in Figure 2.70.

Octets: 1	8	Variable
Status	IEEEAddr	PowerDescriptor

Figure 2.70 Format of the Power_Desc_store_rsp Command Frame

Table 2.97 specifies the fields of the Power_Desc_store_rsp command frame.

Table 2.97 Fields of the Power_Desc_store_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INSUFFICIENT_SPACE or NOT_SUPPORTED	The status of the Power_store_rsp command

2.4.4.1.15.1 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 whether insufficient space exists.

2.4.4.1.15.2 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 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.

2.4.4.1.16 Active_EP_store_rsp

The Active_EP_store_rsp command (ClusterID=0x8019) shall be formatted as illustrated in Figure 2.71.

Octets: 1
Status

Figure 2.71 Format of the Active_EP_store_rsp Command Frame

Table 2.98 specifies the fields of the Active_EP_store_rsp command frame.

Table 2.98 Fields of the Active_EP_store_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INSUFFICIENT_SPACE or NOT_SUPPORTED	The status of the Active_EP_store_rsp command

2.4.4.1.16.1 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), whether the request is not supported (meaning the Remote Device is not a Primary Discovery Cache device) or whether insufficient space exists.

2.4.4.1.16.2 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 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.

2.4.4.1.17 Simple_Desc_store_rsp

The Simple_Desc_store_rsp command (ClusterID=0x801a) shall be formatted as illustrated in Figure 2.72.

Octets: 1
Status

Figure 2.72 Format of the Simple_Desc_store_rsp Command Frame

Table 2.99 specifies the fields of the Simple_Desc_store_rsp command frame.

Table 2.99 Fields of the Simple_Desc_store_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INSUFFICIENT_SPACE or NOT_SUPPORTED	The status of the Simple_desc_store_rsp command

2.4.4.1.17.1 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), whether the request is not supported (meaning the Remote Device is not a Primary Discovery Cache device) or whether insufficient space exists.

2.4.4.1.17.2 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 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.

2.4.4.1.18 Remove_node_cache_rsp

The Remove_node_cache_rsp command (ClusterID=0x801b) shall be formatted as illustrated in Figure 2.73.

Octets: 1
Status

Figure 2.73 Format of the Remove_node_cache_rsp Command Frame

Table 2.100 specifies the fields of the Remove_node_cache_rsp command frame.

Table 2.100 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

2.4.4.1.18.1 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 whether the request is not supported (meaning the Remote Device is not a Primary Discovery Cache device).

2.4.4.1.18.2 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.

2.4.4.1.19 Find_node_cache_rsp

The Find_node_cache_rsp command (ClusterID=0x801c) shall be formatted as illustrated in Figure 2.74.

Octets: 2	2	8
CacheNWKAddress	NWKAddr	IEEEAddr

Figure 2.74 Format of the Find_node_cache_rsp Command Frame

Table 2.101 specifies the fields of the Find_node_cache_rsp command frame.

Table 2.101 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

2.4.4.1.19.1 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.

2.4.4.1.19.2 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 End Device Bind, Bind, Unbind Bind Management Server Services

Table 2.102 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 subclauses.

Table 2.102 End Device Bind, Unbind and Bind Management Server Services Primitives

End Device Bind, Bind and Unbind Server Service Commands	Server Processing
End_Device_Bind_rsp	O
Bind_rsp	O
Unbind_rsp	O
Bind_Register_rsp	O
Replace_Device_rsp	O
Store_Bkup_Bind_Entry_rsp	O
Remove_Bkup_Bind_Entry_rsp	O
Backup_Bind_Table_rsp	O
Recover_Bind_Table_rsp	O
Backup_Source_Bind_rsp	O
Recover_Source_Bind_rsp	O

2.4.4.2.1 End_Device_Bind_rsp

The End_Device_Bind_rsp command (ClusterID=0x8020) shall be formatted as illustrated in Figure 2.75.

Octets: 1
Status

Figure 2.75 Format of the End_Device_Bind_rsp Command Frame

Table 2.103 specifies the fields of the End_Device_Bind_rsp command frame.

Table 2.103 Fields of the End_Device_Bind_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, INVALID_EP, TIMEOUT or NO_MATCH	The status of the End_Device_Bind_req command

2.4.4.2.1.1 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. Else, End_Device_Bind_req processing is performed as described below including transmission of the End_Device_Bind_rsp.

2.4.4.2.1.2 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 values 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. Else, 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 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[9] directed to the Local Device supplying the End_Device_Bind_req with matched elements of the OutClusterList.[10]

9. CCB #602

2.4.4.2.2 Bind_rsp

The Bind_rsp command (ClusterID=0x8021) shall be formatted as illustrated in Figure 2.76.

Octets: 1
Status

Figure 2.76 Format of the Bind_rsp Command Frame

Table 2.104 specifies the fields of the Bind_rsp command frame.

Table 2.104 Fields of the Bind_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, INVALID_EP or TABLE_FULL	The status of the Bind_req command

2.4.4.2.2.1 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 the a Primary binding table cache 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.[11] 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.

2.4.4.2.2.2 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.

10. CCB #518
11. CCB #531

2.4.4.2.3 Unbind_rsp

The Unbind_rsp command (ClusterID=0x8022) shall be formatted as illustrated in Figure 2.77.

Octets: 1
Status

Figure 2.77 Format of the Unbind_rsp Command Frame

Table 2.105 specifies the fields of the Unbind_rsp command frame.

Table 2.105 Fields of the Unbind_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, NOT_SUPPORTED, INVALID_EP or NO_ENTRY	The status of the Unbind_req command

2.4.4.2.3.1 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.[12] 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.

2.4.4.2.3.2 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.

12. CCB #531

2.4.4.2.4 Bind_Register_rsp

The Bind_Register_rsp command (ClusterID=0x8023) shall be formatted as illustrated in Figure 2.78.

Octets: 1	2	2	Variable
Status	BindingTableEntries	BindingTableListCount	BindingTableList

Figure 2.78 Format of the Bind_Register_rsp Command Frame

Table 2.106 specifies the fields of the Bind_Register_rsp command frame.

Table 2.106 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
BindingTable Entries	Integer	0x0000 - 0ffff	Number of binding table entries for the requesting device held by the primary binding table cache
BindingTable ListCount	Integer	0x0000 - 0xffff	Number of source binding table entries contained in this response
BindingTable List	List of source binding descriptors	This list shall contain the number of elements given by the BindingTableListCount	A list of source binding

2.4.4.2.4.1 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.

2.4.4.2.4.2 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.

2.4.4.2.5 Replace_Device_rsp

The Replace_Device_rsp command (ClusterID=0x8024) shall be formatted as illustrated in Figure 2.79.

Octets: 1
Status

Figure 2.79 Format of the Replace_Device_rsp Command Frame

Table 2.107 specifies the fields of the Replace_Device_rsp command frame

Table 2.107 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

2.4.4.2.5.1 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.

2.4.4.2.5.2 Effect on Receipt

The requesting device is notified of the status of its Replace_Device_req command.

2.4.4.2.6 Store_Bkup_Bind_Entry_rsp

The Store_Bkup_Bind_Entry_rsp command (ClusterID=0x8025) shall be formatted as illustrated in Figure 2.80.

Octets: 1
Status

Figure 2.80 Format of the Store_Bkup_Bind_Entry_rsp Command Frame

Table 2.108 specifies the fields of the Store_Bkup_Bind_Entry_rsp command frame.

Table 2.108 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

2.4.4.2.6.1 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.

2.4.4.2.6.2 Effect on Receipt

The requesting device is notified of the status of its attempt to store a bind entry.

2.4.4.2.7 Remove_Bkup_Bind_Entry_rsp

The Remove_Bkup_Bind_Entry_rsp command (ClusterID=0x8026) shall be formatted as illustrated in Figure 2.81.

Octets: 1
Status

Figure 2.81 Format of the Remove_Bkup_Bind_Entry_rsp Command Frame

Table 2.109 specifies the fields of the Remove_Bkup_Bind_Entry_rsp command frame.

Table 2.109 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

2.4.4.2.7.1 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.

2.4.4.2.7.2 Effect on Receipt

The requesting device is notified of the status of its attempt to remove a bind entry from the backup cache.

2.4.4.2.8 Backup_Bind_Table_rsp

The Backup_Bind_Table_rsp command (ClusterID=0x8027) shall be formatted as illustrated in Figure 2.82.

Octets: 1	2
Status	EntryCount

Figure 2.82 Format of the Backup_Bind_Table_rsp Command Frame

Table 2.110 specifies the fields of the Backup_Bind_Table_rsp command frame.

Table 2.110 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 bind table

2.4.4.2.8.1 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.

2.4.4.2.8.2 Effect on Receipt

The requesting device is notified of the status of its attempt to store a bind table.

2.4.4.2.9 Recover_Bind_Table_rsp

The Backup_Bind_Table_rsp command (ClusterID=0x8028) shall be formatted as illustrated in Figure 2.83.

Octets: 1	2	2	2	Variable
Status	BindingTableEntries	StartIndex	BindingTableListCount	BindingTableList

Figure 2.83 Format of the Backup_Bind_Table_rsp Command Frame

Table 2.111 specifies the fields of the Recover_Bind_Table_rsp command frame.

Table 2.111 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
BindingTable Entries	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
BindingTable ListCount	Integer	0x0000 - 0xffff	Number of binding entries included within BindingTableList
BindingTable List	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

2.4.4.2.9.1 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, BindingTableListCount is the number of entries which is being returned in the response.

2.4.4.2.9.2 Effect on Receipt

The requesting device is notified of the status of its attempt to restore a bind table.

2.4.4.2.10 Backup_Source_Bind_rsp

The Backup_Source_Bind_rsp command (ClusterID=0x8029) shall be formatted as illustrated in Figure 2.84.

Octets: 1
Status

Figure 2.84 Format of the Backup_Source_Bind_rsp Command Frame

Table 2.112 specifies the fields of the Backup_Source_Bind_rsp command frame.

Table 2.112 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

2.4.4.2.10.1 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 bind 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.

2.4.4.2.10.2 Effect on Receipt

The requesting device is notified of the status of its attempt to backup the source bind table.

2.4.4.2.11 Recover_Source_Bind_rsp

The Recover_Source_Bind_rsp command (ClusterID=0x802a) shall be formatted as illustrated in Figure 2.85.

Octets: 1	2	2	2	Variable
Status	SourceTableEntries	StartIndex	SourceTableListCount	SourceTableList

Figure 2.85 Format of the Recover_Source_Bind_rsp Command Frame

Table 2.113 specifies the fields of the Recover_Source_Bind_rsp command frame.

Table 2.113 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
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)

2.4.4.2.11.1 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, SourceTableListCount is the number of entries which is being returned in the response.

2.4.4.2.11.2 Effect on Receipt

The requesting device is notified of the status of its attempt to restore a source bind table.

2.4.4.3 Network Management Server Services

Table 2.114 lists the commands supported by Device Profile: Network Management Server Services. Each of these commands will be discussed in the following sub-clauses.

Table 2.114 Network Management Server Service Commands

Network Management Server Service Command	Server Processing
Mgmt_NWK_Disc_rsp	O
Mgmt_Lqi_rsp	O
Mgmt_Rtg_rsp	O
Mgmt_Bind_rsp	O
Mgmt_Leave_rsp	O
Mgmt_Direct_Join_rsp	O
Mgmt_Permit_Joining_rsp	M
Mgmt_Cache_rsp	O

2.4.4.3.1 Mgmt_NWK_Disc_rsp

The Mgmt_NWK_Disc_rsp command (ClusterID=0x8030) shall be formatted as illustrated in Figure 2.86.

Octets: 1	1	1	1	Variable
Status	Network Count	Start Index	NetworkList Count	NetworkList

Figure 2.86 Format of the Mgmt_NWK_Disc_rsp Command Frame

Table 2.115 specifies the fields of the Mgmt_NWK_Disc_rsp command frame.

Table 2.115 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.

Table 2.115 Fields of the Mgmt_NWK_Disc_rsp Command (Continued)

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.
NetworkListCount	Integer	0x00-0xff	The number of network list descriptors reported within this response.
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 Table 2.116.

Table 2.116 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].

Table 2.116 NetworkList Record Format

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	-		Each of these bit shall be set to 0.

2.4.4.3.1.1 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 processing.

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 2.116, until the limit on MSDU size, i.e. *aMaxMACFrameSize* (see [B1]), is reached. The number of results reported shall be set in the NetworkListCount.

2.4.4.3.1.2 Effect on Receipt

The local device is notified of the results of its attempt to perform a remote network discovery.

2.4.4.3.2 Mgmt_Lqi_rsp

The Mgmt_Lqi_rsp command (ClusterID=0x8031) shall be formatted as illustrated in Figure 2.87.

Octets: 1	1	1	1	Variable
Status	NeighborTable Entries	Start Index	NeighborTable ListCount	NeighborTable List

Figure 2.87 Format of the Mgmt_Lqi_rsp Command Frame

Table 2.117 specifies the fields of the Mgmt_Lqi_rsp command frame.

Table 2.117 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	0x00-0x02	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	0x00-0xff	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 NeighborTableListCount, of the elements in the Remote Device's Neighbor Table including the device address and associated LQI (see Table 2.118 for details).

Table 2.118 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	0x0-0x3	The type of the neighbor device: 0x0 = ZigBee coordinator 0x1 = ZigBee router 0x2 = ZigBee end device 0x3 = Unknown
RxOnWhenIdle	2	0x0-0x2	Indicates if neighbor's receiver is enabled during idle portions of the CAP: 0x0 = Receiver is off 0x1 = Receiver is on 0x2 = unknown
Relationship	3	0x0-0x4	The relationship between the neighbor and the current device: 0x0 = neighbor is the parent 0x1 = neighbor is a child 0x2 = neighbor is a sibling 0x3 = None of the above 0x4 = previous child
Reserved	1		This reserved bit shall be set to 0.
Permit joining	2	0x0-0x2	An indication of whether the neighbor device is accepting join requests: 0x0 = neighbor is not accepting join requests 0x1 = neighbor is accepting join requests 0x2 = unknown

Table 2.118 NeighborTableList Record Format (Continued) (Continued)

Name	Size (Bits)	Valid Range	Description
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

2.4.4.3.2.1 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 2.118) 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, 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 2.118) 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.

2.4.4.3.2.2 Effect on Receipt

The local device is notified of the results of its attempt to obtain the neighbor table.

2.4.4.3.3 Mgmt_Rtg_rsp

The Mgmt_Rtg_rsp command (ClusterID=0x8032) shall be formatted as illustrated in Figure 2.88.

Octets: 1	1	1	1	Variable
Status	RoutingTable Entries	Start Index	RoutingTable ListCount	RoutingTable List

Figure 2.88 Format of the Mgmt_Rtg_rsp Command Frame

Table 2.119 specifies the fields of the Mgmt_Rtg_rsp command frame. .

Table 2.119 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 the Table 2.120 for details).

Table 2.120 RoutingTableList Record Format

Name	Size (Bits)	Valid Range	Description
Destination address	16	The 16-bit network address of this route.	Destination address.

Table 2.120 RoutingTableList Record Format (Continued)

Name	Size (Bits)	Valid Range	Description
Status	3	The status of the route.	0x0=ACTIVE. 0x1=DISCOVERY_UNDERWAY. 0x2=DISCOVERY_FAILED. 0x3=INACTIVE. 0x4-0x7=RESERVED.
Reserved	5		Each of these reserved bits shall be set to zero.
Next-hop address	16	The 16-bit network address of the next hop on the way to the destination.	Next-hop address.

CCB Comment #250

2.4.4.3.3.1 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 2.119) 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.

2.4.4.3.3.2 Effect on Receipt

The local device is notified of the results of its attempt to obtain the routing table.

2.4.4.3.4 Mgmt_Bind_rsp

The Mgmt_Bind_rsp command (ClusterID=0x8033) shall be formatted as illustrated in Figure 2.89.

Octets: 1	1	1	1	Variable
Status	BindingTable Entries	Start Index	BindingTable ListCount	BindingTable List

Figure 2.89 Format of the Mgmt_Bind_rsp Command Frame

Table 2.121 specifies the fields of the Mgmt_Bind_rsp command frame.

Table 2.121 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 2.122 for details).

Table 2.122 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 - 0xff	The source endpoint for the binding entry.
ClusterId	16[a]	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 DstAddrMode 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.

a. CCB #603

2.4.4.3.4.1 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 2.122) 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.

2.4.4.3.4.2 Effect on Receipt

The local device is notified of the results of its attempt to obtain the binding table.

2.4.4.3.5 Mgmt_Leave_rsp

The Mgmt_Leave_rsp command (ClusterID=0x8034) shall be formatted as illustrated in Figure 2.90.

Octets: 1
Status

Figure 2.90 Format of the Mgmt_Leave_rsp Command Frame

Table 2.123 specifies the fields of the Mgmt_Leave_rsp command frame.

Table 2.123 Fields of the Mgmt_Leave_rsp Command

Name	Type	Valid Range	Description
Status	Integer	NOT_SUPPORTED or any status code returned from the NLME-LEAVE.confirm primitive	The status of the Mgmt_Leave_req command.

2.4.4.3.5.1 When Generated

The Mgmt_Leave_rsp is generated in response to a Mgmt_Leave_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 of and after support for the Mgmt_Leave_req has been verified, the Remote Device shall execute the NLME-LEAVE.request to disassociate from the currently associated network. The Mgmt_Leave_rsp shall contain the same status that was contained in the NLME-LEAVE.confirm primitive.

Once a device has disassociated, it may[13] execute pre-programmed logic to perform NLME-NETWORK-DISCOVERY and NLME-JOIN to join/re-join a network.

2.4.4.3.5.2 Effect on Receipt

The local device is notified of the results of its attempt to cause a remote device to leave the network.

2.4.4.3.6 Mgmt_Direct_Join_rsp

The Mgmt_Direct_Join_rsp (ClusterID=0x8035) shall be formatted as illustrated in Figure 2.91.

Octets: 1
Status

Figure 2.91 Format of the Mgmt_Direct_Join_rsp Command Frame

Table 2.124 specifies the fields of the Mgmt_Direct_Join_rsp command frame.

Table 2.124 Fields of the Mgmt_Direct_Join_rsp Command

Name	Type	Valid Range	Description
Status	Integer	NOT_SUPPORTED or any status code returned from the NLME-DIRECT-JOIN.confirm primitive	The status of the Mgmt_Direct_Join_req command.

2.4.4.3.6.1 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 of 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.

13. CCB #553

2.4.4.3.6.2 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.

2.4.4.3.7 Mgmt_Permit_Joining_rsp

The Mgmt_Permit_Joining_rsp command (ClusterID=0x8036) shall be formatted as illustrated in Figure 2.92.

Octets: 1
Status

Figure 2.92 Format of the Mgmt_Permit_Joining_rsp Command Frame

Table 2.125 specifies the fields of the Mgmt_Permit_Joining_rsp command frame .

Table 2.125 Fields of the Mgmt_Permit_Joining_rsp Command

Name	Type	Valid Range	Description
Status	Integer	SUCCESS, INVALID_REQUEST or any status code returned from the NLME-PERMIT-JOINING.confirm primitive	The status of the Mgmt_Permit_Joining_rsp command

2.4.4.3.7.1 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.

2.4.4.3.7.2 Effect on Receipt

The status of the Mgmt_Permit_Joining_req command is notified to the requestor.

2.4.4.3.8 Mgmt_Cache_rsp

The Mgmt_Cache_rsp command (ClusterID=0x8037) shall be formatted as illustrated in Figure 2.93

Octets: 1	1	1	1	Variable
Status	DiscoveryCacheEntries	StartIndex	DiscoveryCacheList Count	DiscoveryCacheList

Figure 2.93 Format of the Mgmt_Cache_rsp Command Frame

Table 2.126 specifies the fields of the Mgmt_Cache_rsp command frame.

Table 2.126 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
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 sub-clause .

Table 2.127 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

2.4.4.3.8.1 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 2.129) 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.

2.4.4.3.8.2 Effect on Receipt

The local device is notified of the results of its attempt to obtain the discovery cache list.

2.4.5 ZDP Enumeration Description

This subclause explains the meaning of the enumerations used in the ZDP. Table shows a description of the ZDP enumeration values.

I

Table 2.128 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 between 0xf1 and 0xff.
NOT_ACTIVE	0x83	The requested endpoint is not described by a simple descriptor.

Table 2.128 ZDP Enumerations Description (Continued)

Enumeration	Value	Description
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
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
	0x8d-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 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 (Reference [B1]) and ZigBee Network Layer (Chapter 3).

ZigBee Device Objects is an application which employs 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. These attributes are not the elements of any cluster. The configuration attributes within ZigBee Device Objects are configuration parameters 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 1-240.
- Assembling configuration information from the end applications to determine and implement the functions described in the following subclauses.

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 2.94, shall be supported by the Primary Discovery Cache device:

- Undiscovered:
    - The client employs the radius limited broadcast to all RxOnWhenIdle devices message Discovery Register request to locate a Primary Discovery Cache device within the radius supplied by the request
- Discovered:

    - The client employs the unicast Discovery cache 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 or TABLE_FULL.

- 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. 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 2.94 for details on state machine processing for the Primary Discovery Cache device.

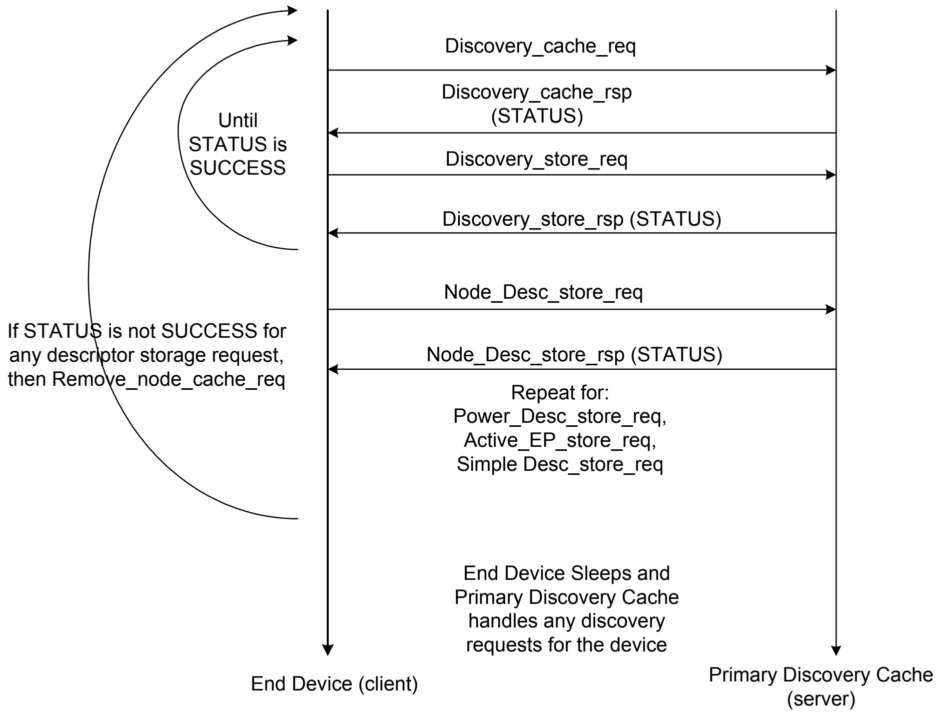

Discovery_cache_req
Discovery_cache_rsp (STATUS)
Until STATUS is SUCCESS
Discovery_store_req
Discovery_store_rsp (STATUS)
Node_Desc_store_req
If STATUS is not SUCCESS for any descriptor storage request, then Remove_node_cache_req
Node_Desc_store_rsp (STATUS)
Repeat for: Power_Desc_store_req, Active_EP_store_req, Simple Desc_store_req
End Device Sleeps and Primary Discovery Cache handles any discovery requests for the device
End Device (client)
Primary Discovery Cache (server)

Figure 2.94 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 ZigBee Coordinator, ZigBee Router and ZigBee End Device types, this function shall perform the following:

- Within each network employing sleeping ZigBee End Devices, some ZigBee Routers (or the ZigBee Coordinator) must 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 shall 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:
  - 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.
  - 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.

    —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. In cases where the application profiles wish to enumerate input clusters and their resulting 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. It shall be assumed, in these cases, that the application profile provides details about the use of the cluster identifier for both inputs and response output.

—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.

—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:

- Establish Key
- Transport Key
- Request Key
- Update Device
- Remove Device
- Switch Key

The Security Manager function addresses the Security Services Specification (Chapter 4). Security Management, implemented by APSME primitive calls by ZDO, performs the following:

- Contacts the Trust Center (assumed to be located on the ZigBee Coordinator) to obtain the Master Key between this device and the Trust Center (step is omitted if this device is the ZigBee Coordinator or the Master Key with the Trust Center has been pre-configured). This step employs the Transport Key primitive.

- Establishes a Link Key with the Trust Center. This step employs the APSME-Establish-Key primitive.
- Obtains the NWK Key from the Trust Center using secured communication with the Trust Center. This step employs the APSME-TRANSPORT-KEY primitive.
- Establishes Link Keys and master keys, as required, with specific devices in the network identified as messaging destinations. These steps employ the APSME-ESTABLISH-KEY and/or APSME-REQUEST-KEY primitives.
- Informs the trust center of any devices that join the network using the APSME-DEVICE-UPDATE 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 and join by proxy features provided by the Network Layer. 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.

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 parameter 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 PDA 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.

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
- 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 of 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

- Provides a remote management command to Permit or disallow joining on particular routers or to generally allow or disallow joining via the Trust center.

2.5.3 Layer Interface Description

Unlike other device descriptors for applications residing above Endpoints 1-240, the ZigBee Device Objects (ZDO) interface to the APS via the APSME-SAP and to NWK via the NLME-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).

Due to the omission of a Simple Descriptor for the Device Profile and due to the role of the Device Profile in discovery itself, the use of binding and indirect messaging on endpoint 0 is prohibited.

2.5.4 System Usage

The header will be on the same page as the diagram with the release version of the specification.

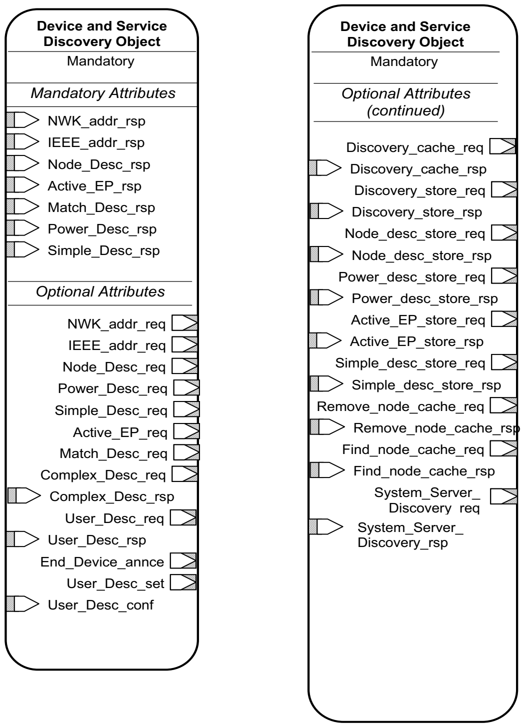
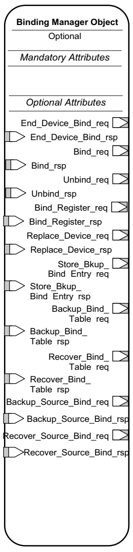
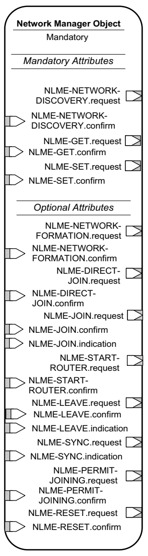
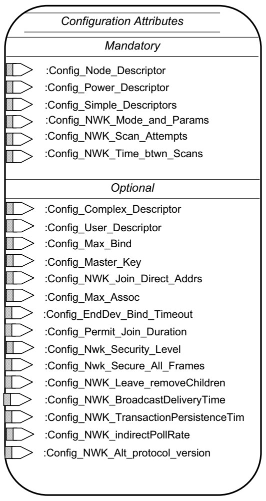
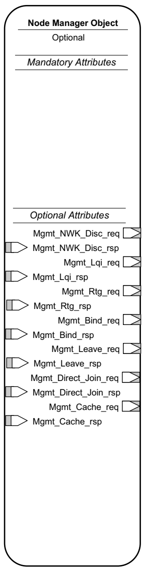


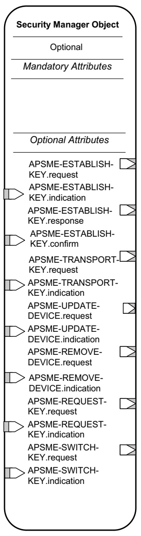
Figure 2.95 System Usage ZigBee Device Object Details


Binding Manager Object
Optional
Mandatory Attributes
Optional Attributes
End_Device_Bind_req
End_Device_Bind_rsp
Bind_req
Bind_rsp
Unbind_req
Unbind_rsp
Bind_Register_req
Bind_Register_rsp
Replace_Device_req
Replace_Device_rsp
Store_Bkup_ Bind_Entry_req
Store_Bkup_ Bind_Entry_rsp
Backup_Bind_ Table_req
Backup_Bind_ Table_rsp
Recover_Bind_ Table_req
Recover_Bind_ Table_rsp
Backup_Source_Bind_req
Backup_Source_Bind_rsp
Recover_Source_Bind_req
Recover_Source_Bind_rsp


Network Manager Object
Mandatory
Mandatory Attributes
NLME-NETWORK-DISCOVERY.request
NLME-NETWORK-DISCOVERY.confirm
NLME-GET.request
NLME-GET.confirm
NLME-SET.request
NLME-SET.confirm
Optional Attributes
NLME-NETWORK-FORMATION.request
NLME-NETWORK-FORMATION.confirm
NLME-DIRECT-JOIN.request
NLME-DIRECT-JOIN.confirm
NLME-JOIN.request
NLME-JOIN.confirm
NLME-JOIN.indication
NLME-START-ROUTER.request
NLME-START-ROUTER.confirm
NLME-LEAVE.request
NLME-LEAVE.confirm
NLME-LEAVE.indication
NLME-SYNC.request
NLME-SYNC.indication
NLME-PERMIT-JOINING.request
NLME-PERMIT-JOINING.confirm
NLME-RESET.request
NLME-RESET.confirm


Configuration Attributes
Mandatory
:Config_Node_Descriptor
:Config_Power_Descriptor
:Config_Simple_Descriptors
:Config_NWK_Mode_and_Params
:Config_NWK_Scan_Attempts
:Config_NWK_Time_btwn_Scans
Optional
:Config_Complex_Descriptor
:Config_User_Descriptor
:Config_Max_Bind
:Config_Master_Key
:Config_NWK_Join_Direct_Addrs
:Config_Max_Assoc
:Config_EndDev_Bind_Timeout
:Config_Permit_Join_Duration
:Config_Nwk_Security_Level
:Config_Nwk_Secure_All_Frames
:Config_NWK_Leave_removeChildren
:Config_NWK_BroadcastDeliveryTime
:Config_NWK_TransactionPersistenceTim
:Config_NWK_indirectPollRate
:Config_NWK_Alt_protocol_version


Node Manager Object
Optional
Mandatory Attributes
Optional Attributes
Mgmt_NWK_Disc_req
Mgmt_NWK_Disc_rsp
Mgmt_Lqi_req
Mgmt_Lqi_rsp
Mgmt_Rtg_req
Mgmt_Rtg_rsp
Mgmt_Bind_req
Mgmt_Bind_rsp
Mgmt_Leave_req
Mgmt_Leave_rsp
Mgmt_Direct_Join_req
Mgmt_Direct_Join_rsp
Mgmt_Cache_req
Mgmt_Cache_rsp


Security Manager Object
Optional
Mandatory Attributes
Optional Attributes
APSME-ESTABLISH-KEY.request
APSME-ESTABLISH-KEY.indication
APSME-ESTABLISH-KEY.response
APSME-ESTABLISH-KEY.confirm
APSME-TRANSPORT-KEY.request
APSME-TRANSPORT-KEY.indication
APSME-UPDATE-DEVICE.request
APSME-UPDATE-DEVICE.indication
APSME-REMOVE-DEVICE.request
APSME-REMOVE-DEVICE.indication
APSME-REQUEST-KEY.request
APSME-REQUEST-KEY.indication
APSME-SWITCH-KEY.request
APSME-SWITCH-KEY.indication

2.5.5 Object Definition and Behavior

2.5.5.1 Object Overview

ZigBee Device Objects contains five Objects:

- Device and Service Discovery
- Network Manager
- Binding Manager
- Security Manager
- Node Manager

Table 2.129 describes these ZigBee Device Objects.

Table 2.129 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.

2.5.5.2 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.

2.5.5.3 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 Network Key, as opposed to individual Link Keys.

2.5.5.4 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 240).

2.5.5.5 State Machine Functional Descriptions

2.5.5.5.1 ZigBee Coordinator

2.5.5.5.1.1 Initialization

Provision shall be made within the implementation to supply a single copy of desired network configuration parameters (:Config_NWK_Mode_and_Params) to the Network Object of ZigBee Device Objects. 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 configuration 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 sub-clause 2.5.2.1.

If supported, provision shall be made to supply configuration elements for the Complex Descriptor, User Descriptor, the maximum number of bind entries and the master key. These elements shall be embodied in :Config_Complex_Descriptor, :Config_User_Descriptor, :Config_Max_Bind and :Config_Master_Key.

The device application shall use NLME-NETWORK-DISCOVERY.request with the ChannelList portion of :Config_NWK_Mode_and_Params to scan the specified channels. The resulting NLME-NETWORK-DISCOVERY.confirm shall supply a NetworkList detailing active PANs within that range. The device application shall compare the ChannelList to the NetworkList and select an unused channel. Specification of the algorithm for selection of the unused channel shall be left to the implementer. Once the unused channel is identified, the device application shall set the *nwkSecurityLevel* and *nwkSecureAllFrames* NIB attributes according to the values established by convention within the Stack Profile employed by the device. It shall then employ the NLME-NETWORK-FORMATION.request using the parameters specified within :Config_NWK_Mode_and_Params to establish a PAN on that channel. The Extended PAN ID field shall be set within *nwkExtendedPANID*. 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 parameter value supplied using the NLME-PERMIT-JOINING.request. Additionally, the *nwkNetworkBroadcastDeliveryTime* and *nwkTransactionPersistenceTime* Network Information Block parameters shall be set with :Config_NWK_BroadcastDeliveryTime and :Config_NWK_TransactionPersistenceTime respectively (see Chapter 3).

Provision shall be made to ensure APS primitive calls from the end applications over EP 1 through EP 240 return appropriate error status values prior to completion of the Initialization state by ZigBee Device Objects and transition to the normal operating state.

2.5.5.5.1.2 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 parameter 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. When a ZigBee end device is asked to leave the network, the ReuseAddress parameter of the NLME-LEAVE.request primitive should have a value of TRUE indicating that the NWK layer may give the address formerly in use by the leaving device to another end device that joins subsequently.[14]

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 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. 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 End_Dev_Bind_rsp.

The ZigBee coordinator shall process End_Device_annce messages from ZigBee End Devices. Upon receipt of an End_Device_annce, 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 End_Device_annce message. At minimum, the Binding Table and Trust Center tables shall be checked. If a match is detected, the ZigBee coordinator shall update its APS Information Block address map entries corresponding to the matched 64 bit IEEE address to reflect the updated 16 bit NWK address contained in the End_Device_annce.

14. CCB #675

2.5.5.5.1.3 Trust Center Operation

The Zigbee coordinator shall also function as the trust center when security is enabled on the network.

The trust center is notified of new devices on the network via the APSME-DEVICE-UPDATE .indication. The trust center can either choose to allow the device to remain on the network or force it out of the network using the APSME-REMOVE-DEVICE.request. This choice is made using a network access control policy that is beyond the scope of this specification.

If the trust center decides to allow the device to remain in the network, it shall establish a master key with that device using APSME-TRANSPORT-KEY.request, unless the master key is already available to both the device and trust center using out-of-band mechanisms that guarantee secrecy and authenticity. Upon exchange of the master key, the trust center shall use APSME-ESTABLISH-KEY.request to set up a link key with the device and shall respond to request for link key establishment using the APSME-ESTABLISH-KEY.response.

The trust center shall then provide the device with the NWK key using APSME-TRANSPORT-KEY.request.   It shall also provide the NWK key upon receiving a request from the device via the APSME-REQUEST-KEY.indication.

The trust center shall support the establishment of link keys between any two devices by providing them with a common key. Upon receipt of a APSME-REQUEST-KEY.indication requesting an application key, the trust center shall create a master key or link key and transport it to both devices using the APSME-TRANSPORT-KEY.request.

The trust center shall periodically update the NWK key according to a policy whose details are beyond the scope of this specification. All devices on the network shall be updated with the new NWK key using the APSME-TRANSPORT-KEY.request.

2.5.5.5.2 ZigBee Router

2.5.5.5.2.1 Initialization

Provision shall be made within the implementation to supply a single copy of desired network configuration parameters (:Config_NWK_Mode_and_Params) to the Network Object of ZigBee Device Objects. 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 sub-clause 2.5.2.1.

If supported, provision shall be made to supply configuration elements for the Complex Descriptor, User Descriptor, the maximum number of bind entries and the master key. These elements shall be embodied in :Config_Complex_Descriptor, :Config_User_Descriptor, :Config_Max_Bind and :Config_Master_Key.

The device application shall use NLME-NETWORK-DISCOVERY.request with the ChannelList portion of :Config_NWK_Mode_and_Params and then use the NLME-NETWORK-DISCOVERY.request attribute to scan the specified channels. The resulting NLME-NETWORK-DISCOVERY.confirm shall supply a NetworkList detailing active PANs within that range. 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. The device application shall compare the ChannelList to the NetworkList and select an existing PAN to join. 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, 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 modify its :Config_NWK_Mode_and_Params protocol version number field and 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.

Once the PAN to join is identified, the device application shall employ the NLME-JOIN.request to join the PAN on that channel. The device application shall check the return status via the NLME-JOIN.confirm to verify association to the selected ZigBee Router or ZigBee Coordinator on that PAN. The Extended PAN ID field from the beacon payload of the PAN shall be set within *nwkExtendedPANID*. 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 parameters shall be set with :Config_NWK_BroadcastDeliveryTime and :Config_NWK_TransactionPersistenceTime respectively (see Chapter 2).

Provision shall be made to ensure APS primitive calls from the end applications over EP 1 through EP 240 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 the trust center to supply it with a master key via the APSME-TRANSPORT-KEY.indication and then respond to a request from the trust center to establish a link key using the APSME-ESTABLISH-KEY.request. The device shall then wait for the trust center to provide it with a NWK key using APSME-TRANSPORT-KEY.indication. Upon successful acquisition of the NWK key, the device is authenticated and can start functioning as a router in the network

The device application shall set the *nwkSecurityLevel* and *nwkSecureAllFrames* NIB attributes to the values used in the network and begin functioning as a router using NLME-START-ROUTER.req.

2.5.5.5.2.2 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 2.5.5.5.4.

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.

If security is supported, the ZigBee router shall support the :Config_Master_Key and shall employ the Master Key in key establishment procedures for Link Keys.

Upon presentation of a remote destination address requiring secure communications, the ZigBee router shall support APSME-ESTABLISH-KEY.request to establish a link key with the remote device and APSME-ESTABLISH-KEY.indication to present the request to the destination and shall support APSME-ESTABLISH-KEY.confirm and APSME-ESTABLISH-KEY.response to complete the key establishment of the Link Key. The ZigBee router shall provide the ability to store Link Keys for known destinations requiring secure communications and shall manage key storage of Link Keys. The ZigBee router shall support APSME-TRANSPORT-KEY.indication to receive keys from the trust center. The ZigBee router shall request the trust center to update its NWK key via the APSME-REQUEST-KEY.request.

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-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 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. When a ZigBee end device is asked to leave the network, the ReuseAddress parameter of the NLME-LEAVE.request primitive should have a value of TRUE indicating that the NWK layer may give the address formerly in use by the leaving device to another end device that joins subsequently.[15]

The ZigBee router shall process End_Device_annce messages from ZigBee End Devices. Upon receipt of an End_Device_annce, 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 End_Device_annce message. At minimum, any Binding Table entries held by the ZigBee router shall be checked. If a match is detected, the ZigBee router shall update its APS Information Block address map entries corresponding to the matched 64 bit IEEE address to reflect the updated 16 bit NWK address contained in the End_Device_annce.

The ZigBee router 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.

15. CCB #675

2.5.5.5.3 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 as holding their own source binding tables. If the stack profile specifies it, the primary binding table cache shall provide message reflection using the binding table. If the stack profile specifies that message reflection is not supported, binding entries shall only be kept for devices registered as holding their own source 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 source binding devices for one or more primary binding table caches. It shall support the bind management primitives for backup and restore of these tables.

2.5.5.5.4 Operations to Support Intra-PAN Portability

2.5.5.5.4.1 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-ROUTE-ERROR.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. This process closely mirrors 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. (See clause 2.5.5.5.4.2)

- Inform the rest of the network - if the short address changed than an End_Device_annce ZDO command is broadcast to indicate the updated short address. (See clause 2.4.3.1.11)

These steps are described in detail in the subsections below. The mechanism is illustrated for a ZED and ZR in Figure 2.96 and Figure 2.97, respectively.

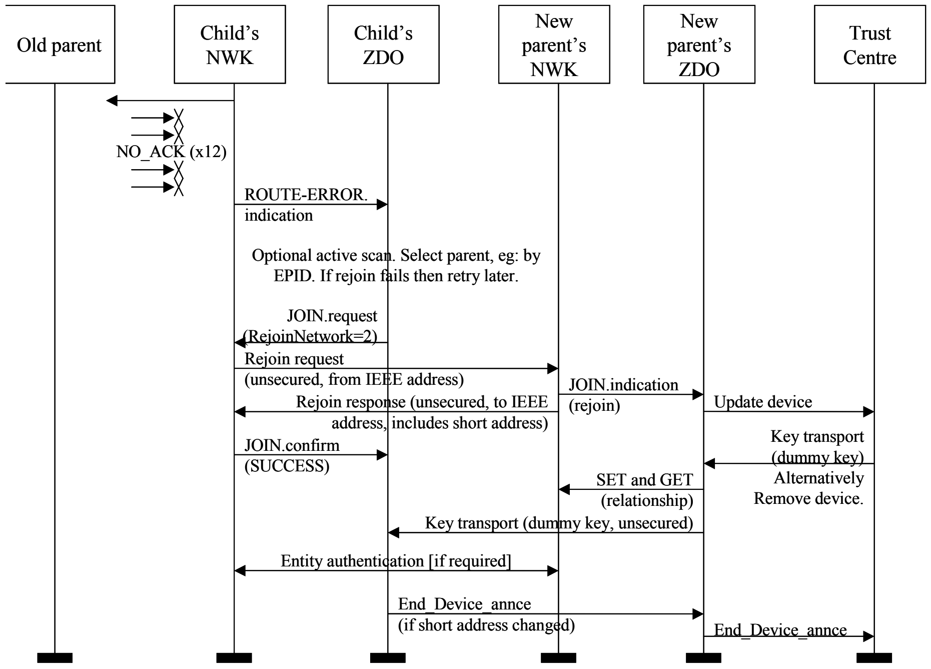

Old parent
Child's NWK
Child's ZDO
New parent's NWK
New parent's ZDO
Trust Centre
NO_ACK (x12)
ROUTE-ERROR. indication
Optional active scan. Select parent, eg: by EPID. If rejoin fails then retry later.
JOIN.request (RejoinNetwork=2)
Rejoin request (unsecured, from IEEE address)
JOIN.indication (rejoin)
Update device
Rejoin response (unsecured, to IEEE address, includes short address)
JOIN.confirm (SUCCESS)
Key transport (dummy key)
Alternatively Remove device.
SET and GET (relationship)
Key transport (dummy key, unsecured)
Entity authentication [if required]
End_Device_annce (if short address changed)
End_Device_annce

Figure 2.96 Portability Message Sequence Chart: ZED Rejoin

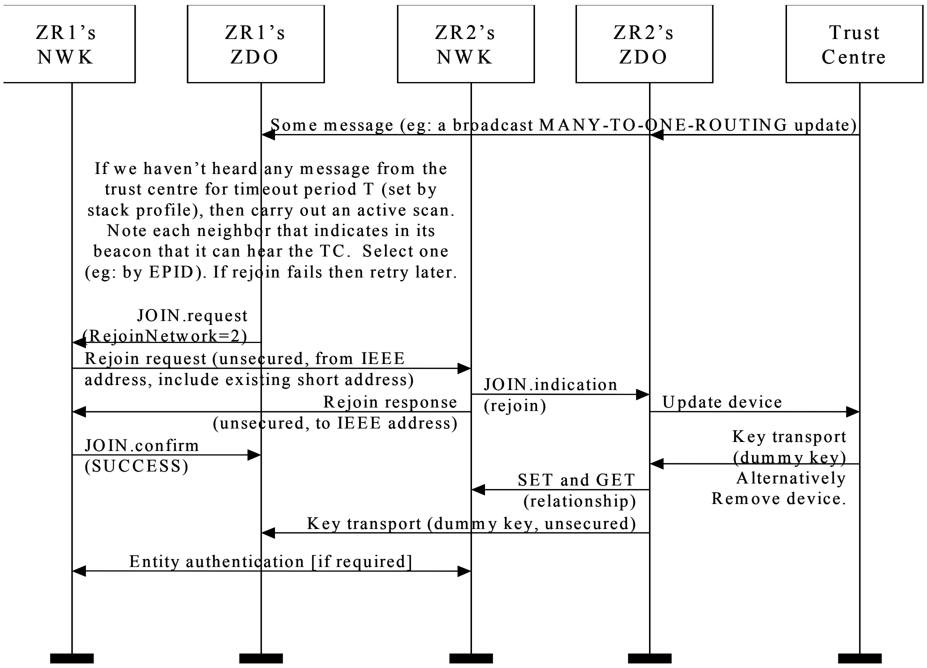

ZR1's NWK
ZR1's ZDO
ZR2's NWK
ZR2's ZDO
Trust Centre
Some message (eg: a broadcast MANY-TO-ONE-ROUTING update)
If we haven't heard any message from the trust centre for timeout period T (set by stack profile), then carry out an active scan. Note each neighbor that indicates in its beacon that it can hear the TC. Select one (eg: by EPID). If rejoin fails then retry later.
JOIN.request (RejoinNetwork=2)
Rejoin request (unsecured, from IEEE address, include existing short address)
JOIN.indication (rejoin)
Update device
Rejoin response (unsecured, to IEEE address)
JOIN.confirm (SUCCESS)
Key transport (dummy key)
Alternatively Remove device.
SET and GET (relationship)
Key transport (dummy key, unsecured)
Entity authentication [if required]

Figure 2.97 Portability Message Sequence Chart: ZR Rejoin

2.5.5.5.4.2 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 handled in the same way as for an association indication.

Note that due to the absence of an entity authentication this child status is initially provisional, and 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 envisaged:

- One or other device 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 other device is a compromised/hacked device. In the case that security is enabled, the measures in sub-clause 4.7.3.6 are additionally applied so that an unauthorized join is revoked.

This verification is carried out using existing commands. Sub-clause 2.5.5.5.4.3 below describes the transmission of an End_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. Correspondingly 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.

2.5.5.5.4.3 Description of Operations for Informing the Rest of the Network

If the ZigBee End Device gets a new short address after rejoining a new parent using the orphaning process, it shall issue an End_Device_annce providing its 64 bit IEEE address, new 16 bit NWK address and its capability information. The message is broadcast to all devices in the network. Upon receiving the End_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 End_Device_annce message. At minimum, any Binding Table shall be checked. If a match is detected, the device shall update its APS Information Block address map entries corresponding to the matched 64 bit IEEE address to reflect the updated 16 bit NWK address contained in the End_Device_annce. In addition, 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-ROUTE-ERROR.indication primitive with status code of "Target Device Unavailable", (see sub-clause 3.3.12.1). As address conflict detection is not provided, parent devices are not permitted, following intra-PAN portability, remove device or any other operation, to reissue a short address for use by a child with a different IEEE address. Also, 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.

2.5.5.5.5 ZigBee End Device

2.5.5.5.5.1 Initialization

Provision shall be made within the implementation to supply a single copy of desired network configuration parameters (:Config_NWK_Mode_and_Params) to the Network Object of ZigBee Device Objects.

If supported, provision shall be made to supply configuration elements for the Complex Descriptor, User Descriptor, the maximum number of bind entries and the master key. These elements shall be embodied in :Config_Complex_Descriptor, :Config_User_Descriptor, :Config_Max_Bind and :Config_Master_Key. If the device application set the NLME-JOIN RxOnWhenIdle parameter to FALSE, the end device shall utilize the procedure described in sub-clause 2.5.2.1 to discover a Primary Discovery Cache device, register with it and to successfully upload its device and service discovery information prior to turning off its receiver as a sleeping end device. 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.

The device application shall use NLME-NETWORK-DISCOVERY.request with the ChannelList portion of :Config_NWK_Mode_and_Params and then use the NLME-NETWORK-DISCOVERY.request attribute to scan the specified channels. The resulting NLME-NETWORK-DISCOVERY.confirm shall supply a NetworkList detailing active PANs within that range. 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. The device application shall compare the ChannelList to the NetworkList and select an existing PAN to join. Specification of the algorithm for selection of the PAN shall be left to the profile descriptions and may include use of the 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 modify its :Config_NWK_Mode_and_Params protocol version number field and 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.

Once the PAN to join is identified, the device application shall employ the NLME-JOIN. request to join the PAN on that channel. The device application shall check the return status via the NLME-JOIN.confirm to verify association to the selected ZigBee Router or ZigBee Coordinator on that PAN. The Extended PAN ID field from the beacon payload of the PAN shall also be set within *nwkExtendedPANID*.

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 an End_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 240 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 the trust center to supply it with a master key via the APSME-TRANSPORT-KEY.indication and then respond to a request from the trust center to establish a link key using the APSME-ESTABLISH-KEY.response. The device shall then wait for the trust center to provide it with a NWK key using APSME-TRANSPORT-KEY.indication. Upon successful acquisition of the NWK key, the device is joined and authenticated.

2.5.5.5.5.2 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.

The ZigBee end device shall respond to any device discovery or service discovery operations requested of its own device using the attributes described in sub-clause 2.5.4.

If security is enabled, the ZigBee end device shall support the :Config_Master_Key and shall employ the Master Key in key establishment procedures for Link Keys. Upon presentation of a remote destination address requiring secure communications, the ZigBee end device shall support APSME-ESTABLISH-KEY.request to establish a link key with the remote device and support APSME-ESTABLISH-KEY.indication to present the request to the destination and shall support APSME-ESTABLISH-KEY.confirm and APSME-ESTABLISH-KEY.response to complete the key establishment of the Link Key. The ZigBee end device shall provide the ability to store Link Keys for known destinations requiring secure communications and shall manage key storage of Link Keys. The ZigBee end device shall support APSME-TRANSPORT-KEY.indication to receive keys from the trust center. The ZigBee end device shall request the trust center to update its NWK key via the APSME-REQUEST-KEY.request.

The ZigBee End Device shall process End_Device_annce messages from other ZigBee End Devices. Upon receipt of an End_Device_annce, 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 End_Device_annce message. At minimum, any Binding Table entries held by the ZigBee End Device shall be checked. If a match is detected, the ZigBee End Device shall update its APS Information Block address map entries corresponding to the matched 64 bit IEEE address to reflect the updated 16 bit NWK address contained in the End_Device_annce.

The ZigBee End Device shall process the NLME-ROUTE-ERROR.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 gets a new short address after rejoining a new parent using the rejoin process, it shall issue an End_Device_annce providing its 64 bit IEEE address, new 16 bit NWK address and its capability information. The message is broadcast to all devices in the network.

2.5.5.6 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).

2.5.5.6.1 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 2.130 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 clause 2.4 for a description of any of these attributes.

Table 2.130 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
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
End_Device_annce	O	Public
User_Desc_set	O	Public
User_Desc_conf	O	Public
System_Server_Discovery_req	O	Public
System_Server_Discovery_rsp	O	Public

Table 2.130 Device and Service Discovery Attributes (Continued)

Attribute	M/O	Type
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
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	M	Public

2.5.5.7 Security Manager

The security manager determines whether security is enabled or disabled and, if enabled, shall perform the following:

- Establish Key
- Transport Key
- Authentication

2.5.5.7.1 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 clause 2.4 for a description of any of the primitives listed in Table 2.131.

Table 2.131 Security Manager Attributes

Attribute	M/O	Type
APSME-ESTABLISH-KEY.request	O	Public
APSME-ESTABLISH-KEY.response	O	Public
APSME-ESTABLISH-KEY. indication	O	Public
APSME-ESTABLISH-KEY.confirm	O	Public
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
APSME-REQUEST-KEY.request	O	Public
APSME-REQUEST-KEY.indication	O	Public
APSME-SWITCH-KEY.request	O	Public
APSME-SWITCH-KEY.indication	O	Public

2.5.5.8 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 or router, supporting the binding table.

2.5.5.8.1 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. Additionally, responses shall be supported on the ZigBee Coordinator or responses shall be supported on ZigBee Routers and ZigBee End 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 2.132 summarizes Binding Manager attributes.

Table 2.132 Binding Manager Attributes

Attribute	Method	M/O	Type
End_Device_Bind_req	See clause 2.4.	O	Public
End_Device_Bind_rsp	See clause 2.4.	O	Public
Bind_req	See clause 2.4.	O	Public
Bind_rsp	See clause 2.4.	O	Public
Unbind_req	See clause 2.4.	O	Public
Unbind_rsp	See clause 2.4.	O	Public
Bind_Register_req	See clause 2.4.	O	Public
Bind_Register_rsp	See clause 2.4.	O	Public
Replace_Device_req	See clause 2.4.	O	Public
Replace_Device_rsp	See clause 2.4.	O	Public
Store_Bkup_Bind_Entry_req	See clause 2.4.	O	Public
Store_Bkup_Bind_Entry_rsp	See clause 2.4.	O	Public
Remove_Bkup_Bind_req	See clause 2.4.	O	Public
Remove_Bkup_Bind_rsp	See clause 2.4.	O	Public
Backup_Bind_Table_req	See clause 2.4.	O	Public
Backup_Bind_Table_rsp	See clause 2.4.	O	Public
Recover_Bind_Table_req	See clause 2.4.	O	Public
Recover_Bind_Table_rsp	See clause 2.4.	O	Public
Backup_Source_Bind_req	See clause 2.4.	O	Public
Backup_Source_Bind_rsp	See clause 2.4.	O	Public
Recover_Source_Bind_req	See clause 2.4.	O	Public
Recover_Source_Bind_rsp	See clause 2.4.	O	Public
APSME-BIND.request	See clause 2.2.	O	Private
APSME-BIND.confirm	See clause 2.2.	O	Private
APSME-UNBIND.request	See clause 2.2.	O	Private
APSME-UNBIND.confirm	See clause 2.2.	O	Private

2.5.5.9 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

Network Management performs the above functions with NLME-SAP primitives (see Chapter 3).

2.5.5.9.1 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, and NWK Permit Joining confirm shall be supported. The NWK Direct Join request, NWK Direct Join confirm, NWK Route Discovery request and NWK Route Discovery confirm may be supported. The NWK Join request and the NWK Join response shall not be supported.

If the ZigBee logical device type is ZigBee Router, the NWK Formation request and confirm shall not be supported. 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 and NWK Permit Joining confirm shall be supported. The NWK Direct Join request, NWK Direct Join confirm, NWK Route Discovery request and NWK Route Discovery 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, NWK Leave indication and NWK Permit Joining request shall not be supported. The NWK Join request, NWK Join confirm, NWK Leave request, 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 2.133 summarizes Network Manager Attributes. See Chapter 3 for a description of any of the primitives listed in Table 2.133.

For all ZigBee logical device types, reception of the NWK Route Error indication shall be supported, but no action is required in this version of the specification.

Table 2.133 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[a]	O	Private
NLME-JOIN.request	O	Private
NLME-JOIN.confirm	O	Private
NLME-PERMIT-JOINING.request	O	Public
NLME-PERMIT-JOINING.confirm[b]	O	Public
NLME-DIRECT-JOIN.request	O	Public
NLME-DIRECT-JOIN.confirm	O	Public
NLME_LEAVE.request	M	Public
NLME-LEAVE.confirm	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

Table 2.133 Network Manager Attributes (Continued)

Attribute	M/O	Type
NLME-ROUTE-ERROR.indication	M	Private
NLME-ROUTE-DISCOVERY.request	O	Public
NLME-ROUTE-DISCOVERY.confirm	O	Private

a. CCB #563

b. CCB #563

2.5.5.10 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.

2.5.5.10.1 Optional and Mandatory Attributes Within Node Manager

The Node Manager is an optional object for all ZigBee Device Types. All request and response attributes within Node Manager are also optional if the Node Manager object is present. Table 2.134 summarizes Node Manager attributes. See clause 2.4 for a description of these attributes.

Table 2.134 Node Manager Attributes

Attribute	M/O	Type
Mgmt_NWK_Disc_req	O	Public
Mgmt_NWK_Disc_rsp	O	Public
Mgmt_Lqi_req	O	Public
Mgmt_Lqi_rsp	O	Public
Mgmt_Rtg_req	O	Public
Mgmt_Rtg_rsp	O	Public
Mgmt_Bind_req	O	Public
Mgmt_Bind_rsp	O	Public
Mgmt_Leave_req	O	Public
Mgmt_Leave_rsp	O	Public
Mgmt_Direct_Join_req	O	Public
Mgmt_Direct_Join_rsp	O	Public
Mgmt_Permit_Joining_req	O	Public

Table 2.134 Node Manager Attributes

Mgmt_Permit_Joining_rsp	O	Public
Mgmt_Cache_req	O	Public
Mgmt_Cache_req	O	Public

2.5.6 Configuration Attributes

This attribute is used to represent the minimum mandatory and/or optional attributes used as configuration attributes for a device.

Table 2.135 Configuration Attributes

Attribute	M/O	Type
:Config_Node_Descriptor	M	Public
:Config_Power_Descriptor	M	Public
:Config_Simple_Descriptors	M	Public
:Config_NWK_Mode_and_Params	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_Master_Key	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

Table 2.135 Configuration Attributes (Continued)

Attribute	M/O	Type
:Config_Parent_Link_Retry_Threshold	O	Public
:Config_Rejoin_Interval	O	Public
:Config_Max_Rejoin_Interval	O	Public

2.5.6.1 Configuration Attribute Definitions

Table 2.136 Configuration Attribute Definitions

Attribute	Description	When Updated
:Config_Node_Descriptor	Contents of the Node Descriptor for this device (see sub-clause 2.3.2.4).	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 sub-clause 2.3.2.5).	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 sub-clause 2.3.2.6).	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.

Table 2.136 Configuration Attribute Definitions (Continued)

Attribute	Description	When Updated
:Config_NWK_Mode_and_Params	Consists of the following fields: Channel List – The list of channels to be scanned when using NLME-NETWORK-DISCOVERY. Extended PAN ID - Used in NLME-NETWORK-FORMATION and NLME-JOIN (Chapter 3) Protocol Version – Used in NLME-NETWORK-FORMATION and NLME-JOIN (Chapter 3). Stack Profile – Used in NLME-NETWORK-FORMATION and NLME-JOIN (Chapter 3). Beacon Order – Used in NLME-NETWORK-FORMATION (Chapter 3). Superframe Order – Used in NLME-NETORK-FORMATION (Chapter 3). BatteryLifeExtension – TRUE or FALSE (sub-clause 3.3.3)	The :Config_Node_Descriptor contains a field describing this device's Logical Device Type. That information plus the specific logic employed in this device's ZDO permits the device application to use the parameters in :Config_NWK_Mode_and_Params to form or join a network consistent with the applications supported on the device.

Table 2.136 Configuration Attribute Definitions (Continued)

Attribute	Description	When Updated
: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 sub-clause 2.5.5.5). 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).
:Config_NWK_Time_btwn_ Scans	Integer value representing the time duration (in seconds) between each NWK discovery attempt described by :Config_NWK_Scan_ Attempts (see sub-clause 2.5.5.5). This attribute has a default value of 1 (second) and valid values between 1 and 255 (seconds).	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 sub-clause 2.3.2.7).	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 sub-clause 2.3.2.8).	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.

Table 2.136 Configuration Attribute Definitions (Continued)

Attribute	Description	When Updated
:Config_Max_Bind	A constant which describes the maximum number of binding entries permitted if this device is a ZigBee Coordinator or ZigBee Router.	The :Config_Max_Bind is a maximum number of supported Binding Table entries for this device.
:Config_Master_Key	Master Key used if security is enabled for this device (see Chapter 4).	The :Config_Master_Key is either present 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 security operations on the device if security is supported and enabled.
:Config_EndDev_Bind_Timeout	Timeout value in seconds employed in End Device Binding (see sub-clause 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.
: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 parameter is found in the Stack Profile employed.

Table 2.136 Configuration Attribute Definitions (Continued)

Attribute	Description	When Updated
:Config_NWK_BroadcastDeliveryTime	See Table 3.1.	The value for this configuration attribute is established in the Stack Profile.
:Config_NWK_TransactionPersistenceTime	See Table 3.41. 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 parameter 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.
:Config_NWK_indirectPollRate	Sets the poll rate 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 (such as with the Home Controls stack profile).

Table 2.136 Configuration Attribute Definitions (Continued)

Attribute	Description	When Updated
: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 MasterKey - If security is enabled, master key for use in the key-pair descriptor for this new device (see Table 4.29) See sub-clause 3.3.7.1 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 sub-clause 2.5.5.5.5.2)	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 sub-clause 2.5.5.5.5.2)	The :Config_Rejoin_Interval is either created when the application is first loaded or initialized with a commissioning tool. It is used for the ZED to decide how often it should initiate the rejoin process.
:Config_MAX_Rejoin_Interval	Contents of the maximal rejoin interval 2.5.5.5.5.2)	The :Config_MAX_Rejoin_Interval is either created when the application is first loaded or initialized with a commissioning tool. It is used for the ZED to decide how often it should initiate the rejoin process.

CHAPTER

3

NETWORK SPECIFICATION

3.1 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 3.1.

Table 3.1 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

Table 3.1 NWK Layer Status Values (Continued)

Name	Value	Description
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
LEAVE_UNCONFIRMED	0xcb	A device failed to confirm its departure from the network
MAX_FRM_CNTR	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
BAD_CCM_OUTPUT	0xce	Security processing has been attempted on an outgoing frame, and has failed because security engine produced erroneous output
NO_ROUTING CAPACITY	0xcf	An attempt to discover a route has failed due to a lack of routing table or discovery table capacity
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
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	A non-member mode multicast frame was discarded pending route discovery

3.2 General Description

3.2.1 Network (NWK) Layer Overview

The network layer is required to provide functionality to ensure correct operation of the IEEE 802.15.4-2003 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).

3.2.1.1 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.

3.2.1.2 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:** 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:** The ability to establish a new network.
- **Joining and leaving a network**: The ability to join or leave a network as well as the ability for a ZigBee coordinator or ZigBee router to request that a device leave the network.
- **Addressing**: The ability of ZigBee coordinators and routers to assign addresses to devices joining the network.
- **Neighbor discovery**: The ability to discover, record, and report information pertaining to the one-hop neighbors of a device.
- **Route discovery**: The ability to discover and record paths through the network, whereby messages may be efficiently routed.

- **Reception control:** The ability for a device to control when the receiver is activated and for how long, enabling MAC sub-layer synchronization or direct reception.

3.3 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 though 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.

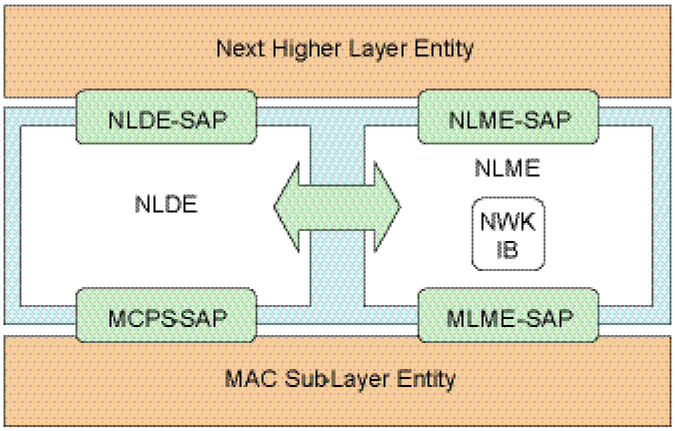


Figure 3.1 The NWK Layer Reference Model

3.3.1 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 3.2 lists the primitives supported by the NLDE-SAP and the sub-clauses in which these primitives are discussed.

Table 3.2 NLDE-SAP Primitives

NLDE-SAP Primitive	Request	Confirm	Indication
NLDE-DATA	3.3.1.1	3.3.1.2	3.3.1.3

3.3.1.1 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.

3.3.1.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLDE-DATA.request          {
                           DstAddrMode,
                           DstAddr,
                           NsduLength,
                           Nsdu,
                           NsduHandle,
                           Radius,
                           NonmemberRadius,
                           DiscoverRoute,
                           SecurityEnable
                           }
```

Table 3.3 specifies the parameters for the NLDE-DATA.request primitive.

Table 3.3 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 NWK address of a device or a 16-bit broadcast address
DstAddr	16-bit Address	0x0000-0xFFFF	Destination address
NsduLength	Integer	<aMaxMACFrameSize - 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
Radius	Unsigned Integer	0x00 – 0xff	The distance, in hops, that a frame will be allowed to travel through the network

Table 3.3 NLDE-DATA.request Parameters (Continued)

Name	Type	Valid Range	Description
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
DiscoverRoute	Integer	0x00 – 0x01	The DiscoverRoute parameter may be used to control route discovery operations for the transit of this frame (see sub-clause 3.7.3.4): 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 security level specified in the NIB is 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

3.3.1.1.2 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.

3.3.1.1.3 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 and 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 a value has been supplied for the Radius parameter, it will be placed in the radius field of the NWK header. If a value is not supplied, then the radius field of the NWK header will be set to twice the value of the *nwkMaxDepth* attribute of the NWK IB. The NWK layer will generate a sequence number for the frame as described in sub-clause 3.7.2.1. The sequence number value shall be inserted into the sequence number field of the NWK header of the frame. 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 sub-clause 3.7.3.3, if it is a unicast; sub-clause 3.7.5 if it is a broadcast; or sub-clause 3.7.6.1 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. The TxOptions parameter should always be non-zero when bitwise ANDed with the value 0x01, denoting that acknowledged transmission is required. On receipt of the MCPS-DATA.confirm primitive, the NLDE issues the NLDE-DATA.confirm primitive with a status equal to that received from the MAC sub-layer.

If the network-wide security level specified in the NIB 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 4.4. Otherwise, no security processing will be performed at the NWK layer for this frame. If security processing is done 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.

3.3.1.2 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.

3.3.1.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLDE-DATA.confirm                {
                                 NsduHandle,
                                 Status
                                 }
```

Table 3.4 specifies the parameters for the NLDE-DATA.confirm primitive.

Table 3.4 NLDE-DATA.confirm Parameters

Name	Type	Valid Range	Description
NsduHandle	Integer	0x00 – 0xff	The handle associated with the NSDU being confirmed
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

3.3.1.2.2 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 sub-clause 3.3.1.1.3.

3.3.1.2.3 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.

3.3.1.3 NLDE-DATA.indication

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

3.3.1.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLDE-DATA.indication         {
                             DstAddrMode,
                             DstAddr,
                             SrcAddr,
                             NsduLength,
                             Nsdu,
                             LinkQuality
                             }
```

Table 3.5 specifies the parameters for the NLDE-DATA.indication primitive.

Table 3.5 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 NWK address of a device or a 16-bit broadcast address
DstAddr	16-bit Address	0x0000-0xFFFF	The destination address where the NSDU is 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	< aMaxMACFrameSize – nwkcMinHeaderOverhead	The number of octets comprising the NSDU being indicated
Nsdu	Set of octets	–	The set of octets comprising the NSDU being indicated
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])

3.3.1.3.2 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.

3.3.1.3.3 Effect on Receipt

On receipt of this primitive, the APS sub-layer is notified of the arrival of data at the device.

3.3.1.3.4 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 3.6 lists the primitives supported by the NLME through the NLME-SAP interface and the sub-clauses containing details on each of these primitives.

Table 3.6 Summary of the Primitives Accessed Through the NLME-SAP

Name	Subclause Number in this Specification			
	Request	Indication	Response	Confirm
NLME-NETWORK-DISCOVERY	3.3.2.1			3.3.2.2
NLME-NETWORK-FORMATION	3.3.3.1			3.3.3.2
NLME-PERMIT-JOINING	3.3.4.1			3.3.4.2
NLME-START-ROUTER	3.3.5.1			3.3.5.2
NLME-JOIN	3.3.6.1	3.3.6.2		3.3.6.3
NLME-DIRECT-JOIN	3.3.7.1			3.3.7.2
NLME-LEAVE	3.3.8.1	3.3.8.2		3.3.8.3
NLME-RESET	3.3.9.1			3.3.9.2
NLME-SYNC	3.3.10.1	3.3.10.2		3.3.10.3
NLME-GET	3.3.11.1			3.3.11.2
NLME-SET	3.3.11.3			3.3.11.4
NLME-ROUTE-ERROR		3.3.12.1		
NLME-ROUTE-DISCOVERY	3.3.13.1			3.3.13.2

3.3.2 Network Discovery

The NWK layer management entity SAP (NLME-SAP) supports the discovery of operating networks. The primitives employed in network discovery are the NLME-NETWORK-DISCOVERY primitives.

3.3.2.1 NLME-NETWORK-DISCOVERY.request

This primitive allows the next higher layer to request that the NWK layer discover networks currently operating within the POS.

3.3.2.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-NETWORK-DISCOVERY.request      {
                                    ScanChannels,
                                    ScanDuration
                                    }
```

Table 3.7 specifies the parameters for the NLME-NETWORK-DISCOVERY.request primitive.

Table 3.7 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])

3.3.2.1.2 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).

3.3.2.1.3 Effect on Receipt

On receipt of this primitive, the NWK layer will attempt to discover networks operating within the device's POS by scanning over the channels specified in the ScanChannels argument for the period specified in the ScanDuration parameter.

If the device is an IEEE 802.15.4-2003 FFD [B1], then it will perform an active scan. If it is an RFD, it will perform an active scan provided that the device implements active scan. Otherwise, it will perform a passive scan using the MLME-SCAN.request primitive. On receipt of the MLME-SCAN.confirm primitive is assembled and 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.

3.3.2.2 NLME-NETWORK-DISCOVERY.confirm

This primitive reports the results of a network discovery operation.

3.3.2.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-NETWORK-DISCOVERY.confirm       {
                                     NetworkCount,
                                     NetworkDescriptor,
                                     Status
                                     }
```

Table 3.8 describes the arguments of the NLME-NETWORK-DISCOVERY.confirm primitive.

Table 3.8 NLME-NETWORK-DISCOVERY.confirm Parameters

Name	Type	Valid Range	Description
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 3.9 gives a detailed account of the contents of each item
Status	Status	Any Status value returned with the MLME-SCAN.confirm primitive	See [B1]

Table 3.9 gives a detailed account of the contents of a network descriptor from the NetworkDescriptor parameter.

Table 3.9 Network Descriptor Information Fields

Name	Type	Valid Range	Description
PanID	Integer	0x0000 – 0x3fff	The 16-bit PAN identifier of the discovered network; the 2 highest-order bits of this parameter are reserved and shall be set to 0
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 (see [B1])
PermitJoining	Boolean	TRUE or FALSE	A value of TRUE indicates that at least one ZigBee router on the network currently permits joining; That is, its NWK has been issued an NLME-PERMIT-JOINING primitive and, the time limit if given, has not yet expired

3.3.2.2.2 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.

3.3.2.2.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of a network search.

3.3.3 Network Formation

This set of primitives defines how the next higher layer of a device can initialize itself as the ZigBee coordinator of a new network and subsequently make changes to its superframe configuration.

3.3.3.1 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.

3.3.3.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-NETWORK-FORMATION.request        {
                                      ScanChannels,
                                      ScanDuration,
                                      BeaconOrder,
                                      SuperframeOrder,
                                      BatteryLifeExtension
                                      }
```

Table 3.10 specifies the parameters for the NLME-NETWORK-FORMATION.request primitive.

Table 3.10 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* * (2^n + 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

3.3.3.1.2 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 and subsequently make changes to its superframe configuration.

3.3.3.1.3 Effect on Receipt

On receipt of this primitive by a device that 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 the device is to be initialized as a ZigBee coordinator, 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. 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 0x0000 as the 16-bit short MAC address and inform the MAC sub-layer. 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 shall be initialized with the value of the MAC constant aExtendedAddress. 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.

To initialize a new superframe configuration or modify an existing one, 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 and SuperframeOrder given to the MLME-START.request primitive will be the same 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 initialize a new superframe. The CoordRealignment parameter is set to TRUE 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-NETWORK-FORMATION.confirm primitive to the next higher layer with the status returned from the MLME-START.confirm primitive.

3.3.3.2 NLME-NETWORK-FORMATION.confirm

This primitive reports the results of the request to initialize a ZigBee coordinator in a network.

3.3.3.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-NETWORK-FORMATION.confirm    {
                                  Status
                                  }
```

Table 3.11 specifies the parameters for the NLME-NETWORK-FORMATION.confirm primitive.

Table 3.11 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 or request a change to the superframe configuration

3.3.3.2.2 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 sub-clause 3.3.3.1.3.

3.3.3.2.3 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 or request a change to the superframe configuration. If the NLME has been successful, the status parameter will be set to SUCCESS. Otherwise, the status parameter indicates the error.

3.3.4 Allowing Devices to Join

This primitive defines how the next higher layer of a ZigBee coordinator or router can request that devices be permitted to join its network.

3.3.4.1 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.

3.3.4.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-PERMIT-JOINING.request    {
                               PermitDuration
                               }
```

Table 3.12 specifies the parameters for the NLME-PERMIT-JOINING.request primitive.

Table 3.12 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

3.3.4.1.2 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.

3.3.4.1.3 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.

3.3.4.2 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.

3.3.4.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-PERMIT-JOINING.confirm       {
                                  Status
                                  }
```

Table 3.13 specifies the parameters for the NLME-PERMIT-JOINING.confirm primitive.

Table 3.13 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

3.3.4.2.2 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 issues an error code of INVALID_REQUEST. The reasons for these status values are described in sub-clause 3.3.4.1.

3.3.4.2.3 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.

3.3.5 Begin as a Router

This set of primitives allows a ZigBee router that is newly joined to a network to begin engaging in the activities expected of a ZigBee router including the routing of data frames, route discovery, route repair and the accepting of requests to join the network from other devices. It also allows a ZigBee router to set up its superframe configuration. It may also be used by a ZigBee router to reconfigure its superframe.

3.3.5.1 NLME-START-ROUTER.request

This primitive allows the next higher layer of a ZigBee router to initialize or change its superframe configuration.

3.3.5.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-START-ROUTER.request    {
                             BeaconOrder,
                             SuperframeOrder,
                             BatteryLifeExtension
                             }
```

Table 3.14 specifies the parameters for NLME-START-ROUTER.request.

Table 3.14 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

3.3.5.1.2 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. It may also be issued to the NLME of a device that is already operating as a ZigBee router to adjust the configuration of its superframe.

3.3.5.1.3 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.

To initialize a new superframe configuration or to reconfigure an already existing one, the NLME issues the MLME-START.request primitive to the MAC sub-layer. The CoordRealignment parameter in the MLME-START.request primitive is set to FALSE if the primitive is issued to initialize a new superframe. The CoordRealignment parameter is set to TRUE 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, route repair and the accepting of requests to join the network from other devices. Otherwise, the device is expressly forbidden to engage in these activities.

3.3.5.2 NLME-START-ROUTER.confirm

This primitive reports the results of the request to initialize or change the superframe configuration of a ZigBee router.

3.3.5.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-START-ROUTER.confirm     {
                              Status
                              }
```

Table 3.15 specifies the parameters for NLME-START-ROUTER.confirm.

Table 3.15 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

3.3.5.2.2 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 sub-clause 3.3.5.1.3.

3.3.5.2.3 Effect on Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to initialize or change the superframe configuration of a ZigBee router. If the NLME has been successful, the status parameter will be set to SUCCESS. Otherwise, the status parameter indicates the error.

3.3.6 Joining a Network

This set of primitives defines how the next higher layer of a device can:

- Request to join a network through association
- Request to join a network directly
- Request to re-join a network if orphaned

3.3.6.1 NLME-JOIN.request

This primitive allows the next higher layer to request to join a network either through association or directly or to re-join a network if orphaned.

3.3.6.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-JOIN.request    {
                     ExtendedPANId,
                     JoinAsRouter,
                     RejoinNetwork,
                     ScanChannels,
                     ScanDuration,
                     PowerSource,
                     RxOnWhenIdle,
                     MACSecurity
                     }
```

Table 3.16 specifies the parameters for the NLME-JOIN.request primitive.

Table 3.16 NLME-JOIN.request Parameters

Name	Type	Valid Range	Description
ExtendedPANId	Integer	0x0000000000000001 – 0xfffffffffffffffe	The 64-bit PAN identifier of the network to join
JoinAsRouter	Boolean	TRUE or FALSE	The parameter is TRUE if the device is attempting to join the network in the capacity of a ZigBee router; Otherwise, it is FALSE; The parameter is valid in requests to join through association and ignored in requests to join directly or to re-join through orphaning
RejoinNetwork	Integer	0x00 – 0x02	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.
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]

Table 3.16 NLME-JOIN.request Parameters (Continued)

Name	Type	Valid Range	Description
PowerSource	Integer	0x00 – 0x01	This parameter becomes a part of the CapabilityInformation parameter passed to the MLME-ASSOCIATE.request primitive that is generated as the result of a successful executing of a NWK join. The values are: 0x01 = Mains-powered device 0x00 = other power source (see [B1])
RxOnWhenIdle	Boolean	TRUE or FALSE	This parameter indicates whether the device can be expected to receive packets over the air during idle portions of the CAP. The values are: TRUE = The receiver is enabled when the device is idle FALSE = The receiver may be disabled when the device is idle RxOnWhenIdle shall have a value of TRUE for ZigBee coordinators and ZigBee routers operating in a non-beacon-oriented network.
MACSecurity	Integer	0x00 – 0x01	This parameter becomes a part of the CapabilityInformation parameter passed to the MLME-ASSOCIATE.request primitive that is generated as the result of a successful executing of a NWK join; The values are: 0x01 = MAC security enabled 0x00 = MAC security disabled (see [B1])

3.3.6.1.2 When Generated

The next higher layer of a device generates this primitive to request to:

- Join a new network using the MAC sub-layer association procedure
- Join a new network directly using the MAC sub-layer orphaning procedure, or
- Locate and re-join a network after being orphaned.

3.3.6.1.3 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 ExtendedPANId parameter. The NLME issues an MLME-ASSOCIATE.request with its CoordAddress parameter set to the address of a router for which following conditions are true:

- The router belongs to the network identified by the ExtendedPANId parameter
- The router 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 sub-clause 3.7.3.1

If a device exists in the neighbor table for which these conditions are true, the LogicalChannel parameter of the MLME-ASSOCIATE.request primitive is 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 3.17 and the capability information assembled here shall be stored as the value of the *nwkCapabilityInformation* NIB attribute (see Table 3.41). If more than one device meets these requirements and the *nwkUseTreeAlloc* attribute of the NIB has a value of TRUE, then the joining device shall select the parent with the smallest tree depth.

Table 3.17 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 and the JoinAsRouter parameter has a value of TRUE; 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 shall 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

Table 3.17 Capability Information Bit-fields (Continued)

Bit	Name	Description
3	Receiver on when idle	This field shall 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 be set to the value of lowest-order bit of the MACSecurity parameter passed to the NLME-JOIN-request primitive. The values are: 0x01 = MAC security enabled 0x00 = MAC security disabled
7	Allocate address	This field will always have a value of 1 in implementations of this specification, indicating that the joining device must be issued a 16-bit short address

If no device exists in the neighbor table for which the conditions are true, then the NWK layer will issue an NLME-JOIN.confirm with the Status parameter set to NOT_PERMITTED. 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.

If a device receives this primitive and the RejoinNetwork parameter is equal to 0x01 then it issues an MLME-SCAN.request with the ScanType parameter set to indicate an orphan scan. Values for the ScanChannels and ScanDuration parameters of the MLME-SCAN.request primitive will equal those provided by the NLME-JOIN.request primitive. Upon receipt of the MLME-SCAN.confirm primitive, the NLME issues the NLME-JOIN.confirm with the Status parameter set to NO_NETWORKS, if the device was unable to find a network to join, or else to the status parameter value returned by the MLME-SCAN.confirm primitive.

On receipt of this primitive by a device that is currently not joined to a network and the RejoinNetwork parameter is equal to 0x02, 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 currently joined to a network, with the RejoinNetwork parameter is equal to 0x02, the device attempts to rejoin its current network. In this case the NLME initiates the rejoin procedure by sending a rejoin request command to the address of a router in its neighbor table for which following conditions are true:

1. The router is advertising capacity to accept the device type defined by the JoinAsRouter parameter.
2. 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 sub-clause 3.7.3.1.
3. If the *nwkUseTreeAddrAlloc* NIB attribute has a value of TRUE and more than one potential parent exists that meets the previous two conditions, then the joining device shall select the address of the router with the smallest tree depth.

If a device exists in the neighbor table for which these conditions are true, the destination of the rejoin request command is set to the network address of the potential parent. The bit-fields of the CapabilityInformation parameter shall have the values shown in Table 3.17 and the capability information assembled here shall be stored as the value of the *nwkCapabilityInformation* NIB attribute (see Table 3.41).

If no device exists in the neighbor table for which the conditions are true, then the NWK layer will issue an NLME-JOIN.confirm with the Status parameter set to NOT_PERMITTED. Otherwise, the NLME issues the NLME-JOIN.confirm with the Status parameter set to the status parameter value received in the rejoin response command.

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 is joined.

3.3.6.2 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 sub-clause 3.7.1.3.3.

3.3.6.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-JOIN.indication     {
                         ShortAddress,
                         ExtendedAddress,
                         CapabilityInformation,
                         SecureJoin
                         }
```

Table 3.18 specifies the parameters for the NLME-JOIN.indication primitive.

Table 3.18 NLME-JOIN.indication Parameters

Name	Type	Valid Range	Description
ShortAddress	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
SecureJoin	Boolean	TRUE or FALSE	This parameter will be TRUE if the underlying MAC association was performed in a secure manner; Otherwise this parameter will be FALSE

3.3.6.2.2 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 3.24, or on allowing a device to rejoin the network using the NWK rejoining procedure as shown in Figure 3.29.

3.3.6.2.3 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.

3.3.6.3 NLME-JOIN.confirm

This primitive allows the next higher layer to be notified of the results of its request to join a network.

3.3.6.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-JOIN.confirm             {
                              ShortAddress,
                              PANId,
                              HaveNetworkKey
                              Status
                              }
```

Table 3.19 specifies the parameters for the NLME-JOIN.confirm primitive.

Table 3.19 NLME-JOIN.confirm Parameters

Name	Type	Valid Range	Description
ShortAddress	Integer	0x0001 – 0xFFFF	The 16-bit short address that was allocated to this device; This parameter will be equal to 0xFFFF if the join attempt was unsuccessful
PANId	Integer	0x0000 – 0x3fff	The PAN identifier for the network of which the device is now a member.
HaveNetwork Key	Boolean	TRUE or FALSE	The parameter is TRUE if this device and its parent are known to have the same network key sequence number and FALSE otherwise.
Status	Status	INVALID_REQUEST, NOT_PERMITTED, NO_NETWORKS or any status value returned from the MLME-ASSOCIATE.confirm primitive or the MLME-SCAN.confirm primitive	The status of the corresponding request

3.3.6.3.2 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 shall 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 or the MLME-SCAN.confirm primitive. The reasons for these status values are fully described in sub-clause 3.3.6.1.3.

3.3.6.3.3 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, in order to join directly using the MAC sub-layer orphaning procedure or to re-join a network once it has been orphaned.

3.3.7 Joining a Device Directly to a Network

This set of optional primitives defines how the next higher layer of a ZigBee coordinator or router can request to directly join another device to its network.

3.3.7.1 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.

3.3.7.1.1 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 3.17	The operating capabilities of the device being directly joined

3.3.7.1.2 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.

3.3.7.1.3 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.

3.3.7.2 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.

3.3.7.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-DIRECT-JOIN.confirm      {
                              DeviceAddress,
                              Status
                              }
```

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
DeviceAddress	64-bit IEEE address	Any 64-bit, IEEE address	The 64-bit IEEE address in the request to which this is a confirmation
Status	Status	SUCCESS, ALREADY_PRESENT, NEIGHBOR_TABLE_FULL	The status of the corresponding request

3.3.7.2.2 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 sub-clause 3.3.7.1.3.

3.3.7.2.3 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.

3.3.8 Leaving a Network

This set of primitives defines how the next higher layer of a device can request to leave or request that a child device leaves a network. This set of primitives also defines how the next higher layer of a ZigBee coordinator device can be notified of an attempt by a device to leave its network.

3.3.8.1 NLME-LEAVE.request

This primitive allows the next higher layer to request that it or another device leaves the network.

3.3.8.1.1 Semantics of the Service Primitive

This semantics of this primitive are as follows:

```
NLME-LEAVE.request          {
                            DeviceAddress,
                            RemoveChildren,
                            Rejoin,
                            ReuseAddress,
                            Silent
                            }
```

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 a 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
ReuseAddress	Boolean	TRUE or FALSE	In the case where the DeviceAddress parameter has a non-NULL value, indicating that another device is being asked to leave the network, this parameter has a value of TRUE if the NWK layer may reuse the address formerly in use by the leaving device and FALSE otherwise.
Silent	Boolean	TRUE or FALSE	In the case where the DeviceAddress parameter has a non-NULL value, indicating that another device is being asked to leave the network, this parameter, if it has a value of TRUE, indicates that the leave procedure outlined here and in sub-clause 3.7.1.8.2 but without actually transmitting a leave command frame.

3.3.8.1.2 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.

3.3.8.1.3 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

NULL and the RemoveChildren parameter equal to FALSE, the NLME will remove the device itself from the network using the procedure described in sub-clause 3.7.1.8.1. The NLME will then clear its routing table and route discovery table and issue an MLME-RESET.request primitive to the MAC sub-layer. If the NLME receives an MLME-RESET.confirm primitive with the Status parameter set to anything other than SUCCESS, the NLME may choose to re-issue the reset request. 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 sub-clause 3.7.1.8.3.

On receipt of this primitive by a ZigBee coordinator or ZigBee router and with the DeviceAddress parameter not equal to NULL, the NLME determines whether the specified device is a child device. If the requested device does not exist, 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 sub-clause 3.7.1.8.2. 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. Then the relationship field for the neighbor table entry corresponding to the removed device will be updated. The relationship field will be updated according to the value of the Rejoin parameter from the NLME-LEAVE.request. If Rejoin is equal to FALSE then the relationship field is set to 0x03, indicating no relationship. If however Rejoin is equal to TRUE then the relationship field is set to 0x04, indicating that the address belonged to a previous child.

3.3.8.2 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.

3.3.8.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-LEAVE.indication          {
                               DeviceAddress,
                               Rejoin
                               }
```

Table 3.23 specifies 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 disassociate from the current parent is requested to rejoin the network; Otherwise, this parameter has a value of FALSE

3.3.8.2.2 When Generated

This primitive is generated by the NLME of a ZigBee coordinator or ZigBee router and issued to its next higher layer on the 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.

3.3.8.2.3 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 indicate that the next higher layer of a ZigBee router or end device is informed that it has been removed from the network by its associated ZigBee router or ZigBee coordinator.

If the Rejoin parameter is equal to TRUE, it is expected that the next higher layer will subsequently rejoin the network using the NLME-JOIN.request primitive and may employ the entire procedure outlined in sub-clause 3.7.1.3. If the Rejoin parameter is equal to FALSE, the leaving device should not automatically attempt to rejoin the network, although it may rejoin at a later point in response to instructions from a higher layer.

3.3.8.3 NLME-LEAVE.confirm

This primitive allows the next higher layer to be notified of the results of its request for itself or another device to leave the network.

3.3.8.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-LEAVE.confirm          {
                            DeviceAddress,
                            Status
                            }
```

Table 3.24 specifies the parameters for the NLME-LEAVE.confirm primitive.

Table 3.24 NLME-LEAVE.confirm Parameters

Name	Type	Valid Range	Description
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
Status	Status	SUCCESS, INVALID_REQUEST, UNKNOWN_DEVICE or any status returned by the MCPS-DATA.confirm primitive	The status of the corresponding request

3.3.8.3.2 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 sub-clause 3.3.8.1.3.

3.3.8.3.3 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.

3.3.9 Resetting a Device

This set of primitives defines how the next higher layer of a device can request that the NWK layer is reset.

3.3.9.1 NLME-RESET.request

This primitive allows the next higher layer to request that the NWK layer performs a reset operation.

3.3.9.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-RESET.request            {
                              }
```

This primitive has no parameters.

3.3.9.1.2 When Generated

This primitive is generated by the next higher layer and issued to its NLME to request the reset of the NWK layer to its initial condition.

3.3.9.1.3 Effect on Receipt

On receipt of this primitive, 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.

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.

3.3.9.2 NLME-RESET.confirm

This primitive allows the next higher layer to be notified of the results of its request to reset the NWK layer.

3.3.9.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-RESET.confirm            {
                              Status
                              }
```

Table 3.25 specifies the parameters for this primitive.

Table 3.25 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

3.3.9.2.2 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 indicates a successful reset attempt. Otherwise, the status parameter indicates an error code of DISABLE_TRX_FAILURE. The reasons for these status values are fully described in sub-clause 3.3.9.1.3.

3.3.9.2.3 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.

3.3.9.3 Network Layer Reset Message Sequence Chart

Figure 3.2 illustrates the sequence of messages necessary for resetting the NWK layer.

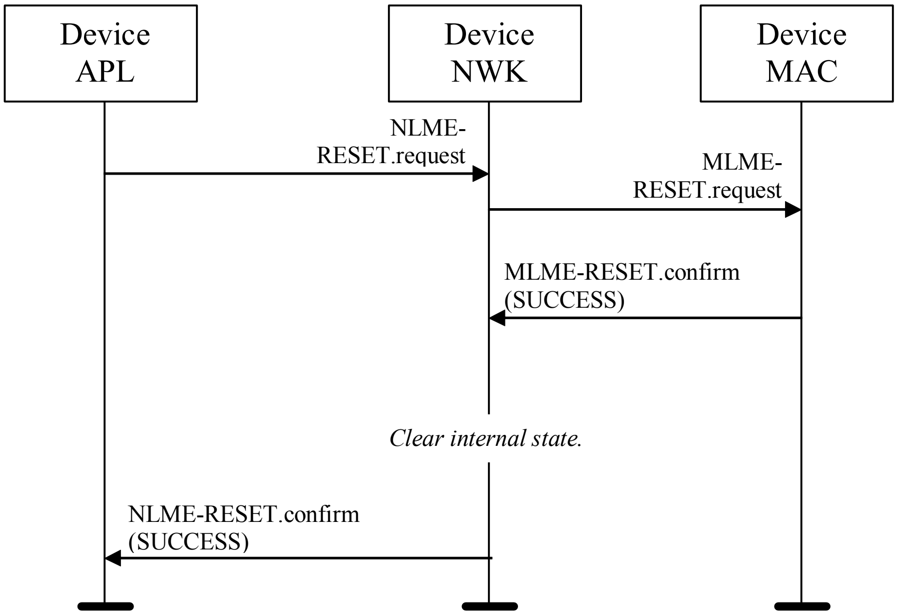


Figure 3.2 Message Sequence Chart for Resetting the Network Layer

3.3.10 Receiver Synchronization

This set of primitives defines how the next higher layer of a device can synchronize with a ZigBee coordinator or router and extract pending data from it.

3.3.10.1 NLME-SYNC.request

This primitive allows the next higher layer to synchronize or extract data from its ZigBee coordinator or router.

3.3.10.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-SYNC.request          {
                           Track
                           }
```

Table 3.26 specifies the parameters for this primitive.

Table 3.26 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

3.3.10.1.2 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.

3.3.10.1.3 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.

3.3.10.2 NLME-SYNC.indication

This primitive allows the next higher layer to be notified of the loss of synchronization at the MAC sub-layer.

3.3.10.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-SYNC.indication        {
                            }
```

This primitive has no parameters.

3.3.10.2.2 When Generated

This primitive is generated following 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.

3.3.10.2.3 Effect on Receipt

The next higher layer is notified of the loss of synchronization with the beacon.

3.3.10.3 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.

3.3.10.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-SYNC.confirm        {
                         Status
                         }
```

Table 3.27 specifies the parameters for this primitive.

Table 3.27 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

3.3.10.3.2 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 state change attempt. Otherwise, the status parameter indicates an error code. The reasons for these status values are fully described in sub-clause 3.3.10.1.3.

3.3.10.3.3 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.

3.3.10.4 Message Sequence Charts For Synchronizing with a Coordinator

Figure 3.3 and Figure 3.4 illustrate the sequence of messages necessary for a device to successfully synchronize with a ZigBee coordinator. Figure 3.3 illustrates the case for a non-beaconing network, and Figure 3.4 illustrates the case for a beaconing network.

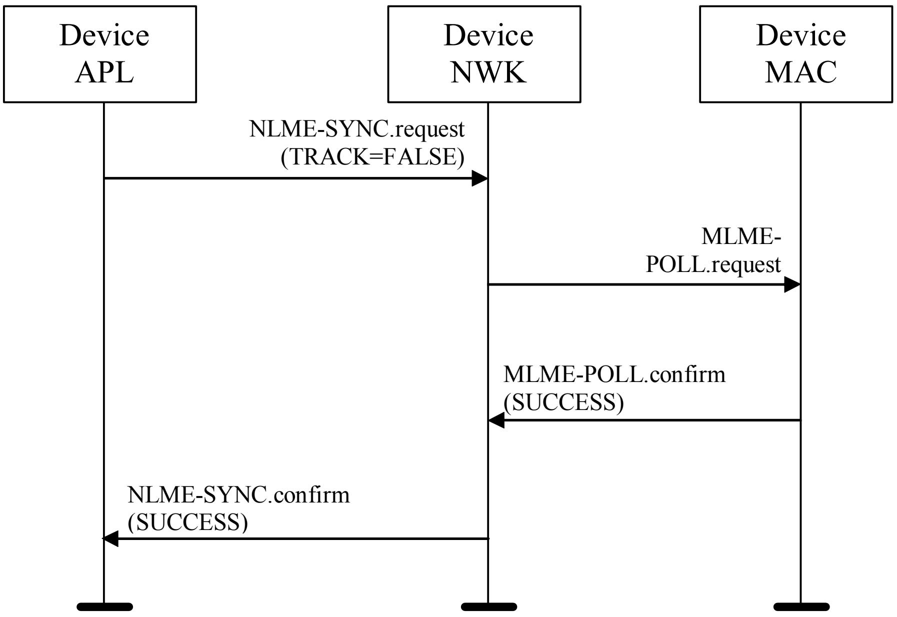

Device APL
Device NWK
Device MAC
NLME-SYNC.request (TRACK=FALSE)
MLME-POLL.request
MLME-POLL.confirm (SUCCESS)
NLME-SYNC.confirm (SUCCESS)

Figure 3.3 Message Sequence Chart for Synchronizing in a Non-Beaconing Network

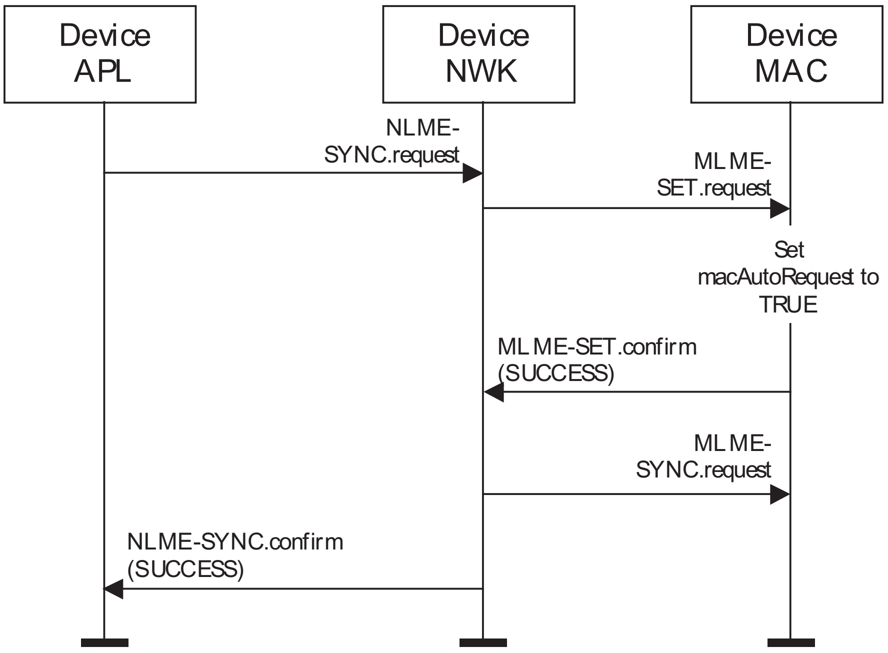


Figure 3.4 Message Sequence Chart for Synchronizing in a Beacon-Enabled Network

3.3.11 Information Base Maintenance

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

3.3.11.1 NLME-GET.request

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

3.3.11.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-GET.request          {
                          NIBAttribute
                          }
```

Table 3.28 specifies the parameters for this primitive.

Table 3.28 NLME-GET.request Parameters

Name	Type	Valid Range	Description
NIBAttribute	Integer	See Table 3.41	The identifier of the NIB attribute to read

3.3.11.1.2 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.

3.3.11.1.3 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.

3.3.11.2 NLME-GET.confirm

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

3.3.11.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-GET.confirm            {
                            Status,
                            NIBAttribute,
                            NIBAttributeLength,
                            NIBAttributeValue
                            }
```

Table 3.29 specifies the parameters for this primitive.

Table 3.29 NLME-GET.confirm Parameters

Name	Type	Valid Range	Description
Status	Enumeration	SUCCESS or UNSUPPORTED _ATTRIBUTE	The results of the request to read a NIB attribute value
NIBAttribute	Integer	See Table 3.41	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 3.41)	The value of the NIB attribute that was read.

3.3.11.2.2 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 sub-clause 3.3.11.1.3.

3.3.11.2.3 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.

3.3.11.3 NLME-SET.request

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

3.3.11.3.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-SET.request        {
                        NIBAttribute,
                        NIBAttributeLength,
                        NIBAttributeValue
                        }
```

Table 3.30 specifies the parameters for this primitive.

Table 3.30 NLME-SET.request Parameters

Name	Type	Valid Range	Description
NIBAttribute	Integer	See Table 3.41	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 3.41)	The value of the NIB attribute that should be written

3.3.11.3.2 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.

3.3.11.3.3 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.

3.3.11.4 NLME-SET.confirm

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

3.3.11.4.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-SET.confirm          {
                          Status,
                          NIBAttribute
                          }
```

Table 3.31 specifies the parameters for this primitive.

Table 3.31 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 3.41	The identifier of the NIB attribute that was written

3.3.11.4.2 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 sub-clause 3.3.11.3.3.

3.3.11.4.3 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 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.

3.3.12 Route Error Reporting

The primitive described here enables the NWK layer to inform higher layers on a particular device that a routing failure has occurred and, as a result, at least one unicast or multicast frame sent or relayed by that device has not been delivered as intended. Route errors for broadcast frames, that is, frames sent to one of the broadcast addresses listed in Table 3.32, are not reported.

3.3.12.1 NLME-ROUTE-ERROR.indication

This primitive allows the next higher layer to be notified of the failure of a communication path across the network.

3.3.12.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-ROUTE-ERROR.indication     {
                                ShortAddr,
                                Status
                                }
```

Table 3.32 specifies the parameters for this primitive.

Table 3.32 NLME-ROUTE-ERROR.indication Parameters

Name	Type	Valid Range	Description
ShortAddr	Integer	0x0000 – 0xFFF7	The 16-bit network address of the destination device associated with the routing failure
Status	Status	Any route error status code (see Table 3.39)	The error code associated with the routing failure

3.3.12.1.2 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 ShortAddr 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 ShortAddr parameter, due to one of the reasons given in Table 3.39
- The NWK layer has received a route error command frame destined for the device. In this case, the values of the ShortAddr and Status parameters will reflect the values of the destination address and error code fields of the command frame.

3.3.12.1.3 Effect on Receipt

The next higher layer is notified of a failure to communicate with the identified device.

3.3.13 Route Discovery

This set of primitives defines how the next higher layer can initiate route discovery of various kinds — specifically, unicast route discovery, multicast route discovery and many-to-one route discovery — and be informed of the results.

3.3.13.1 NLME-ROUTE-DISCOVERY.request

This primitive allows the next higher layer to initiate route discovery.

3.3.13.1.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-ROUTE-DISCOVERY.request                     {
                                                 DstAddrMode,
                                                 DstAddr,
                                                 Radius
                                                 }
```

Table 3.33 specifies the parameters for this primitive.

Table 3.33 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 NWK address of a multicast group 0x02 = 16-bit NWK address of an individual device
DstAddr	16-bit NWK Address	Any NWK 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 shall be the 16-bit NWK 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 NWK 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.

3.3.13.1.2 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.

3.3.13.1.3 Effect on Receipt

On receipt of this primitive by the NLME of a ZigBee end device, the NLME shall 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 shall 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 shall issue the NLME-ROUTE-DISCOVERY.confirm to the next higher layer with a Status value of NO_ROUTING_CAPACITY.

On receipt of this primitive on a ZigBee router or ZigBee coordinator that has routing capacity, with the DstAddrMode parameter equal to 0x02, the NLME shall initiate discovery of an unicast route between the current device and the network device with the 16-bit NWK address given by the DstAddr parameter. The procedure for initiating discovery of an unicast route is outlined in sub-clause 3.7.3.4.1.

On receipt of this primitive on a ZigBee router or ZigBee coordinator that has routing capacity, with the DstAddrMode parameter equal to 0x01, the NLME shall first check to see if the device is a member of the multicast group identified by the DstAddr parameter. That is, it will check if there is an entry in the nwkGroupIDTable 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 an unicast route between the current device and the multicast group having the identified by the DstAddr parameter. The procedure for initiating discovery of an unicast route is also outlined in sub-clause 3.7.3.4.1.

On receipt of this primitive on a ZigBee router or ZigBee coordinator with the DstAddrMode parameter equal to 0x00, the NLME shall initiate many-to-one route discovery. The procedure for initiating many-to-one route discovery is outlined in sub-clause 3.7.3.4.1.

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 NWK IB 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 else the route discovery operation times out as described in sub-clause 3.7.3.4. 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_DISCOVERY_FAILED.

3.3.13.2 NLME-ROUTE-DISCOVERY.confirm

This primitive informs the next higher layer about the results of an attempt to initiate route discovery.

3.3.13.2.1 Semantics of the Service Primitive

The semantics of this primitive are as follows:

```
NLME-ROUTE-DISCOVERY.confirm                    {
                                                Status
                                                }
```

Table 3.34 specifies the parameters for the NLME-ROUTE-DISCOVERY.confirm primitive.

Table 3.34 NLME-ROUTE-DISCOVERY.confirm Parameters

Name	Type	Valid Range	Description
Status	Status Value	INVALID_REQUEST, NO_ROUTING_CAPACITY, ROUTE_DISCOVERY_FAILED or any status value returned by the MCPS-DATA.confirm primitive	The status of an attempt to initiate route discovery

3.3.13.2.2 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.

3.3.13.2.3 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 sub-clause 3.3.13.1.3.

3.4 Frame Formats

This sub-clause 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 sub-clause are depicted in the order in which they are transmitted by the MAC sub-layer, from left to right, where the left-most bit is transmitted first in time. Bits within each field are numbered from 0 (left-most and least significant) to k-1 (right-most 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.

3.4.1 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. However, the multicast control field is present only if the multicast flag has the value 1. The NWK frame shall be formatted as illustrated in Figure 3.5.

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

Figure 3.5 General NWK Frame Format

3.4.1.1 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.

Bits: 0-1	2-5	6-7	8	9	10	11	12	13-15
Frame type	Protocol version	Discover route	Multicast flag	Security	Source Route	Destination IEEE Address	Source IEEE Address	Reserved

Figure 3.6 Frame Control Field

3.4.1.1.1 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 3.35.

Table 3.35 Values of the Frame Type Sub-field

Frame Type Value b_1 b_0	Frame Type Name
00	Data
01	NWK command
10 – 11	Reserved

3.4.1.1.2 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*.

3.4.1.1.3 Discover Route Sub-field

The discover route sub-field may be used to control route discovery operations for the transit of this frame (see sub-clause 3.7.3.4).

Table 3.36 Values of the Discover Route Sub-field

Discover Route Field Value	Field Meaning
0x00	Suppress route discovery
0x01	Enable route discovery
0x02	Force route discovery
0x03	Reserved

For NWK layer command frames, the discover route sub-field shall be set to 0x00 indicating suppression of route discovery.

3.4.1.1.4 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.

3.4.1.1.5 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.

3.4.1.1.6 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.

3.4.1.1.7 Destination IEEE Address Sub-field

The Destination IEEE address shall have a value of 1, if and only if the network header is to include the full IEEE address of the destination.

3.4.1.1.8 Source IEEE Address Sub-field

The Source IEEE address shall have a value of 1, if and only if the network header is to include the full IEEE address of the source device.

3.4.1.2 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). 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 always be the same as its IEEE 802.15.4-2003 MAC short address.

3.4.1.3 Source Address Field

The source address field shall always be present. It will 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 always be the same as its IEEE 802.15.4-2003 MAC short address.

3.4.1.4 Radius Field

The radius field shall always be present. It will be 1 octet in length and specifies the range of a radius transmission. The field shall be decremented by 1 by each receiving device.

3.4.1.5 Sequence Number Field

The sequence number field is present in every frame and is 1 octet in length. The sequence number value will be incremented by 1 with each new transmitted frame. 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 1-octet range. For more details on the use of the sequence number field, see sub-clause 3.7.2.

3.4.1.6 Destination IEEE Address Field

The destination IEEE address field is present if the corresponding sub-field in the frame control field is set and, if present, contains the 64-bit IEEE address of the eventual destination of the frame.

3.4.1.7 Source IEEE Address Field

The source IEEE address field is present if the corresponding sub-field in the frame control field is set and, if present, contains the 64-bit IEEE address of the source of the frame.

3.4.1.8 Multicast Control Field

The multicast control sub-field is 1 octet in length and is only present if the multicast flag sub-field has value 1. It is divided into three sub-fields as illustrated in Figure 3.7.

Bits: 0 – 1	**2 – 4**	**5 – 7**
Multicast mode	NonmemberRadius	MaxNonmemberRadius

Figure 3.7 Multicast Control Field Format

3.4.1.8.1 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 part of the multicast group.

Table 3.37 Values of the Multicast Mode Sub-field

Multicast Mode Field Value: b^0 b^1	Field Meaning
00	Non-member mode
01	Member mode
10	Reserved
11	Reserved

3.4.1.8.2 NonmemberRadius Sub-field

The NonmemberRadius 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 field. Receiving devices that are not members of the destination group will discard the frame if the NonmemberRadius field has value 0 and will decrement the field if the NonmemberRadius field has a value in the range 0x01 through 0x06. A value of 0x07 indicates an infinite range and is not decremented.

3.4.1.8.3 MaxNonmemberRadius Sub-field

The maximum value of the NonmemberRadius field for this frame.

3.4.1.9 Source Route Subframe Field

The source route subframe field is only 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.

Octets: 1	1	Variable
Relay count	Relay index	Relay list

Figure 3.8 Source Route Subframe Format

3.4.1.9.1 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.

3.4.1.9.2 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 0 by the originator of the packet, and is incremented by 1 by each receiving relay.

3.4.1.9.3 Relay List Sub-field

The relay list sub-field is a list of the 2-byte short addresses of the nodes to be used as relays for the purpose of source routing the packet. Addresses are formatted least significant byte first, and appear in the order they are used in the source route.

3.4.1.10 Frame Payload Field

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

3.4.2 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 sub-clauses.

3.4.2.1 Data Frame Format

The data frame shall be formatted as illustrated in Figure 3.9.

Octets: 2	Variable	Variable
Frame control	Routing fields	Data payload
NWK header		NWK payload

Figure 3.9 Data Frame Format

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.

3.4.2.1.1 Data Frame Nwk Header Field

The NWK header field of a data 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 data frame, as shown in Table 3.35. 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).

3.4.2.1.2 Data Payload Field

The data payload field of a data frame shall contain the sequence of octets, which the next higher layer has requested the NWK layer to transmit.

3.4.2.2 NWK Command Frame Format

The NWK command frame shall be formatted as illustrated in Figure 3.10.

Octets: 2	Variable	1	Variable
Frame control	Routing fields	NWK command identifier	NWK command payload
NWK header		NWK payload	

Figure 3.10 NWK Command Frame Format

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.

3.4.2.2.1 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 3.35. 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.

3.4.2.2.2 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 3.38.

3.4.2.2.3 NWK Command Payload Field

The NWK command payload field of a NWK command frame shall contain the NWK command itself.

3.5 Command Frames

The command frames defined by the NWK layer are listed in Table 3.38. The following sub-clauses detail how the NLME shall construct the individual commands for transmission.

Table 3.38 NWK Command Frames

Command Frame Identifier	**Command Name**	**Reference**
0x01	Route request	3.5.1
0x02	Route reply	3.5.2
0x03	Route Error	3.5.3
0x04	Leave	3.5.4
0x05	Route Record	3.5.5
0x06	Rejoin request	3.5.6
0x07	Rejoin response	3.5.7
0x00, 0x08 – 0xff	Reserved	—

3.5.1 Route Request Command

The route request command allows a device to request that other devices within radio range 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.

Octets: 1	**1**	**1**	**2**	**1**
Command frame identifier (see Table 3.38)	Command options	Route request identifier	Destination address	Path cost
NWK payload				

Figure 3.11 Route Request Command Frame Format

3.5.1.1 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 must be set to the broadcast address of 0xffff
- The source MAC address and PAN identifier shall be set to the 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

3.5.1.2 NWK Header Fields

In order to send the route request command frame, the source address field in the NWK header shall be set to the address of the originating device.

The destination address in the NWK header shall be set to the router-only broadcast address (see Table 3.50).

3.5.1.3 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, and an up-to-date summation of the path cost.

The command frame identify shall contain the value indicating a route request command frame.

3.5.1.3.1 Command Options Field

The format of the 8-bit command options field is shown in Figure 3.12.

Bit: 0-5	6	7
Reserved	Multicast	Route repair

Figure 3.12 Route Request Command Options Field

3.5.1.3.1.1 The 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.

3.5.1.3.1.2 The Route Repair Sub-field

The route repair sub-field is a single-bit field. It shall have a value of 1 if and only if the route request command frame is being generated as part of a route repair operation for mesh network topology (see sub-clause 3.7.3.6.1).

3.5.1.3.2 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.

3.5.1.3.3 Destination Address

The destination address shall be 2 octets in length and represents the intended destination of the route request command frame.

3.5.1.3.4 Path Cost

The path cost field is 8 bits in length and is used to accumulate routing cost information as a route request command frame moves through the network (see sub-clause 3.7.3.4.2).

3.5.2 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.

Octets: 1	**1**	**1**	**2**	**2**	**1**
Command frame identifier (see Table 3.38)	Command options	Route request identifier	Originator address	Responder address	Path cost
NWK payload					

Figure 3.13 Route Reply Command Format

3.5.2.1 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 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.

3.5.2.2 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 frame type sub-field of the NWK frame control field should be set to indicate that this frame is a NWK layer command 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.
- The source address in the NWK header shall be set to the NWK 16-bit network address of the device that is transmitting the frame.

3.5.2.3 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.

3.5.2.3.1 Command Options Field

The format of the 8-bit command options field is shown in Figure 3.14.

Bit: 0 – 5	6	7
Reserved	Multicast	Route repair

Figure 3.14 Route Reply Command Options Field

3.5.2.3.1.1 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.

3.5.2.3.1.2 Route Repair Sub-field

The route repair sub-field is a single-bit field. It shall have a value of 1 if and only if the route reply command frame is being generated as part of a route repair operation for mesh network topology (see sub-clause 3.7.3.6.1).

3.5.2.3.2 Route Request Identifier

The route request identifier is the 8-bit sequence number of the route request to which this frame is a reply.

3.5.2.3.3 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.

3.5.2.3.4 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.

3.5.2.3.5 Path Cost

The path cost field is used to sum link cost as the route reply command frame transits the network (see sub-clause 3.7.3.4.3).

3.5.3 Route Error Command

A device uses the route error command when it is unable to forward a data frame. The command notifies the source device of the data frame about the failure in forwarding the frame. The payload of a route error command shall be formatted as illustrated in Figure 3.15.

Octets: 1	1	2
Command frame identifier (see Table 3.38)	Error code	Destination address

Figure 3.15 Route Error Command Frame Format

3.5.3.1 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 address and PAN identifier, respectively, of the first hop in the path back to the source of the data frame that encountered a forwarding failure.
- The source MAC address and PAN identifier shall be set to the address and PAN identifier of the device sending the route error 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 implementer shall determine whether an acknowledgment shall be requested.
- The addressing mode and intra-PAN flags shall be set to support the addressing fields described here.

3.5.3.2 NWK Header Fields

In order to send the route error command frame, 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.

The source address in the NWK header shall be set to the address of the device sending the route error command.

3.5.3.3 NWK Payload Fields

The NWK frame payload of the route error command frame contains a command frame identifier field, an error code field and a destination address field as described below. The command frame identifier shall be set so as to specify the route error command frame as defined in Table 3.38.

3.5.3.3.1 Error Code

The error code shall be set to one of the non-reserved values shown in Table 3.39.

Table 3.39 Error Codes for Route Error Command Frame

Value	Error 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

Table 3.39 Error Codes for Route Error Command Frame (Continued)

Value	Error Code
0x08	Target Address Unallocated
0x09	Parent Link Failure
0x0a	Validate route
0x0b	Source route failure
0x0c	Many-to-one route failure
0x0d – 0xff	Reserved

These error codes are used both as values for the error code field of a route error command frame and as values of the Status parameter of the NLME-ROUTE-ERROR.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 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.
- **Validate route:** The multicast route identified in the destination address field should be validated.

- **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.

3.5.3.3.2 Destination Address

The destination address is 2 octets in length and shall contain the destination address from the data frame that encountered the forwarding failure.

3.5.4 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

Octets: 1	1
Command frame identifier (see Table 3.38)	Command options

Figure 3.16 Leave Command Frame Format

3.5.4.1 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 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 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.

3.5.4.2 NWK Header Fields

In order to send a leave command frame, if the request subfield 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, 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 device being requested to leave. If the request subfield is set to 0 then the destination address field in the NWK header shall be set to 0xFFFC so that the indication is received by all ZigBee Routers, 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 leaving the network. The radius field in the NWK header shall be set to 1.

3.5.4.3 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 3.38.

3.5.4.3.1 Command Options Field

The format of the 8-bit command options field is shown in Figure 3.17.

Bit: 0 – 4	5	6	7
Reserved	Rejoin	Request	Remove children

Figure 3.17 Leave Command Options Field

3.5.4.3.1.1 Rejoin Sub-field

The Rejoin sub-field is a single-bit field located at bit 5. 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 join the network again.

3.5.4.3.1.2 Request Sub-field

The request sub-field is a single bit field located at bit 6. 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.

3.5.4.3.1.3 Remove Children Sub-field

The remove children sub-field is a single bit field located at bit 7. If this sub-field has a value of 1 then the children of the device that is leaving the network will also be removed.

3.5.5 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.

Octets: 1	**1**	**Variable**
Command frame identifier (see Table 3.38)	Relay count	Relay list
NWK Payload		

Figure 3.18 Route Record Command Format

3.5.5.1 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 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 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.

3.5.5.2 NWK Header Fields

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 source and destination address fields in the NWK header shall be set to the address of the originating and destination devices, respectively. The source route sub-field of the frame control field shall be set to 0.

3.5.5.3 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.

3.5.5.3.1 Relay Count Field

This 1-byte field contains the number of relays in the relay list field of the route record command. It is initialized to 0 by the originator and is incremented by each receiving relay.

3.5.5.3.2 Relay List Field

The relay list field is a list of the 2-byte short addresses of the nodes that have relayed the packet. Addresses are formatted least significant byte first. Receiving relay nodes append their short address to the list before forwarding the packet.

3.5.6 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.

Octets:1	**1**
Command frame identifier (see Table 3.38)	Capability Information
NWK payload	

Figure 3.19 Rejoin Request Command Frame Format

3.5.6.1 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 short address and PAN identifier, respectively, of the prospective parent.
- The source MAC address shall be set to the previous short 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.

3.5.6.2 NWK Header Fields

When sending the rejoin request command frame, the NLME shall set the source address field of the NWK header to the previous short address of the device transmitting the frame, 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 radius field shall be set to 1.

3.5.6.3 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.

3.5.6.3.1 Capability Information Field

This one-byte field has the format of the capability information field in the association request command in [B1], which is also described in Table 3.17.

3.5.7 Rejoin Response Command

The rejoin response command is sent by a device to inform a child of its short address and rejoin status..

Octets:1	2	1
Command frame identifier (see Table 3.38)	Short address	Rejoin status
NWK payload		

Figure 3.20 Rejoin Response Command Frame Format

3.5.7.1 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 previous network address and PAN identifier, respectively, of the device requesting to rejoin the network.
- 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 will request 'indirect transmission' to be used if the 'Receiver on when idle' bit of the 'Capability Information' byte contained in the rejoin request command is equal to 0x00. Otherwise 'direct transmission' will be used.

3.5.7.2 NWK Header Fields

When sending the rejoin response command frame, the NLME shall set the destination address field of the NWK header to the previous short address of the rejoining device. Both the destination IEEE address sub-field and the source IEEE address sub-fields of the frame control field shall be set to 1. The destination IEEE address field shall be set to the IEEE address of the rejoining device and the source IEEE address field shall be set to the IEEE address of the parent.

3.5.7.3 NWK Payload Fields

The NWK frame payload contains a command identifier field, a short address field, and a rejoin status field. The command frame identifier shall contain the value indicating a rejoin response command frame.

3.5.7.3.1 Short Address Field

If the rejoin was successful this two-byte field contains the new short address assigned to the rejoining device. If the rejoin was not successful this contains the broadcast address (0xfffd).

3.5.7.3.2 Rejoin Status Field

This one-byte field shall contain one of the nonreserved association status values specified in [B1].

3.6 Constants and NIB Attributes

3.6.1 NWK Constants

The constants that define the characteristics of the NWK layer are presented in Table 3.40.

Table 3.40 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	Set at build time
nwkcDefaultSecurityLevel	The default security level to be used (see Chapter 4)	ENC-MIC-64
nwkcDiscoveryRetryLimit	The maximum number of times a route discovery will be retried	0x03
nwkcMaxDepth	The maximum depth (minimum number of logical hops from the ZigBee coordinator) a device can have	0x0f
nwkcMinHeaderOverhead	The minimum number of octets added by the NWK layer to a NSDU	0x08
nwkcProtocolVersion	The version of the ZigBee NWK protocol in the device	0x02
nwkcWaitBeforeValidation	Time duration in milliseconds, on the originator of a multicast route request, between receiving a route reply and sending a message to validate the route	0x500
nwkcRepairThreshold	Maximum number of allowed communication errors after which the route repair mechanism is initiated	0x03
nwkcRouteDiscoveryTime	Time duration in milliseconds until a route discovery expires	0x2710
nwkcMaxBroadcastJitter	The maximum broadcast jitter time measured in milliseconds.	0x40
nwkcInitialRREQRetries	The number of times the first broadcast transmission of a route request command frame is retried	0x03

Table 3.40 NWK Layer Constants (Continued)

Constant	Description	Value
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 milliseconds between retries of a broadcast route request command frame	0xfe
nwkcMinRREQJitter	The minimum jitter, in 2 millisecond slots, for broadcast retransmission of a route request command frame	0x01
nwkcMaxRREQJitter	The maximum jitter, in 2 millisecond slots, for broadcast retransmission of a route request command frame	0x40

3.6.2 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. The attributes of the NIB are presented in Table 3.41.

Table 3.41 NWK IB Attributes

Attribute	Id	Type	Range	Description	Default
nwkSequenceNumber	0x81	Integer	0x00 – 0xff	A sequence number used to identify outgoing frames (see sub-clause 3.7.2)	Random value from within the range
nwkPassiveAckTimeout	0x82	Integer	0x00 – 0x0a	The maximum time duration in seconds allowed for the parent and all child devices to retransmit a broadcast message (passive acknowledgment time-out)	0x03
nwkMaxBroadcastRetries	0x83	Integer	0x00 – 0x5	The maximum number of retries allowed after a broadcast transmission failure	0x03
nwkMaxChildren	0x84	Integer	0x00 – 0xff	The number of children a device is allowed to have on its current network	0x07
nwkMaxDepth	0x85	Integer	0x01 – *nwkcMaxDepth*	The depth a device can have	0x05
nwkMaxRouters	0x86	Integer	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	0x05

Table 3.41 NWK IB Attributes (Continued)

Attribute	Id	Type	Range	Description	Default
nwkNeighborTable	0x87	Set	Variable	The current set of neighbor table entries in the device (see Table 3.43)	Null set
nwkNetworkBroadcastDeliveryTime	0x88	Integer	(*nwkPassiveAckTimeouT* nwkBroadcastRetries*0 – 0xff	Time duration in seconds that a broadcast message needs to encompass the entire network	*nwkPassiveAckTimeout * nwkBroadcastRetries*
nwkReportConstantCost	0x89	Integer	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
nwkRouteDiscoveryRetriesPermitted	0x8a	Integer	0x00-x03	The number of retries allowed after an unsuccessful route request	nwkcDiscoveryRetryLimit
nwkRouteTable	0x8b	Set	Variable	The current set of routing table entries in the device (see Table 3.46)	Null set

Table 3.41 NWK IB Attributes (Continued)

Attribute	Id	Type	Range	Description	Default
nwkSymLink	0x8e	Boolea n	TRUE or FALSE	The current route symmetry setting: TRUE means that routes are considers 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	See Table 3.17	This field shall contain the capability device capability information established at network joining time.	0x00

Table 3.41 NWK IB Attributes (Continued)

Attribute	Id	Type	Range	Description	Default
nwkUseTreeAddrAlloc	0x90	Boolean	TRUE or FALSE	A flag that determines whether the NWK layer should use the default distributed address allocation scheme or allow the next higher layer to define a block of addresses for the NWK layer to allocate to its children: TRUE = use distributed address allocation. FALSE = allow the next higher layer to define address allocation.	TRUE
nwkUseTreeRouting	0x91	Boolean	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

Table 3.41 NWK IB Attributes (Continued)

Attribute	Id	Type	Range	Description	Default
nwkNextAddress	0x92	Integer	0x0000 - 0xfffd	The next network address that will be assigned to a device requesting association. This value shall be incremented by *nwkAddressIncrement* every time an address is assigned.	0x0000
nwkAvailableAddresses	0x93	Integer	0x0000 - 0xfffd	The size of remaining block of addresses to be assigned. This value will be decremented by 1 every time an address is assigned. When this attribute has a value of 0, no more associations may be accepted.	0x0000
nwkAddressIncrement	0x94	Integer	0x0000 - 0xfffd	The amount by which *nwkNextAddress* is incremented each time an address is assigned.	0x0001

Table 3.41 NWK IB Attributes (Continued)

Attribute	Id	Type	Range	Description	Default
nwkTransactionPersistenceTime	0x95	Integer	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
nwkShortAddress	0x96	Integer	0x0000 - 0xffff	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	0x00-0x0f	The identifier of the ZigBee stack profile in use for this device.	0

Table 3.41 NWK IB Attributes (Continued)

Attribute	Id	Type	Range	Description	Default
nwkProtocolVersion	0x98	Integer	0x00-0x0f	The version of the ZigBee protocol currently in use by the NWK layer.	nwkProtocolVersion
nwkGroupIDTable	0x99	Set	Variable	The Group ID Table (see Table 3.43)	
nwkExtendedPANID	0x9A	64-bit extended address	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

Table 3.42 Group Table Entry Format

Field Name	Field Type	Valid Range	Reference
GroupID	Integer	0x0000 – 0xffff	The identifier of a multicast group of which this device is a member

3.7 Functional Description

3.7.1 Network and Device Maintenance

All ZigBee devices shall provide the following functionality:

- Join a network
- Leave 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
- 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.

3.7.1.1 Establishing a New Network

The procedure to establish a new network is initiated through the use of the NLME-NETWORK-FORMATION.request primitive. Only devices that are ZigBee coordinator capable and 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, with the ScanType parameter set to energy detection scan, to the MAC sub-layer. The results are communicated back via the MLME-SCAN.confirm primitive.

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 a ScanType parameter set to active scan and ChannelList set to the list of acceptable channels, 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 0x3fff 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 *macShortAddress* PIB attribute in the MAC sub-layer 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 3.21.

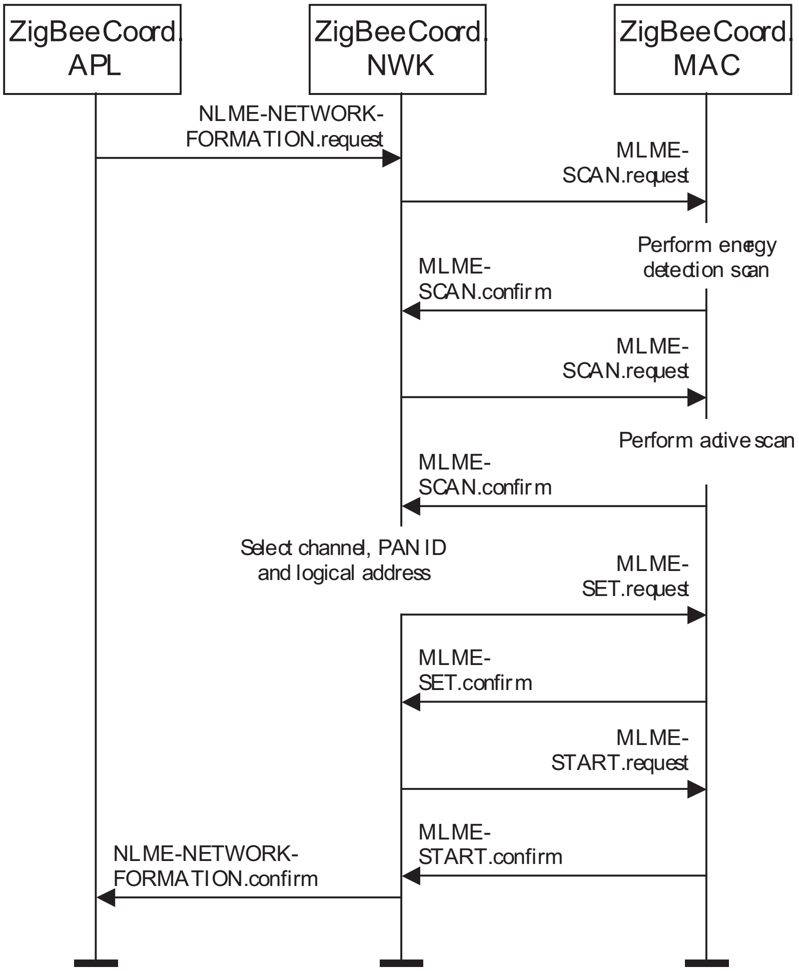


Figure 3.21 Establishing a New Network

3.7.1.2 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 expiry 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 3.22.

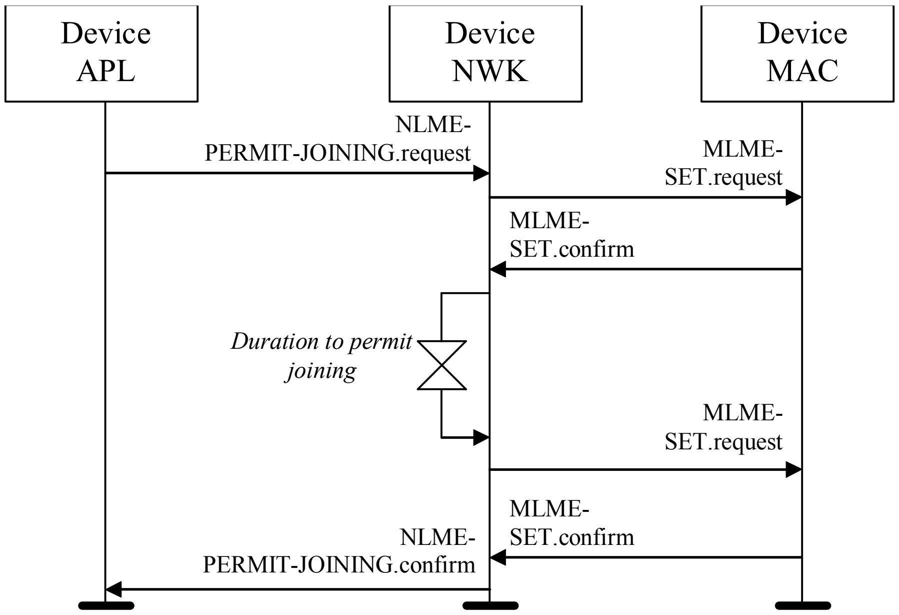


Figure 3.22 Permitting Devices to Join a Network

3.7.1.3 Joining a Network

A parent-child relationship is formed when a device having membership in the network allows a new device to join. The new device becomes the child, while the first device becomes the parent. A child can be added to a network in the following two ways:

- The child can join the network using the MAC layer association procedure
- Or, the child can be added to the network directly by a previously designated parent device.

3.7.1.3.1 Joining a Network Through Association

This sub-clause specifies the procedure a device (child) shall follow to join a network, as well as the procedure a ZigBee coordinator or router (parent) shall follow upon receipt of a join request. Any device may accept a join request from a new device so long as it has the necessary physical capabilities and the available network address space. Only a ZigBee coordinator or a router is physically capable of accepting a join request, while an end device is not.

3.7.1.3.1.1 Child Procedure

The procedure for joining a network using the MAC layer association procedure 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 a passive or 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 3.37 for the structure of the beacon payload) into its neighbor table (see Table 3.43 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 3.9).

Upon receipt of the NLME-NETWORK-DISCOVERY.confirm primitive, the next higher layer is informed of the networks present in the neighborhood. The next higher layer may choose to redo the network discovery to discover more networks or for other reasons. If not, it shall choose a network to join from the discovered networks. 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. A suitable parent device shall permit association and shall have a link cost (see sub-clause 3.7.3.1 for details on link cost) of 3, at most. It shall also have the potential parent field set to 1, if that field is present in the neighbor table entry.

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. If more than one device has a minimum depth, the NWK layer is free to choose from among them.

Once a suitable parent is identified, the NLME shall issue an MLME-ASSOCIATE.request primitive to the MAC sub-layer. 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 max 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 MLME-ASSOCIATE.confirm primitive received by the NWK layer shall contain a 16-bit logical address unique to that network that 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 each added the new device to its neighbor table. Furthermore, the NWK layer will update the value of *nwkShortAddress* 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 to setup its superframe configuration and begin transmitting beacon frames, if applicable. Beacon frames are only transmitted if the BeaconOrder parameter is not equal to 15 (See [B1]). 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 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.

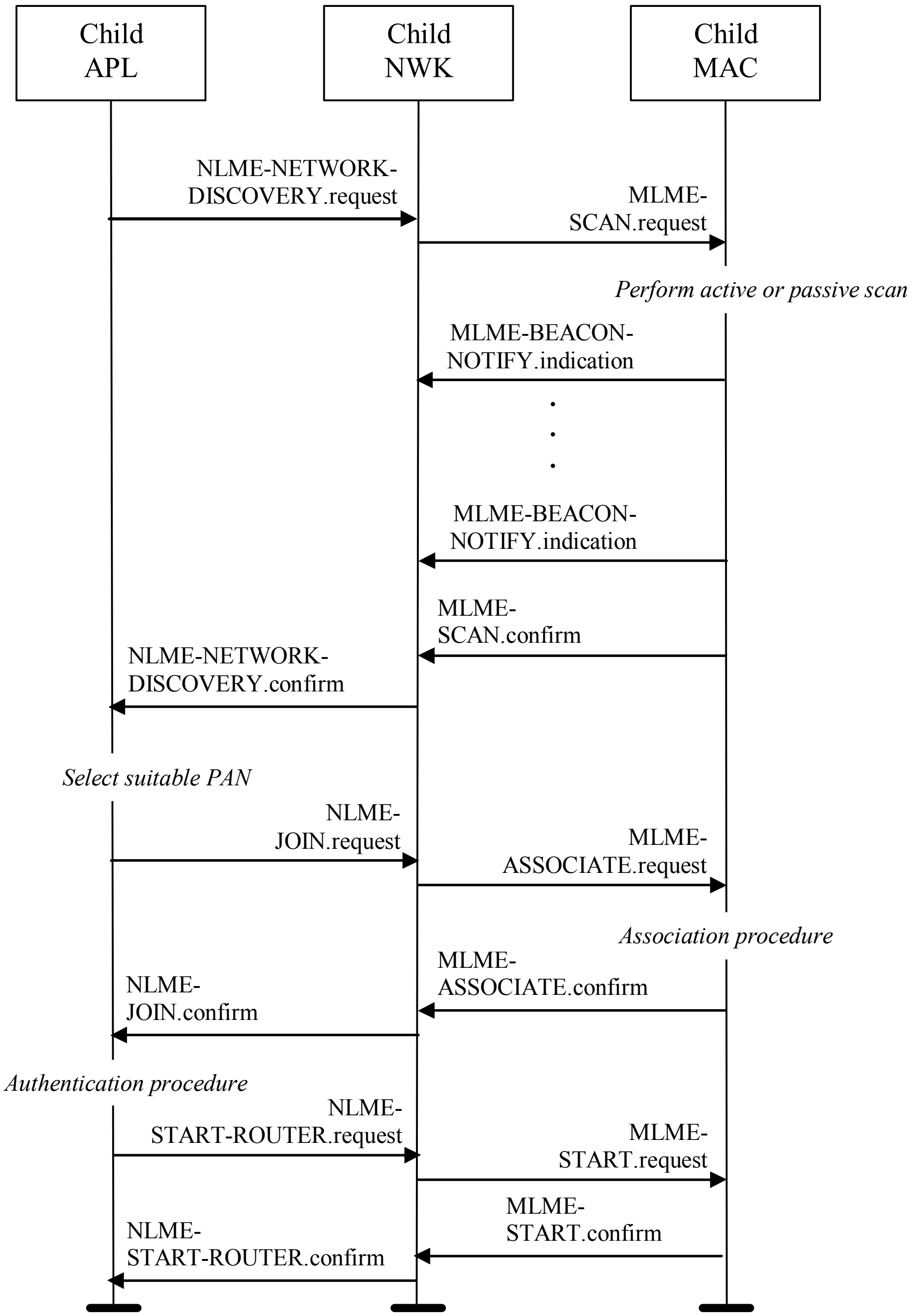


Figure 3.23 Procedure for Joining a Network Through Association

3.7.1.3.1.2 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 a match is found, the NLME shall obtain the corresponding 16-bit network address and issue an association response to the MAC sub-layer. If a match is not found, the NLME shall, if possible, allocate a 16-bit network address for the new device that is unique to that network. A finite address space is allocated to every potential parent device, and a device may disallow a join request once this address space is exhausted. The ZigBee coordinator determines the amount of address space given. See sub-clause 3.7.1.5 and sub-clause 3.7.1.6 for an explanation of the address assignment mechanisms.

If the potential parent has exhausted its allocated address space, 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. This status value uses MAC sub-layer terminology and indicates only that the potential parent does not have the capacity to accept any more children. It is possible in a multi-hop network that other potential parents still having sufficient address space exist within the same network.

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. 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 (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 3.23.

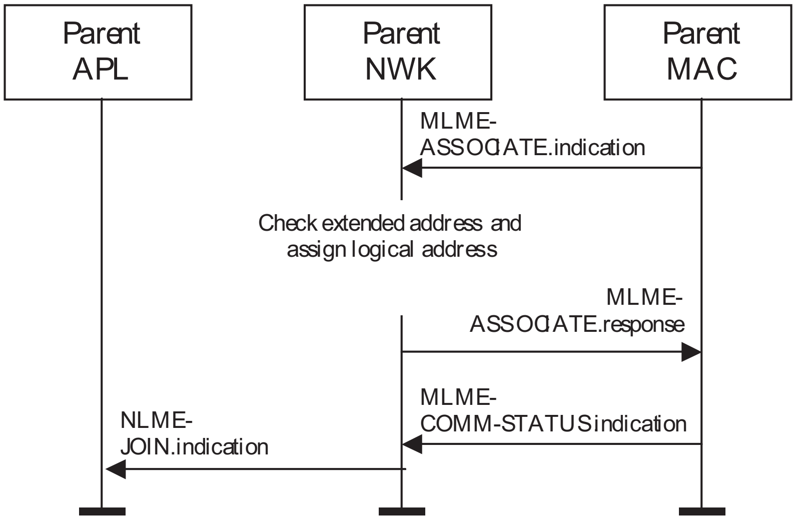

Parent APL
Parent NWK
Parent MAC
MLME-ASSOCIATE.indication
Check extended address and assign logical address
MLME-ASSOCIATE.response
MLME-COMM-STATUS.indication
NLME-JOIN.indication

Figure 3.24 Procedure for Handling a Join Request

3.7.1.3.2 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 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.

3.7.1.3.2.1 Child Procedure

The procedure for rejoining a network using the NWK rejoin procedure shall be initiated by issuing the NLME-JOIN.request primitive, as shown in Figure 3.25, with the RejoinNetwork parameter set to 0x02 and the ExtendedPANId parameter set to the ExtendedPANId of the network to rejoin. The JoinAsRouter parameter will be set to indicate whether the device wants to join as a routing device.

The ScanChannels parameter will 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 a passive or 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 3.37 for the structure of the beacon payload) into its neighbor table (see Table 3.43 and Table 3.44 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 and shall have a link cost (see sub-clause 3.7.3.1) 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 and the *nwkUseTreeAddrAlloc* NIG attribute has a value of TRUE, the device which is at a minimum depth from the ZigBee coordinator shall be chosen. Once a suitable parent is identified, the NLME shall construct a NWK rejoin request command frame. The addressing parameters in the rejoin request command shall be set to contain the addressing information for the device chosen from the neighbor table.

After the successful transmission of the rejoin request command using the MAC data service the network layer will load a count down timer with a value of *aResponseWaitTime* ([B1]), If the RxOnWhenIdle parameter is equal to FALSE, when the timer expires the NWK layer will issue a MLME-POLL.request to the potential parent, to retrieve the rejoin response command.

On receipt of a rejoin response command frame, after the above procedure or at any other time, the device shall check the rejoining device IEEE address field and the parent device IEEE address field of the command frame payload. If the rejoining device IEEE address field is not equal in value to the IEEE address of the receiving device or if the parent device 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 shall 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 could be found, the NLME shall issue the NLME-JOIN.confirm primitive with the Status parameter set to NOT_PERMITTED. 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 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 that 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 each added the new device to its neighbor table. Furthermore, the NWK layer will update the value of *nwkShortAddress* in the NIB.

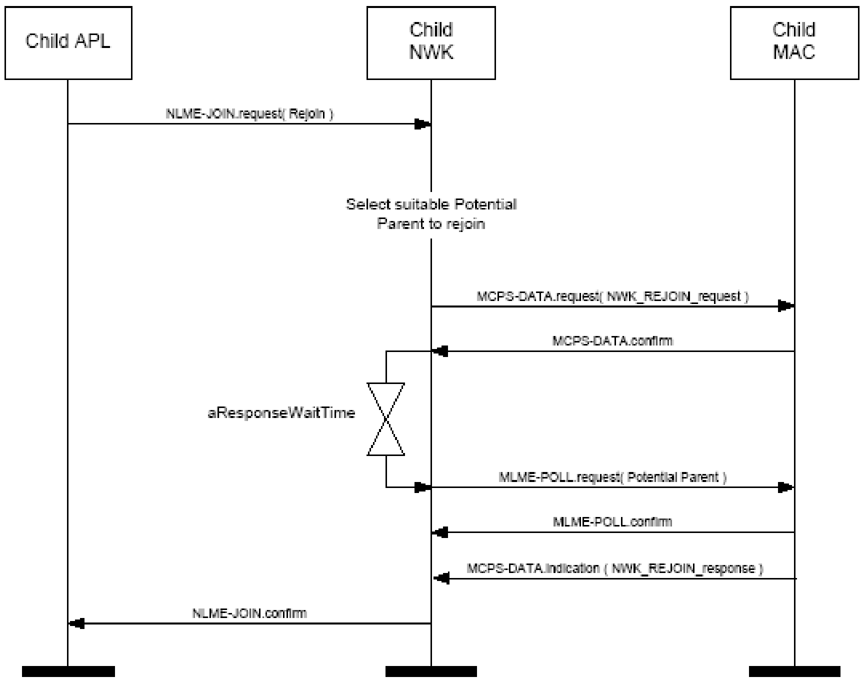

Child APL
Child NWK
Child MAC
NLME-JOIN.request(Rejoin)
Select suitable Potential Parent to rejoin
MCPS-DATA.request(NWK_REJOIN_request)
MCPS-DATA.confirm
aResponseWaitTime
MLME-POLL.request(Potential Parent)
MLME-POLL.confirm
MCPS-DATA.indication (NWK_REJOIN_response)
NLME-JOIN.confirm

Figure 3.25 Child Rejoin Procedure

3.7.1.3.2.2 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 a ZigBee coordinator or a ZigBee router 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 a match is found, 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 match is not found, the NLME shall, if possible, allocate a 16-bit network address for the new device that is unique to that network. A finite address space is allocated to every potential parent device, and a device may disallow a join request once this address space is exhausted. The ZigBee coordinator determines the amount of address space given. See sub-clause 3.7.1.5 and sub-clause 3.7.1.6 for an explanation of the address assignment mechanisms. If the potential parent has exhausted its allocated address space, 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. 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 3.26.

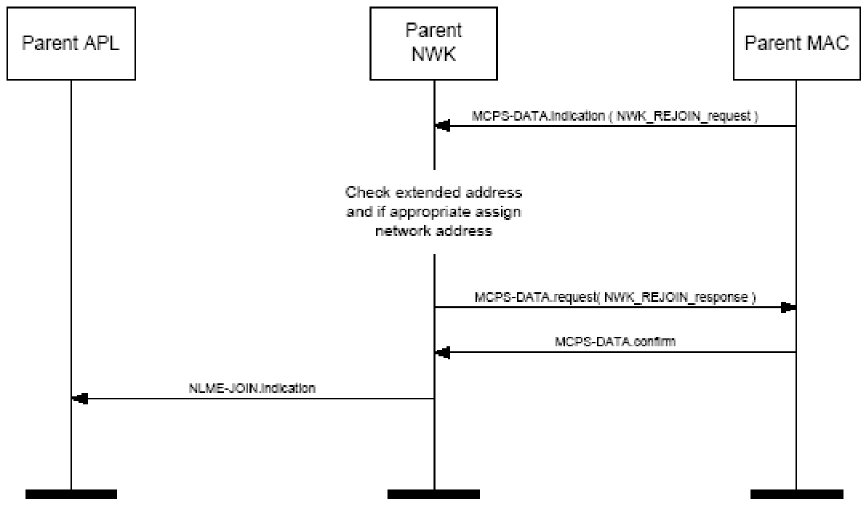

Parent APL
Parent NWK
Parent MAC
MCPS-DATA.indication (NWK_REJOIN_request)
Check extended address and if appropriate assign network address
MCPS-DATA.request(NWK_REJOIN_response)
MCPS-DATA.confirm
NLME-JOIN.indication

Figure 3.26 Parent Rejoin Procedure

3.7.1.3.3 Joining a Network Directly

This sub-clause 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 list by 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, which is unique to that network. A finite address space is allocated to every potential parent device, and the potential parent shall create a new entry for the device in its neighbor table only if it has sufficient capacity. If capacity is not available, 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.

The ZigBee coordinator determines the amount of address space given to every potential parent device. See sub-clause 3.7.1.5 and sub-clause 3.7.1.6 for an explanation of the address assignment mechanism.

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 sub-clause 3.7.1.3.3.1.

A parent that supports direct joining shall follow the procedure illustrated in Figure 3.27 to successfully join a device to the network directly. This procedure does not require any over-the-air transmissions.

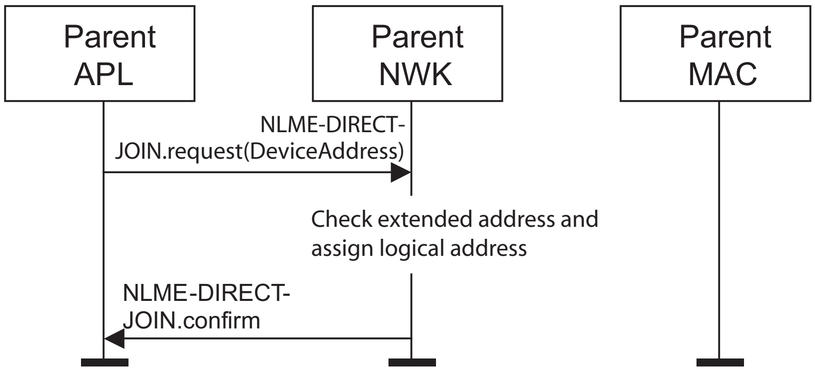

Parent APL
Parent NWK
Parent MAC
NLME-DIRECT-JOIN.request(DeviceAddress)
Check extended address and assign logical address
NLME-DIRECT-JOIN.confirm

Figure 3.27 Joining a Device to a Network Directly

3.7.1.3.3.1 Joining or Re-joining a Network Through Orphaning

This sub-clause 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 communications failure notifications from its MAC sub-layer.

3.7.1.3.3.2 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 orphan scan was successful — that is, the child has found its parent — 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 sub-clause 3.7.7.1, 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 3.28

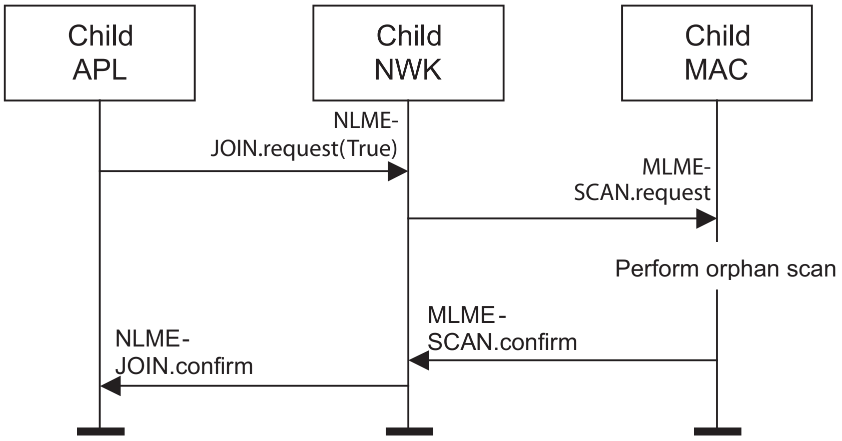


Figure 3.28 Child Procedure for Joining or Re-joining a Network Through Orphaning

3.7.1.3.3.3 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 a ZigBee routers (that is, devices with parental capabilities) 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 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 3.29.

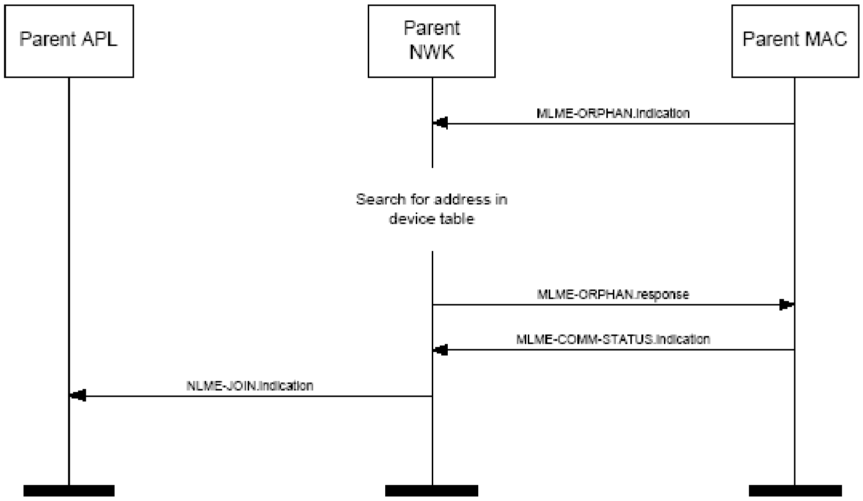


Figure 3.29 Parent Procedure for Joining or Re-joining a Device to its Network Through Orphaning

3.7.1.4 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.

Mandatory and optional data that are used in normal network operation are listed in Table 3.43.

Table 3.43 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; This field shall be present if the neighbor is a parent or child of the device
Network address	Network address	0x0000 – 0xffff	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 the 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 portions of the CAP:
			
			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

Relationship	Integer	0x00 – 0x03	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 This field shall be present in every neighbor table entry

Table 3.43 Neighbor Table Entry Format (Continued)

Field Name	Field Type	Valid Range	Description
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 sub-clause 3.7.3.1 for discussion of how this is calculated. This field shall be present in every neighbor table entry.
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.

Information that may be used during network discovery and rejoining, as described above, is shown in Table 3.44. 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 3.44 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 – *nwkcMaxDepth*	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.

3.7.1.5 Distributed Address Assignment Mechanism

The default value for the NIB attribute *nwkUseTreeAddrAlloc* is TRUE, 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 which 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:

$$Cskip(d) = \begin{cases} 1 + Cm \cdot (Lm - d - 1), \text{if Rm} = 1 \\ \dfrac{1 + Cm - Rm - Cm \cdot Rm^{Lm-d-1}}{1 - Rm}, \text{otherwise} \end{cases}$$

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 *macAssociationPermit* PIB attribute in the MAC sub-layer to FALSE by issuing the MLME-SET.request primitive. It shall respond to any future NLME-PERMIT-JOINING.request primitive with a PermitDuration value greater than or equal to 0x01 by sending a NLME-PERMIT-JOINING.confirm primitive with a Status of INVALID_REQUEST. It shall then terminate the permit joining procedure.

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 nth address, A_n, given by the following equation:

$$A_n = A_{parent} + Cskip(d) \cdot Rm + n$$

Where $1 \le n \le (Cm - Rm)$ and A_{parent} represents the address of the parent.

The *Cskip(d)* values for an example network having *nwkMaxChildren*=4, *nwkMaxRouters*=4 and *nwkMaxDepth*=3 are calculated and listed in Table 3.45. Figure 3.30 illustrates the example network.

Table 3.45 Example Addressing Offset Values for Each Given Depth Within the Network

Depth in the Network, *d*	Offset Value, *Cskip(d)*
0	21
1	5
2	1
3	0

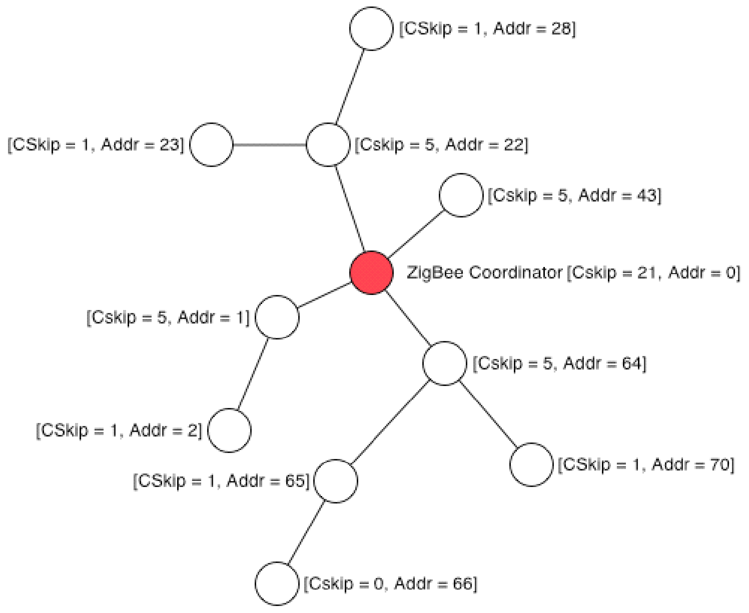


Figure 3.30 Address Assignment in an Example Network

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. 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.

3.7.1.6 Higher-layer Address Assignment Mechanism

When the NIB attribute *nwkUseTreeAddrAlloc* has a value of FALSE, an alternate addressing scheme is used where the block of addresses to be assigned by a device is set by the next higher layer using the NIB attributes *nwkNextAddress*, *nwkAvailableAddresses* and *nwkAddressIncrement*. In this scheme, when a device has *nwkAvailableAddresses* equal to 0, it shall be incapable of accepting association requests. The NLME of such a device shall set the *macAssociationPermit* PIB attribute in the MAC sub-layer to FALSE by issuing the MLME-SET.request primitive and shall respond to future NLME-PERMIT-JOINING.request primitives with a PermitDuration of equal to or greater than 0x01 by sending a NLME-PERMIT-JOINING.confirm primitive with a Status parameter of INVALID_REQUEST. It shall then terminate the permit joining procedure. If the device has *nwkAvailableAddresses* greater than 0, it shall accept association requests by setting the *macAssociationPermit* PIB attribute in the MAC sub-layer to TRUE by issuing the MLME-SET.request primitive. In addition, it shall respond to any future NLME-PERMIT-JOINING.request primitives with a PermitDuration greater than or equal to 0x01 by emitting a NLME-PERMIT-JOINING.confirm primitive with a Status parameter value of SUCCESS. While a device is accepting associations, it shall use the value of *nwkNextAddress* as the address to be assigned to the next device that successfully associates. After a successful association, the value of *nwkNextAddress* shall be incremented by the value of *nwkAddressIncrement* and the value of *nwkAvailableAddresses* shall be decremented by 1.

3.7.1.7 Installation and Addressing

It should be clear that *nwkMaxDepth* roughly determines the “distance” 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 3.30, 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, when *nwkUseTreeAddrAlloc* has a value of TRUE, is not dynamically balanced, 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.

3.7.1.8 Leaving a Network

This sub-clause 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 be slated for removal as well.

3.7.1.8.1 Method for a Device to Initiate Its Own Removal from a Network

This sub-clause 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 3.31.

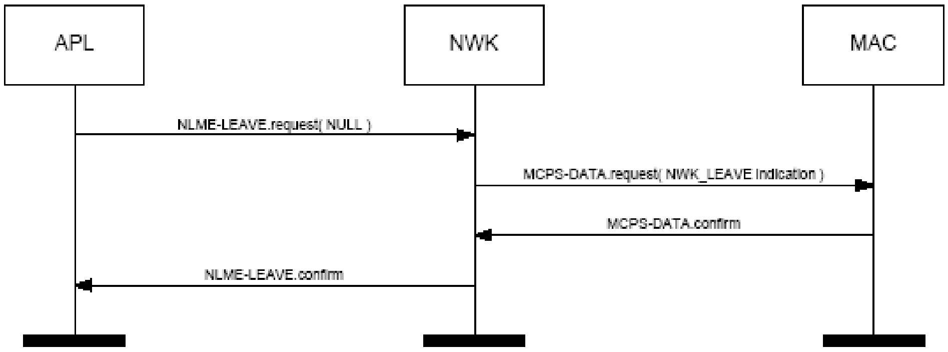

APL
NWK
MAC
NLME-LEAVE.request(NULL)
MCPS-DATA.request(NWK_LEAVE Indication)
MCPS-DATA.confirm
NLME-LEAVE.confirm

Figure 3.31 Initiation of the Leave Procedure

On receipt, by the NWK layer of a ZigBee router or ZigBee coordinator, of the NLME-LEAVE.request primitive with the DeviceAddress parameter equal to NULL (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.

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. After transmission of the leave command frame, it shall issue a NWK-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.

3.7.1.8.2 Method for a Device to Remove Its Child from the Network

This sub-clause 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 3.32.

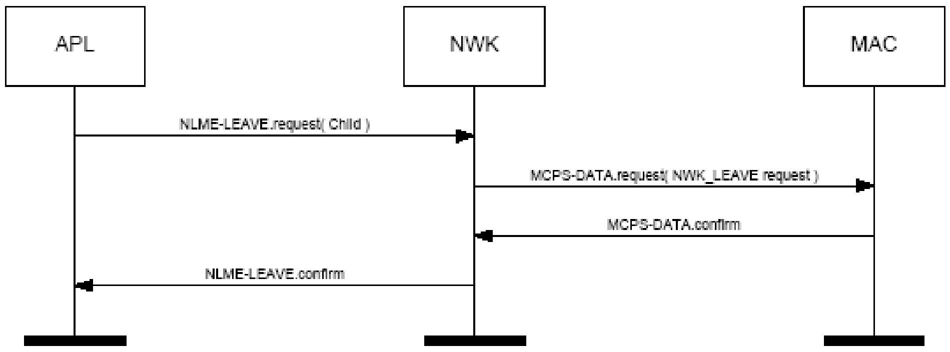

APL
NWK
MAC
NLME-LEAVE.request(Child)
MCPS-DATA.request(NWK_LEAVE request)
MCPS-DATA.confirm
NLME-LEAVE.confirm

Figure 3.32 Procedure for a Device to Remove Its Child

On receipt, by the NWK layer of a ZigBee coordinator or ZigBee router, of the NLME-LEAVE.request primitive with the DeviceAddress parameter equal to the 64-bit IEEE address of a child device, if the Silent parameter of the NLME-LEAVE.request primitive has a value of FALSE, 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 Silent parameter has a value of TRUE the device shall not send a network leave command frame.

The NWK layer shall then issue the NLME-LEAVE.confirm primitive with the DeviceAddress parameter set to the 64-bit IEEE address of the child device being removed and with a status of SUCCESS or, if a network leave command was transmitted and the transition was unsuccessful, with the same value as the status parameter returned by the MCPS-DATA.indication 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 the ReuseAddress parameter of the NLME-LEAVE.request primitive has a value of TRUE 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.

3.7.1.8.3 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 3.33, 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 contained in the Leaving Device IEEE Address subfield 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 it 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.

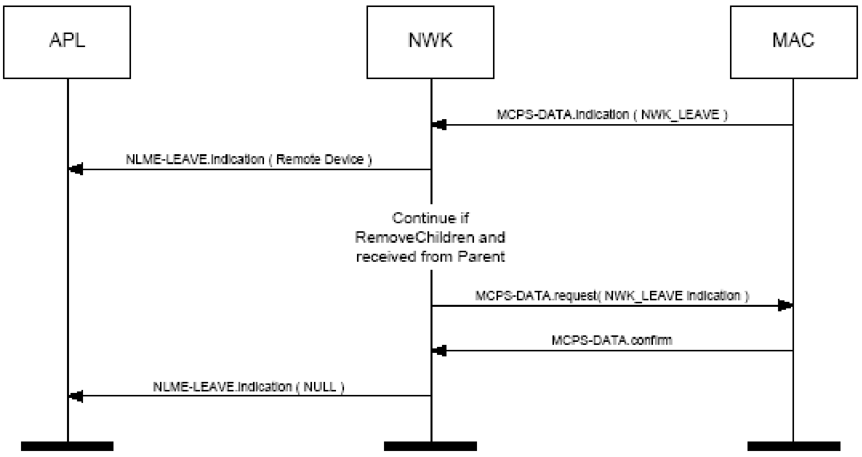

APL
NWK
MAC
MCPS-DATA.indication (NWK_LEAVE)
NLME-LEAVE.indication (Remote Device)
Continue if RemoveChildren and received from Parent
MCPS-DATA.request(NWK_LEAVE indication)
MCPS-DATA.confirm
NLME-LEAVE.indication (NULL)

Figure 3.33 On Receipt of a Leave Command

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 either the value of the request sub-field of the command options field is found to have a value of 1 or 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 subfield 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 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 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 and the source of the leave command frame is a device other than the parent of the recipient, the frame shall be immediately and silently discarded.

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, the receiver 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 sub-clause 3.7.5 to enhance the reliability of the leave procedure but, beyond this note, these mechanisms are outside the scope of this specification.

3.7.1.9 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 the 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].

3.7.1.10 Resetting a Device

The NWK layer of a device shall be reset immediately following initial power-up, before a join attempt and after a leave attempt. 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. However, a device shall discard its network address after the reset. Such a device shall look for an association and get a network address from its parent. 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 and procedures out of the scope of this specification.

3.7.2 Transmission and Reception

3.7.2.1 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.

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 NWK IB (see clause 3.6). 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 constructs a new NWK frame, either as a result of a request from the next higher layer to transmit a new NWK data frame or when it needs to construct a new NWK layer command frame. After being incremented, 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 sub-clause 4.2.3). 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, or if the frame originated at the higher layer and the nwkSecureAllFrames NIB attribute is set to 0x00, 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.

3.7.2.2 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 will 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 will cause the NWK layer 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 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 ID 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:

- Broadcast and multicast data frames shall be relayed according to the procedures outlined in sub-clauses 3.7.5 and 3.7.6.
- 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 sub-clause 3.7.3.3. (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 sub-clause 3.7.3.3.2.
- The procedure for handling route request command frames is outlined in sub-clause 3.7.3.4.2.

The procedure for handling route reply command frames for which the destination address matches the device's network address is outlined in sub-clause 3.7.3.4.3.

- Route reply command frames for which the destination address does not match the device's network address shall be discarded immediately. Route error 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 4.2.3) 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 and the incoming frame is a NWK command 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 data frame, the NLDE shall check the *nwkSecureAllFrames* NIB attribute. If this attribute is set to 0x01, 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.

3.7.3 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
- Participate in route discovery on behalf of end devices
- Participate in end-to-end route repair
- Participate in local route repair
- Employ the ZigBee path cost metric as specified in route discovery and route repair

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 end-to-end route repair
- Initiate local route repair on behalf of other ZigBee routers

3.7.3.1 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 $[D_1, D_2 \ldots D_L]$ and a link, $[D_i, D_{i+1}]$ 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 $C\{[D_i, D_{i+1}]\}$ is referred to as a link cost. The link cost $C\{l\}$ for a link l is a function with values in the interval $[0 \ldots 7]$ defined as:

$$C\{l\} = \begin{cases} 7, \\ \min\left(7, \operatorname{round}\left(\dfrac{1}{p_l^4}\right)\right) \end{cases}$$

where p_l 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 p_l 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.

The question that remains, however, is how p_l is to be estimated or measured. This is primarily an implementation issue and implementers are free to apply their ingenuity. p_l 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 on production hardware, as inaccurate costs will hamper the operating ability of the ZigBee routing algorithm.

3.7.3.2 Routing Tables

A ZigBee router or ZigBee coordinator may maintain a routing table. The information that shall be stored in a ZigBee routing table is shown in Table 3.46. 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 3.46 Routing Table

Field Name	Size	Description
Destination address	2 bytes	The 16-bit network address or Group ID of this route; If the destination device is a ZigBee router or ZigBee coordinator, this field shall contain the actual 16-bit address of that device; If the destination device is an end device, this field shall contain the 16-bit network address of that device's parent
Status	3 bits	The status of the route. See Table 3.47 for values
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 bytes	The 16-bit network address of the next hop on the way to the destination

Table 3.47 enumerates the values for the route status field.

Table 3.47 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 3.48. Routing table entries are long-lived and persistent, while route discovery table entries last only as long as the duration of a single route discovery operation and may be reused.

Table 3.48 Route Discovery Table

Field Name	Size	Description
Route request ID	1 byte	A sequence number for a route request command frame that is incremented each time a device initiates a route request
Source address	2 bytes	The 16-bit network address of the route request's initiator
Sender address	2 bytes	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 byte	The accumulated path cost from the source of the route request to the current device
Residual cost	1 byte	The accumulated path cost from the current device to the destination device
Expiration time	2 bytes	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."

If a device has the capability to initiate a many-to-one route request, it shall also maintain a source route table.

3.7.3.3 Upon Receipt of a Unicast Data Frame

On receipt of a unicast data frame, either from the MAC sub-layer or 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 sub-clause 3.7.2.1. 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 short address if it is a router or the coordinator, or the short address of its parent if it is an end device. Define the *routing destination* of a frame to be the routing address of the frame's NWK destination. Note that ZigBee address assignment makes it possible to derive the routing address of any device from its address. See sub-clause 3.7.1.5 for details.

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 route record required field of the routing table entry is set to TRUE, and the frame being relayed was initiated locally or by a child end device, the device shall first initiate a route record command frame to the destination as specified in sub-clause 3.7.3.4.4.

When relaying a data 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 will 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. The TxOptions parameter should always be non-zero when bitwise ANDed, with the value 0x01, indicating acknowledged transmission. 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 frame shall be treated as though route discovery has been initiated for this frame, as described in sub-clause 3.7.3.4.1. 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. Again, 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, 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, the device shall transmit the frame using source routing as described in sub-clause 3.7.3.3.1. 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 sub-clause 3.7.3.4.1. 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.

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 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 sub-clause 3.7.1.5.

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 data frame for any reason, the router or coordinator shall make its best effort to report the failure. This reporting may take one of two forms. If the failed data frame was being relayed as a result of a request from the next higher layer, then the NWK layer shall issue an NLME-ROUTE-ERROR.indication to the next higher layer. The value of the ShortAddr 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 both issue the NLME-ROUTE-ERROR.indication primitive and send a route error command frame back to the source of the data frame. The destination address field of the route error 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 3.39.

3.7.3.3.1 Originating a Source Routed Data Frame

On receipt of a data frame from the next higher layer, the device shall transmit the frame using source routing under the conditions described in sub-clause 3.7.3.3.

The source route shall be retrieved from the source route 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 byte 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.

3.7.3.3.2 Relaying a Source Routed Data Frame

To relay a source routed data frame received from the MAC sub-layer as described in sub-clause 3.7.3.3, the device shall search the relay list sub-field of the NWK header source route subframe for its short address. If the short address is not found, or if its index in the relay list does not match the value of the relay index sub-field, the frame shall be discarded and no further action shall be taken.

If the relay index sub-field has a value of 0, the device shall relay the frame directly to the NWK header destination using the MCPS-DATA.request primitive.

If the relay index sub-field has a value other than 0, 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.

3.7.3.4 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 multicast group. *Many-to-one route discovery* is performed by a source device to establish routes to itself from all ZigBee routers and ZigBee coordinators within a given radius. Throughout sub-clause 3.7.3.4 *a destination address* may be a 16-bit broadcast address, the 16-bit NWK 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) 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.

3.7.3.4.1 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 on receipt of an NLDE-ROUTE-DISCOVERY.request primitive from the next higher layer where the DstAddrMode parameter has a value of 0x00; Or, on receipt of an NLDE-DATA.request primitive from a higher layer with the DstAddrMode set to 0x00 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 in which 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. In any of these cases, if the device does not have routing capacity and the NIB attribute *nwkUseTreeRouting* has a value of TRUE, the data frame in question shall be routed along the tree using hierarchical routing. If the device does not have routing capacity and the NIB attribute *nwkUseTreeRouting* has a value of FALSE, then the frame shall be discarded and the NWK shall generate a NLME-ROUTE-ERROR.indication or route error command frame as described in 3.7.3.6.

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 sub-clause 3.7.6.1.2.

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, 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 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 sub-clause 3.7.3.2.

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 3.38.
- 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 to 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* milliseconds.

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. It is not necessary for the device to create a route table entry in this case. A route request command frame with destination address set to the broadcast address 0xffff shall be created and broadcast as described in this sub-clause.

3.7.3.4.2 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 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 child devices 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 a destination. The link cost from the next hop device to the current device shall be computed as described in sub-clause 3.7.3.1 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 sub-clause 3.7.3.1. 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 either the device or 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 3.48) 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 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 GroupID 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 3.48) entry exists with the same route request identifier and source address field. If no such entry exists, one shall be created.

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 sub-clause 3.7.3.1, 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 sub-clause 3.7.3.1, and 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 3.48) 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* milliseconds. If a routing table entry corresponding to the routing address of the destination exists and its status is not ACTIVE, 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 *nwkSymLink* attribute is set to TRUE, or 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, the many-to-one field and route table entry shall be set to TRUE, and if the next hop field changed, the route record required field shall be set to TRUE. The status field shall be set to ACTIVE. When the route vrequest 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 sub-clause 3.7.3.1, 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 rebroadcast the route request command frame using the MCPS-DATA.request primitive.

When rebroadcasting a route request command frame, the NWK layer shall delay retransmission by a random jitter amount calculated using the formula:

$$2 \times R[nwkcMinRREQJitter, nwkcMaxRREQJitter]$$

where $R[a, b]$ is a random function on the interval $[a, b]$. 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.

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 network 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 network header of route request command frame. Using the sender address field from the route discovery table entry corresponding to the source address in the network header of route request command frame, the device shall compute the link cost as described in sub-clause 3.7.3.1. This link cost shall be entered in the path cost field. The route reply command frame is then unicast to the destination by using MCPS-DATA.request primitive and the sender address obtained from the route discovery table as the next hop.

3.7.3.4.3 Upon Receipt of a Route Reply Command Frame

On receipt of a route reply command frame, a device performs the following procedure.

If the receiving device has no routing capacity and its NIB attribute *nwkUseTreeRouting* has a value of TRUE, it shall forward the route reply 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 sub-clause 3.7.3.1 and adding this to the value in the route reply path cost field.

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 so, 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 and 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.

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. It is an error here 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 route table entry to be modified, and the route 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* milliseconds if the device is the destination of the route reply and *nwkcRouteDiscoveryTime* milliseconds 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 sub-clause 3.7.3.1. This cost shall be added to the path cost field in the route reply. The destination address in the command frame network 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.

3.7.3.4.4 Initiation and Processing of a Route Record Command Frame

As described in sub-clause 3.7.3.3, if a device is forwarding a unicast data message and the route record required field of the destination's route table entry has a value of TRUE, then the device shall unicast a route record command to the destination, and if it is transmitted successfully to the next hop, the route record required field of the routing table entry shall be set to FALSE.

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 the route record command is transmitted successfully to the next hop, the route record required field of the destination's route table entry shall be set to FALSE. 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 id, the command frame shall be discarded and the NWK will inform the next higher layer using the NLME-ROUTE-ERROR.indication primitive.

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.

3.7.3.5 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* milliseconds. For entries whose GroupId flag 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* milliseconds if the device is the destination of the route reply and *nwkcRouteDiscoveryTime* milliseconds if it is not. When the timer expires, the device shall delete the route request 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 one buffered pending route discovery or a route error command with a error code of 0x0a (validate route). If the routing table entry corresponding to the destination address has any Status field value other than ACTIVE and there are no other entries in the route discovery table with the same destination field value, that routing table entry shall also be deleted.

3.7.3.6 Route Maintenance

A device NWK layer shall maintain a failure counter for each neighbor to which it has an outgoing link, that is, to which it has been required to send data frames. If the value of the outgoing link failure counter ever exceeds *nwkcRepairThreshold* then the link is classified as a failed link and the device shall initiate route repair 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.

3.7.3.6.1 Route Repair for Mesh Network Topology

If a failed link is encountered while a device is forwarding a source routed frame, a route error with error code “Source route failure” shall be unicast back to the source device, where it shall be passed up to the next higher layer using the NLME-ROUTE-ERROR.indication primitive.

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 route error with error code “Many-to-one route failure” shall be generated. The destination address field in the NWK header of the route error 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 route error command payload shall be equal to the source address field in the NWK header of the frame causing the error. The route error 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 route table entry to the destination. Upon receipt of the route error frame, if no route table entry for the destination is present, or if delivery of the route error to the next hop in the route table entry fails, the route error 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 route error is relayed. Upon receipt of the route error frame by the destination, it shall be passed up to the next higher layer using the NLME-ROUTE-ERROR.indication primitive. The route is not automatically rediscovered by the NWK layer.

If a failed link is encountered that is not part of a many-to-one route and the frame being forwarded is not source routed, the upstream device shall initiate route repair. If the upstream device is unable to initiate route repair due to a lack of routing capacity or some other limitation, the device shall issue a route error command frame back to the source device with the error code indicating the reason for the failure (see Table 3.39), and issue an NLME-ROUTE-ERROR.indication to the next higher layer.

If the upstream device is able to initiate route repair, it shall do so by broadcasting a route request command frame with the source address set to its own address and the destination address set to the destination address of the frame that failed in transmission. The route request command frame shall have the route repair sub-field in the command options field of the command frame payload set to 1 indicating route repair.

While a device is performing route repair for a particular destination, a device shall not forward frames to that destination. Any frames that it has pending for that destination at the time route repair is initiated and any frames for that destination that arrive before the completion of route repair shall either be buffered until the completion of route repair or discarded depending on the capabilities of the device.

On receipt of a route request command frame a routing node shall perform the procedure outlined in sub-clause 3.7.3.4.2. If the routing node is the destination of the route request command frame or the destination is one of its end-device children, it shall reply with a route reply command frame. The route reply command frame shall have the route repair sub-field in the command options field of the command frame payload set to 1 indicating route repair.

If a route reply command frame does not arrive at the upstream device within *nwkcRouteDiscoveryTime* milliseconds, the upstream device shall send a route error command frame to the source device. If the upstream device does receive a route reply within the designated time, it will forward any data that it may have buffered pending the repair to the destination.

If the source device receiving a route error command frame does not have routing capacity and its NIB attribute *nwkUseTreeRouting* has a value of TRUE, it shall construct a route request command frame as described in sub-clause 3.7.3.4.1 and unicast the command frame towards its destination along the tree using hierarchical routing. If the source device does have routing capacity, it shall initiate normal route discovery as described in sub-clause 3.7.3.4.1. If route discovery fails, then the NWK shall issue the NLME_ROUTE-ERROR.indication primitive to the next higher layer.

If an end device that is also an RFD encounters a failed link to its parent, the end device shall inform the next higher layer using the NLME-ROUTE-ERROR.indication primitive with a Status parameter value of 0x09 indicating parent link failure (see Table 3.39). 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-ROUTE-ERROR.indication primitive with a Status parameter value of 0x09 indicating parent link failure.

3.7.4 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 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.

3.7.4.1 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 3.34.

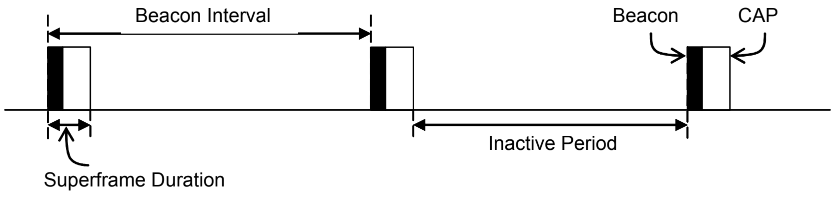


Figure 3.34 Typical Frame Structure for a Beaconing Device

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 sub-clause 3.7.7 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 sub-clause 3.3.6. 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 3.35 illustrates the relationship between the active superframe portions of a parent and its child.

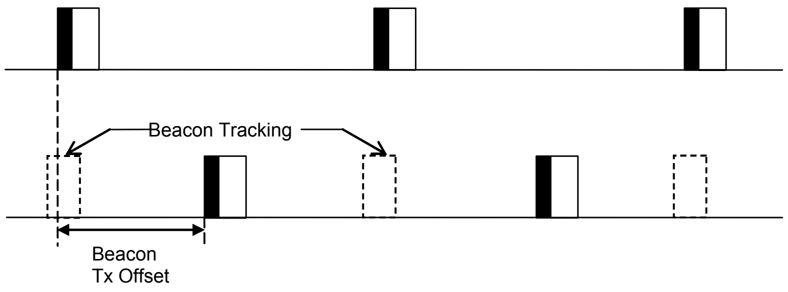


Figure 3.35 Parent-Child Superframe Positioning Relationship

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.

3.7.4.2 MAC Enhancement

In order to employ the beacon scheduling algorithm just described, it is necessary to implement the following enhancement to the IEEE Std 802.15.4-2003 MAC sub-layer.

A new parameter, StartTime, shall be added to the MLME-START.request primitive to specify the time to begin transmitting beacons. The new format of the primitive is as follows:

```
MLME-START.request        {
                          PANID,
                          LogicalChannel,
                          BeaconOrder,
                          SuperframeOrder,
                          PANCoordinator,
                          BatteryLifeExtention,
                          CoordRealignment,
                          SecurityEnable,
                          StartTime
                          }
```

The StartTime parameter is fully described in Table 3.49, and the description of all other parameters can be found in IEEE 802.15.4-2003 [B1].

Table 3.49 Start Time for Beacon Transmissions

Name	Type	Valid Range	Description
StartTime	Integer	0x000000 – 0xffffff	The time at which to begin transmitting beacons. If the device issuing the primitive is the PAN coordinator, this parameter is ignored and beacon transmissions will begin immediately. Otherwise, this parameter specifies the time relative to its parent's beacon. The parameter is specified in symbols and is rounded to a backoff slot boundary. The precision of this value is a minimum of 20 bits, with the lowest 4 bits being the least significant.

3.7.5 Broadcast Communication

This subclause specifies how a broadcast transmission is accomplished within a ZigBee network. This mechanism is used to broadcast all network layer data frames. 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 3.50.

Table 3.50 Broadcast Addresses

Broadcast Address	Destination Group
0xffff	All devices in PAN
0xfffe	Reserved
0xfffd	*macRxOnWhenIdle* = TRUE
0xfffc	All routers and coordinator
0xfff8-0xfffb	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 in the case of non-beacon-enabled ZigBee networks. Passive acknowledgement means that every ZigBee router and ZigBee coordinator keeps track if 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 and each ZigBee router 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 broadcast transaction table (BTT).

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 , the frame shall be discarded. If the destination address corresponds to the device type of the receiver, the device will 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. If the radius field value is greater than 0 it shall retransmit the frame. Otherwise it shall drop the frame. 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 ignored. 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* seconds 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* seconds have elapsed since its creation. The entry should change status 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 the receiving 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 parameter 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 3.36 shows a broadcast transaction between a device and two neighboring devices.

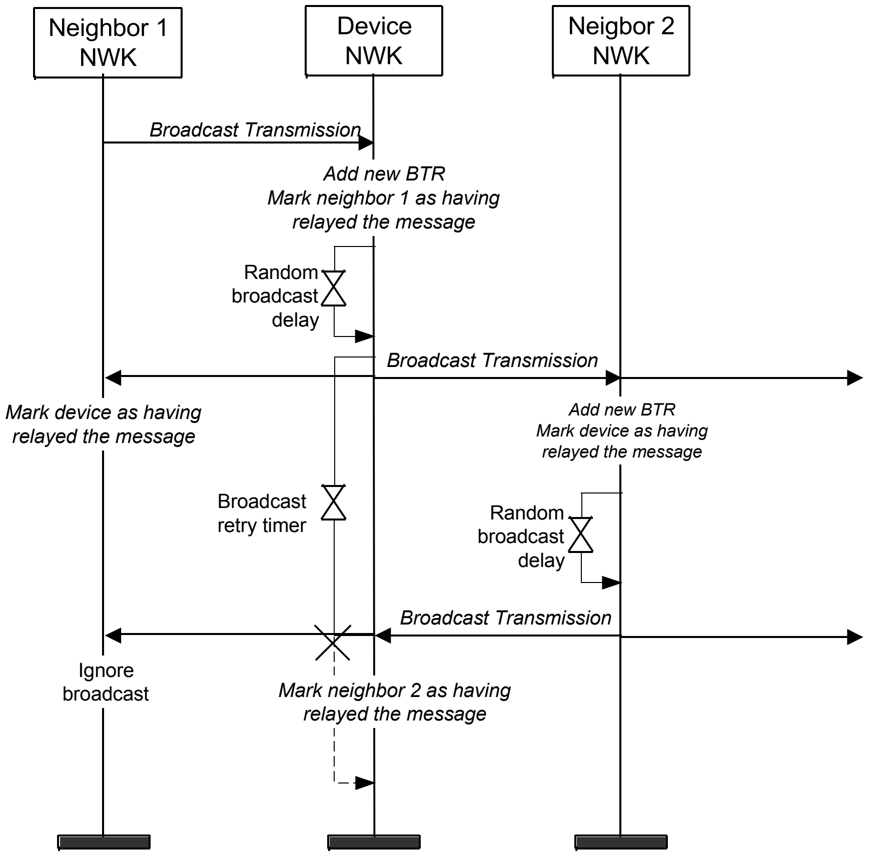


Figure 3.36 Broadcast Transaction Message Sequence Chart

3.7.6 Multicast Communication

This sub-clause specifies how multicast transmission is accomplished within a ZigBee network. Multicasts provide many-to-many routing. Multicast addressing is done using 16-bit multicast group Ids. Each device has a multicast table indicating which multicast groups that device is a member of. A multicast message is sent to a particular destination group and is received by all devices that list that group's ID in their multicast table. Only data frames are multicast; there are no multicast command frames. Multicast frames have a mode flag that indicates whether they are in non-member mode or member mode. Member mode is used to propagate multicasts between the devices that are members of the destination group. Member mode multicasts are recorded in the BTT as if they were broadcasts. 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. Multicasts transmitted by members of the destination multicast group are always in member mode. Multicasts transmitted by non-members are in member mode if they have previously been transmitted by a group member and in non-member mode if not.

3.7.6.1 Upon Receipt of a Multicast Frame from the Next Higher Layer

If a data frame 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* which group ID field matches the destination group ID of the frame. If a matching entry is found, the NWK layer shall multicast the frame according to the procedure outlined in sub-clause 3.7.6.1.1. If a matching entry is not found, the frame shall be initiated as a non-member mode multicast using the procedure outlined in sub-clause 3.7.6.1.2.

3.7.6.1.1 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 will 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 will then be transmitted according to the procedure described in the final paragraph of sub-clause 3.7.6.2

3.7.6.1.2 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 sub-clause 3.7.6.3, 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 sub-clause 3.7.3.4.1. 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.

3.7.6.2 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 ID field matches the destination group ID of the frame. If a matching entry is found, the message will be passed to the next higher layer, the multicast control field shall be set to 0x01 (member mode), the value of the NonmemberRadius field will be set to the value of the maximum NonmemberRadius field, and the message will 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 multicast NonmemberRadius field is 0 the message will be discarded, along with the newly added BTR. Otherwise, the NonmemberRadius field will be decremented if it is less than 0x07 and the frame will be transmitted as outlined in following paragraphs. 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 *nwkPassiveAckTimeout* seconds 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.

3.7.6.3 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 then determine whether an entry exists in the *nwkGroupIDTable* which group ID field matches the destination group ID 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, then the device shall (re)transmit the frame. If the status is VALIDATION_UNDERWAY, then 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 route 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.

3.7.7 NWK Information in the MAC Beacons

This subclause 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.

When the association permit sub-field of the superframe specification field of the beacon frame of the device, as defined in IEEE 802.15.4-2003 [B1], is set to 1 indicating that association is permitted, then the beacon payload shall contain the information shown in Table 3.51. 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 particular neighbor to join. Refer to sub-clause 3.7.1.3.1.1 for a detailed description of the network discovery procedure. This information is not required to be in the beacon payload when the association permit sub-field of the superframe specification field of the beacon frame of the device is set to 0 indicating that association is not permitted.

Table 3.51 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.
Device depth	Integer	0x00 – *nwkMaxDepth*	The tree depth of this device. A value of 0x00 indicates that this device is the ZigBee coordinator for the network

Table 3.51 NWK Layer Information Fields (Continued)

Name	Type	Valid Range	Description
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 only included when implementing a multi-hop beaconing network

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 anytime the network configuration (any of the parameters specified in Table 3.11) 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 byte sequence representing the beacon payload is written into the *macBeaconPayload* attribute. The formatting of the byte sequence representing the beacon payload is shown in Figure 3.37.

Bits: 0 – 7	**8 – 11**	**12 – 15**	**16 – 17**	**18**	**19 – 22**	**23**	**24 – 87**	**88 – 111**
Protocol ID	Stack profile	nwkcProtocol Version	Reserved	Router capacity	Device depth	End device capacity	*nwk Extended PANID*	Tx Offset (optional)

Figure 3.37 Format of the MAC Sub-Layer Beacon Payload

3.7.7.1 Persistent Data

Devices operating in the field may be reset either manually or programmatically by maintenance personnel, or may be reset 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. At a minimum, 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
- The 64-bit IEEE address and 16-bit network address of each associated child
- The stack profile in use
- The values of nwkNextAddress and nwkAvailableAddresses NIB attributes, if the alternative addressing is in use
- The device's tree depth, if the distributed addressing scheme is in use

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

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 sub-clause 4.2.1. In this clause, the overall security architecture is discussed; basic security services provided by each layer of this architecture are introduced. Sub-clauses 4.2.2, 4.2.3, and 4.2.4 give the ZigBee Alliance's security specifications for the Medium Access Control (MAC) layer, 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. Clause 4.6 describes security elements common to the MAC, NWK, and APS layers. Clause 4.7 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 MAC, 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, security architecture is described. Where applicable, this architecture complements and makes use of 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. In the case of indirect addressing, it is assumed that the binding manager is trusted.

Implementations of security protocols, such as key establishment, are assumed to properly execute the complete protocol and do not leave out any steps hereof. 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. An exception to this assumption occurs when a device that has not been preconfigured joins the network. In this case, a single key may be sent unprotected, thus resulting in a brief moment of vulnerability.

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 and 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 even not possible (barring certification) to verify whether cryptographic separation between different applications on another device, or even between different layers of the communication stack hereof, is indeed properly implemented. Hence, one has to assume that separate applications using the same radio trust each other; that is, there is no cryptographic task separation. In addition, 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 sub-clause 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.

Another consideration is 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 MAC layer disassociate frame needs protection, MAC layer security shall be used. Likewise, if a NWK command frame needs protection, NWK layer security shall be used.
- Second, if protection from theft of service is required (that is, 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 Network key). Thus, only a device that has joined the network and successfully received the 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, NWK layer frames, or MAC layer commands. Reuse of keys helps reduce storage costs.
- Fourth, end-to-end security is enabled such as to make it possible that only source and destination devices have access to their shared key. This limits the trust requirement to those devices whose information is at stake. Additionally, this ensures that routing of messages between devices can be realized independent of trust considerations (thus, facilitating considerable separation of concern).
- Fifth, to simplify interoperability of devices, the security level used by all devices in a given network, and by all layers of a device, shall be the same if

security is used. In particular, the security level indicated in the PIB and NIB shall be the same. If an application needs more security for its payload than is provided by a given network, it shall form its own separate network with a higher security level.

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.

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 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 or a Network key.

A device shall acquire link keys either via key-transport, key-establishment, or pre-installation (for example, during factory installation). A device shall acquire a Network key via key-transport or pre-installation. The key-establishment technique for acquiring a link key (see sub-clause 4.2.4.1) is based on a 'master' key. A device shall acquire a master key (for purposes of establishing corresponding link keys) via key-transport or pre-installation. Ultimately, security between devices depends on the secure initialization and installation of these keys.

In a secured network there are a variety of security services available. Prudence dictates that one would like to avoid re-use of keys across different security services, which otherwise may 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 sub-clause 4.6.3). The use of uncorrelated keys ensures the logical separation of executions of different security protocols. The key-load key is used to protect transported master keys; the key-transport key is used to protect other transported keys. The Network key may be used by the MAC, 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 and master keys may be used only by the APS sublayer. As such, the link and master keys shall be available only to the APL layer.

4.2.1.4 ZigBee Security Architecture

ZigBee applications communicate using the IEEE 802.15.4 wireless standard [B1], which specifies two layers, the Physical (PHY) and Medium Access Control (MAC) layers. ZigBee builds on these layers a Network (NWK) layer and an Application (APL) layer. The PHY layer provides the basic communication capabilities of the physical radio. The MAC layer provides services to enable reliable, single-hop communication links between devices. The ZigBee NWK layer provides routing and multi-hop functions needed for creating different network topologies; for example, star, tree, and mesh structures. The APL layer includes an Application Support (APS) sublayer, the ZigBee Device Object (ZDO), and applications. The ZDO is responsible for overall device management. The APS layer provides a foundation for servicing the ZDO and ZigBee applications.

The architecture includes security mechanisms at three layers of the protocol stack. The MAC, 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, as is the processing of secure MAC frames.

4.2.2 MAC Layer Security

When a frame originating at the MAC layer needs to be secured, ZigBee shall use MAC layer security as specified by the 802.15.4 specification [B1] and augmented by clause 4.3. A security corrigendum proposal [B3] is being developed to augment the MAC layer specification and include the security elements needed by ZigBee. Specifically, at least one of ZigBee's security needs is the ability to protect incoming and outgoing frames using the security levels based on CCM* (see sub-clause 4.6.2.1, and Table 4.30 for a description of ZigBee security levels). CCM* is a minor modification of CCM specified in Clause 7 and Annex B of the 802.15.4 MAC layer specification [B1]. CCM* includes all of the features of CCM and additionally offers encryption-only and integrity-only capabilities. These extra capabilities simplify security by eliminating the need for CTR and CBC-MAC modes. Also, unlike other MAC layer security modes which require a different key for every security level, the use of CCM* enables the use of a single key for all CCM* security levels. With the use of CCM* throughout the ZigBee stack, the MAC, NWK, and APS layers can reuse the same key.

The MAC layer is responsible for its own security processing, but the upper layers shall determine which security level to use. For ZigBee, MAC layer frames requiring security processing shall be processed using the security material from the *macDefaultSecurityMaterial* or the *macACLEntryDescriptorSet* attributes of the MAC PIB. The upper layer (for example, APL) shall set *macDefaultSecurityMaterial* to coincide with the active Network key and counters from the NWK layer and shall set *macACLEntryDescriptorSet* to coincide with any link keys from the APS layer that are shared with neighboring devices, for example, a parent and child. The security suite shall be CCM* and the upper layers shall set the security level to coincide with the *nwkSecurityLevel* attribute in the NIB. For ZigBee, MAC layer link keys shall be preferred, but if not available, the default key (that is, *macDefaultSecurityMaterial*) shall be used. Figure 4.1 shows an example of the security fields that may be included in an outgoing frame with security applied at the MAC level.

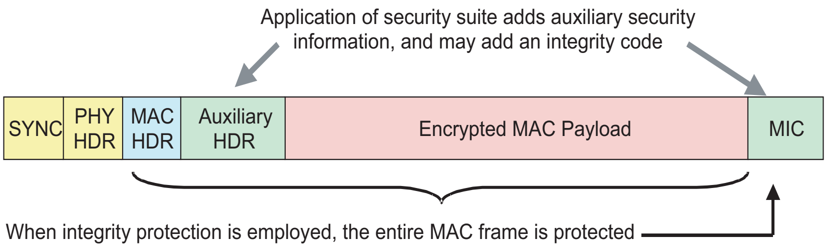

Application of security suite adds auxiliary security information, and may add an integrity code
SYNC
PHY HDR
MAC HDR
Auxiliary HDR
Encrypted MAC Payload
MIC
When integrity protection is employed, the entire MAC frame is protected

Figure 4.1 ZigBee Frame with Security at the MAC Level

4.2.3 NWK Layer Security

When a frame originating at the NWK layer with nwkSecurityLevel > 0, or when a frame originates at a higher layer and the *nwkSecureAllFrames* attribute in the NIB is TRUE, ZigBee shall use the frame protection mechanism specified in sub-clause 4.4.1 of this specification, unless the SecurityEnable parameter of the NLDE-DATA.request primitive is FALSE, explicitly prohibiting security. Like the MAC layer, 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 given 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.

One responsibility of the NWK layer is to route messages over multi-hop links. As part of this responsibility, the NWK layer will broadcast route request messages and process received route reply messages. Route request messages are simultaneously broadcast to nearby devices and route reply messages originate from nearby devices. If the appropriate link key is available, the NWK layer shall use the link key to secure outgoing NWK frames. If the appropriate link key is not available, in order to secure messages against outsiders the NWK layer shall use its active Network key to secure outgoing NWK frames and either its active Network key or an alternate Network key to secure incoming NWK frames.

In this scenario, the frame format explicitly indicates the key used to protect the frame; Thus, intended recipients can deduce which key to use for processing an incoming frame and also determine if the message is readable by all network devices, rather than just by itself.

Figure 4.2 shows an example of the security fields that may be included in a NWK frame.

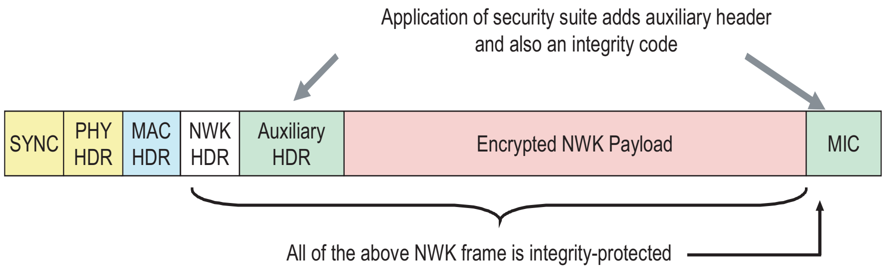

Application of security suite adds auxiliary header and also an integrity code
SYNC
PHY HDR
MAC HDR
NWK HDR
Auxiliary HDR
Encrypted NWK Payload
MIC
All of the above NWK frame is integrity-protected

Figure 4.2 ZigBee Frame with Security on the NWK Level

4.2.4 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 specified in sub-clause 4.5.1 of this specification. The APS layer allows frame security to be based on link keys or the Network key. Figure 4.3 shows an example of the security fields that may be included in an APL frame. Another security responsibility of the APS layer is to provide applications and the ZDO with key establishment, key transport, and device management services.

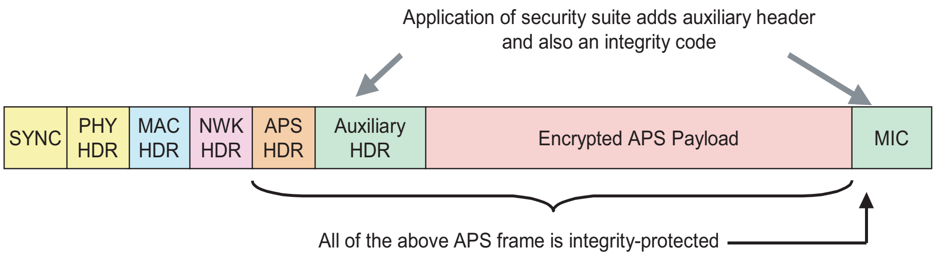

Application of security suite adds auxiliary header and also an integrity code
SYNC
PHY HDR
MAC HDR
NWK HDR
APS HDR
Auxiliary HDR
Encrypted APS Payload
MIC
All of the above APS frame is integrity-protected

Figure 4.3 ZigBee Frame with Security on the APS Level

4.2.4.1 Key Establishment

The APS sublayer's key establishment services provide the mechanism by which a ZigBee device may derive a shared secret key, the so-called link key (see sub-clause 4.2.1.3), with another ZigBee device. Key establishment involves two entities, an initiator device and a responder device, and is prefaced by a trust-provisioning step.

Trust information (for example, a master key) provides a starting point for establishing a link key and can be provisioned in-band or out-band. Once trust information is provisioned, a key-establishment protocol involves three conceptual steps:

- The exchange of ephemeral data;
- The use of this ephemeral data to derive the link key;
- The confirmation that this link key was correctly computed.

In the Symmetric-Key Key Establishment (SKKE) protocol, an initiator device establishes a link key with a responder device using a master key. This master key, for example, may be pre-installed during manufacturing, may be installed by a trust center (for example, from the initiator, the responder, or a third party device acting as a trust center), or may be based on user-entered data (for example, PIN, password, or key). The secrecy and authenticity of the master key needs to be upheld in order to maintain a trust foundation.

4.2.4.2 Transport Key

The transport-key service provides secured and unsecured means to transport a key to another device or other devices. The secured transport-key command provides a means to transport a master, link, or Network key from a key source (for example, trust center) to other devices. The unsecured transport-key command provides a means for loading a device with an initial key. This command does not cryptographically protect the key being loaded. In this case, the security of the transported key can be realized by non-cryptographic means, for example, by communicating the command via an out-of-band channel that guarantees secrecy and authenticity.

4.2.4.3 Update Device

The update-device service provides a secure means for a device (for example, a router) to inform a second device (for example, a 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.

4.2.4.4 Remove Device

The remove device service provides a secure means by which a device (for example, a trust center) may inform another device (for example, a router) that one of its children should be removed from the network. For example, this may be employed to remove a device from the network that has not satisfied the trust center's security requirements for network devices.

4.2.4.5 Request Key

The request-key service provides a secure means for a device to request the current Network key, or an end-to-end application master key, from another device (for example, its trust center).

4.2.4.6 Switch Key

The switch-key service provides a secure means for a device (for example, a trust center) to inform another device that it should switch to a different active Network key.

4.2.5 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 end-to-end application configuration management. All members of the network shall recognize exactly one trust center, and there shall be exactly one trust center in each secure network.

In high-security, commercial applications (see sub-clause 4.7.2.1) a device can be pre-loaded with the trust center address and initial master key (for example, via an unspecified mechanism). Alternatively, if the application can tolerate a moment of vulnerability, the master key can be sent via an in-band unsecured key transport. If not pre-loaded, a device's trust center defaults to the ZigBee coordinator or a device designated by the ZigBee coordinator.

In low-security, residential applications (see sub-clause 4.7.2.2) a device securely communicates with its trust center using the Network key, which can be preconfigured or sent via an in-band unsecured key transport.

The functions performed by the trust center can be subdivided into three sub-roles: trust manager, network manager, and configuration manager. A device trusts its trust manager to identify the device(s) that take on the role of its network and configuration manager. A network manager is responsible for the network and distributes and maintains the Network key to devices it manages. A configuration manager is responsible for binding two applications and enabling end-to-end security between devices it manages (for example, by distributing master keys or link keys). To simplify trust management, these three sub-roles are contained within a single device – the trust center.

For purposes of trust management, a device shall accept an initial master or Network key originating from its trust center via unsecured key transport. For purposes of network management, a device shall accept an initial Network key and updated Network keys only from its trust center (that is, its network manager). For purpose of configuration, a device shall accept master keys or link keys for the purpose of establishing end-to-end security between two devices only from its trust center (that is, its configuration manager). Aside from the initial master key, additional link, master, and Network keys shall only be accepted if they originate from a device's trust center via secured key transport.

4.3 MAC Layer Security

The MAC layer is responsible for the processing steps needed to securely transmit outgoing MAC frames and securely receive incoming MAC 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 MAC frames are described in sub-clauses 4.3.1.1 and 4.3.1.2, respectively.

4.3.1.1 Security Processing of Outgoing Frames

If the MAC layer has a frame requiring security protection, that consists of a header *MacHeader* and payload *Payload*, it shall apply security as follows:

1 Obtain the security material (as specified in sub-clause 4.3.2), including the key, outgoing frame counter *FrameCount*, key sequence count *SeqCount*, and security level identifier (as specified in Table 4.30) from the MAC PIB using the following procedure. If the outgoing frame counter has as its value the 4-

octet representation of the integer 2^{32}-1 or any of this security material cannot be determined, then security processing shall fail and no further security processing shall be done on this frame.

a First, an attempt shall be made to retrieve the security material and security level identifier associated with the destination address of the outgoing frame from the *macACLEntryDescriptorSet* attribute in the MAC PIB.

b If the first attempt fails, then security material shall be obtained by using the *macDefaultSecurityMaterial* attribute from the MAC PIB and the security level identifier shall be obtained from the *MacDefaultSecuritySuite* attribute from the MAC PIB.

2 The Security Control Field *SecField* is the 1-octet field formatted as in sub-clause 4.6.1.1, with the following settings:

a The security level subfield shall be set to the security level obtained in step 1;

b The key identifier subfield shall be set to the 2-bit field '00';

c The extended nonce subfield shall be set to the 1-bit field '0';

d The reserved bits shall be set to the 2-bit field '00'.

3 Execute the CCM* mode encryption and authentication operation, as specified in Annex A, with the following instantiations:

a The parameter *M* shall be obtained from Table 4.30 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 local device's 64-bit extended address, *SecField* from step 1, and *FrameCount* from step 1 (see Figure 72 from [B1]);

d If the security level requires encryption, the octet string *a* shall be the string *MacHeader* and the octet string *m* shall be the string *Payload*. Otherwise, the octet string *a* shall be the string *MacHeader* || *Payload* and the octet string *m* shall be a string of length zero. Note that ZigBee interprets IEEE Std. 802.15.4 802 [B1] to mean that frame counters are authenticated.

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 4. If the security level requires encryption, the secured outgoing frame shall be *MacHeader* || *FrameCount* || *SeqCount* || *c,* otherwise the secured outgoing frame shall be *MacHeader* || *FrameCount* || *SeqCount* || *Payload* || *c.*

6 If the secured outgoing frame size is greater than *aMaxPHYPacketSize* (from [B1]), 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 location from which the security material was obtained in step 1 (that is, either the *macDefaultSecurityMaterial* attribute or the *MacDefaultSecuritySuite* attribute).

4.3.1.2 Security Processing of Incoming Frames

If the MAC layer receives a secured frame (consisting of a header *MacHeader*, frame count *ReceivedFrameCount*, sequence count *ReceivedSeqCount*, and payload *SecuredPayload*) it shall perform security processing as follows:

1 If *ReceivedFrameCount* has as value 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 Obtain the security material (as specified in sub-clause 4.3.2), including the key, optional external frame counter *FrameCount*, optional key sequence count *SeqCount*, and security level identifier (as specified in Table 4.30) from the MAC PIB using the following procedure. If the security material cannot be obtained or if *SeqCount* exists and does not match *ReceivedSeqCount*, security processing shall fail and no further security processing shall be done on this frame.

- a First, an attempt shall be made to retrieve the security material and security level identifier associated with the source address of the incoming frame from the *macACLEntryDescriptorSet* attribute in the MAC PIB.
- b If the first attempt fails, then security material shall be obtained by using the *macDefaultSecurityMaterial* attribute from the MAC PIB and the security level identifier shall be obtained from the *MacDefaultSecuritySuite* attribute from the MAC PIB.

3 If *FrameCount* exists and if *ReceivedFrameCount* is less than *FrameCount*, security processing shall fail and no further security processing shall be done on this frame.

4 The Security Control Field *SecField* is the 1-octet field formatted as in sub-clause 4.6.1.1, Table 4.17, with the following settings:

- a The security level subfield shall be set to the security level from the MACPIB (as specified in Table 4.30);
- b The key identifier subfield shall be set to the 2-bit field '00';
- c The extended nonce subfield shall be set to the 1-bit field '0';

   d The reserved bits shall be set to the 2-bit field '00'.

5 Execute the CCM* mode decryption and authentication checking operation, as specified in sub-clause A.3, with the following instantiations:

   a The parameter *M* shall be obtained from Table 4.30 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 64-bit extended sender address, *SecField* from step 4, and *ReceivedFrameCount* from step 1 (see Figure 72 from [B1]); The nonce N shall be formatted according to the endianness convention used in this specification (the octet containing the lowest numbered bits first to the octet containing the higher numbered bits).

   d Parse the octet string *SecuredPayload* as $Payload_1 \| Payload_2$, where the right-most 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;

   e If the security level requires encryption, the octet string *a* shall be the string *MacHeader* || *ReceivedFrameCount* || *ReceivedSeqCount* and the octet string *c* shall be the string *SecuredPayload*. Otherwise, the octet string *a* shall be the string *MacHeader* || *ReceivedFrameCount* || *ReceivedSeqCount* || $Payload_1$ and the octet string *c* shall be the string $Payload_2$.

6 Return the results of the CCM* operation:

   a If the CCM* mode invoked in step 5 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 5. If the security level requires encryption, set the octet string *UnsecuredMacFrame* to the string *MACHeader* || *m*. Otherwise, set the octet string *UnsecuredMacFrame* to the string *MACHeader* || Payload_1.

7 If the optional *FrameCount* (obtained in step 2) exists, set it to *ReceivedFrameCount* and update MAC PIB. *UnsecuredMacFrame* now represents the unsecured received MAC layer frame.

4.3.2 Security-related MAC PIB Attributes

The security-related MAC PIB attributes shall be those as defined in Table 72 of [B1]. The security material used for CCM* mode shall the same as given for CCM mode in Figure 70 of IEEE Std. 802.15.4 802 [B1].

For the *macDefaultSecurityMaterial* attribute from the MAC PIB, the upper layer shall set the symmetric key, outgoing frame counter, and optional external key sequence counter equal to the corresponding elements of the network security material descriptor in the *nwkSecurityMaterialSet* of the NIB referenced by the *nwkActiveKeySeqNumber* attribute of the NIB. The optional external frame counter shall not be used and the optional external key sequence counter shall correspond to the sequence number of the Network key.

For the *macACLEntryDescriptorSet* attribute from the MAC PIB, the upper layer shall set the symmetric key, and outgoing frame counter equal to the corresponding elements of the Network key-pair descriptor in the *apsDeviceKeyPairSet* of the AIB. The optional external frame counter shall be set to the incoming frame counter, The key sequence counter shall be set to 0x00, and the optional external key sequence counter shall not be used.

4.4 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.

4.4.1 Frame Security

The detailed steps involved in security processing of outgoing and incoming NWK frames are described in sub-clauses 4.4.1.1 and 4.4.1.2, respectively.

4.4.1.1 Security Processing of Outgoing Frames

If the NWK layer has a frame, consisting of a header *NwkHeader* and payload *Payload*, that needs security protection and *nwkSecurityLevel* > 0, 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 has as its value the 4-octet representation of the 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.
2. Construct auxiliary header *AuxiliaryHeader* (see sub-clause 4.6.1):
   a. The security control field shall be set 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 Network key).

iii The extended nonce sub-field shall be set to 1.

iv The source address field shall be set to the 64-bit extended address of the local device.

b The frame counter field shall be set to the outgoing frame counter from step 1.

c The key sequence number field shall be set 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 The parameter *M* shall be obtained from Table 4.30 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 c, and the source address field from step b (see sub-clause 4.6.2.2);

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* (see IEEE Std. 802.15.4 802 [B1]), 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 *nwkActiveKeySeqNumber* in the NIB; that is, the outgoing frame counter value associated with the key used to protect the frame is updated.

8 Over-write the security level subfield of the security control field by the 3-bit all-zero string '000'.

4.4.1.2 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 from the NIB. Over-write the 3-bit security level subfield of the security control field of the AuxillaryHeader with this value. Determine the sequence number SequenceNumber, sender address SenderAddress, and received frame count ReceivedFrameCount from the auxiliary header AuxiliaryHeader (see sub-clause 4.6.1). If ReceivedFrameCounter has as value 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 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 SequenceNumber does not equal *nwkActiveKeySeqNumber*, the neighbor table entry corresponding to SenderAddress shall be modified to show that the network key is stale. If the security material cannot be obtained, security processing shall fail and no further security processing shall be done on this frame.[16]

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 fail and no further security processing shall be done on this frame.

4 Execute the CCM* mode decryption and authentication checking operation, as specified in sub-clause A.2, with the following instantiations:

a The parameter *M* shall obtained from Table 4.30 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 sub-clause 4.6.2.2). 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.

16. CCB #666

d Parse the octet string *SecuredPayload* as $Payload_1$ || $Payload_2$, where the right-most 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;

e If the security level requires encryption, 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* || $Payload_1$ and the octet string *c* shall be the string $Payload_2$.

5 Return the results of the CCM* operation:

a If the CCM* mode invoked in step 4 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 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* || $Payload_1$.

6 Set *FrameCount* to (*ReceivedFrameCount* + 1) and store both *FrameCount* and *SenderAddress* in the NIB. *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*, then the *nwkActiveKeySeqNumber* shall be set to received sequence number.[17]

4.4.2 Secured NPDU Frame

The NWK layer frame format (see sub-clause 3.4.1) 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 sub-clause 4.6.1. The

17. CCB #666

format of a secured NWK layer frame is shown in Figure 4.4. The auxiliary frame header is situated between the NWK header and payload fields.

Octets: Variable	**14**	**Variable**	
Original NWK Header ([B3], Clause 7.1)	Auxiliary frame header	Encrypted Payload	Encrypted Message Integrity Code (MIC)
		Secure frame payload = Output of CCM*	
Full NWK header		Secured NWK payload	

Figure 4.4 Secured NWK Layer Frame Format

4.4.3 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 NLME-GET.request and NLME-SET.request primitives, respectively. The security-related attributes contained in the NWK PIB are presented in Tables 4.1 through 4.3.

Table 4.1 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 4.30	0x06
nwkSecurity-MaterialSet	0xa1	A set of 0, 1, or 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	-
nwkActiveKey SeqNumber	0xa2	Octet	0x00-0xFF	The sequence number of the active Network key in *nwkSecurityMaterialSet*	0x00

Table 4.1 NIB Security Attributes (Continued)

Attribute	Identifier	Type	Range	Description	Default
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	TRUE
nwkSecureAll Frames	0xa5	Boolean	TRUE | FALSE	This indicates if security shall be applied to incoming and outgoing NWK data frames; if set to 0x01, security processing shall be applied to all incoming and outgoing frames except data frames destined for the current device that have the security sub-field of the frame control field set to 0; if this attribute has a value of 0x01 the NWK layer shall not relay frames that have the security sub-field of the frame control field set to 0; The SecurityEnable parameter of the NLDE-DATA.request primitive shall override the setting of this attribute	TRUE

Table 4.2 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	00
OutgoingFrame Counter	Ordered set of 4 octets	0x00000000-0xFFFFFFFF	Outgoing frame counter used for outgoing frames	0x00000000
IncomingFrame CounterSet	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	-

4.5 APS Layer Security

Table 4.3 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

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 4.4 lists the primitives available for key management and maintenance. Upper layers also determine which security level to use when protecting outgoing frames.

Table 4.4 The APS Layer Security Primitives

APSME Security Primitives	Request	Confirm	Indication	Response	Description
APSME-ESTABLISH-KEY	4.5.2.1	4.5.2.2	4.5.2.3	4.5.2.4	Establishes link key with another ZigBee device using the SKKE method
APSME-TRANSPORT-KEY	4.5.3.1	-	4.5.3.2	-	Transports security material from one device to another
APSME-UPDATE-DEVICE	4.5.4.1	-	4.5.4.2	-	Notifies the trust center when a new device has joined, or an existing device has left the network
APSME-REMOVE-DEVICE	4.5.5.1	-	4.5.5.2	-	Used by the trust center to notify a router that one of the router's child devices should be removed from the network
APSME-REQUEST-KEY	4.5.6.1	-	4.5.6.2	-	Used by a device to request that the trust center send an application master key or current Network key
APSME-SWITCH-KEY	4.5.7.1	-	4.5.7.2	-	Used by the trust center to tell a device to switch to a new Network key

4.5.1 Frame Security

The detailed steps involved in security processing of outgoing and incoming APS frames are described in sub-clauses 4.5.1.1 and 4.5.1.2, respectively.

4.5.1.1 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. If the *TxOptions* parameter specifies that the NWK key is required to secure the data frame, then security material shall be obtained by using the *nwkActiveKeySeqNumbe*r from the NIB to retrieve the active Network key, outgoing frame counter, and sequence number from the *nwkSecurityMaterialSet* attribute in the NIB. *KeyIdentifier* shall be set to '01' (that is, the Network key).
      ii. Otherwise, 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). Note, if the frame is being transmitted using indirect addressing, the destination address shall be the address of the binding manager.
   b. If the frame is a result of an APS command:
      i. First, 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. 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, application master key, or trust center master key. See sub-clause 4.6.3 for a description of the key-transport and key-load keys.
      ii. If the first attempt fails, then security material shall be obtained by using the *nwkActiveKeySeqNumbe*r from the NIB to retrieve the active Network key, outgoing frame counter, and sequence number from the *nwkSecurityMaterialSet* attribute in the NIB. *KeyIdentifier* shall be set to '01' (that is, the Network key).
2. If the key identifier is equal to 01 (that is, network key), the APS layer shall first verify that the NWK layer is not also applying security. If the NWK layer is applying security, then the APS layer shall not apply any security. The APS layer can determine that the NWK layer is applying security by verifying that the value of the *nwkSecureAllFrames* attribute of the NIB has a value of TRUE and the *nwkSecurityLevel* NIB attribute has a non-zero value.

3 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 has as its value the 4-octet representation of the 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.

4 Obtain the security level from the *nwkSecurityLevel* attribute from the NIB. If the frame is a result of an APS command, the security level identifier shall be forced to 7 (ENC-MIC-128, as indicated in Table 4.30.)

5 Construct auxiliary header *AuxiliaryHeader* (see sub-clause 4.6.1): the security control field shall be set as follows:

   a The security level sub-field shall be the security level obtained from step 4.

      i The key identifier sub-field shall be set to *KeyIdentifier*.

      ii The extended nonce sub-field shall be set to 0.

   b The frame counter field shall be set to the outgoing frame counter from step 3.

   c If *KeyIdentifier* is 1, 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.

6 Execute the CCM* mode encryption and authentication operation, as specified in sub-clause A.2, with the following instantiations:

   a The parameter *M* shall obtained from Table 4.30 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 sub-clause 4.6.2.2);

   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.

7 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.

8 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*.

9 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.

10 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.

11 Over-write the security level subfield of the security control field by the 3-bit all-zero string '000'.

4.5.1.2 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 23^2-1, 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 AIB, using the source address in the APS frame as the index. If the source address is incomplete or unavailable, security processing shall fail and no further security processing shall be done on this frame. If the delivery-mode sub-field of the frame control field of *ApsHeader* has a value of 1 (that is, indirect addressing), the source address shall be the address of the binding manager, as described in the APS specification [B7].

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’ (that is, 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 *KeyIdentifier* is ‘01’ (that is, Network key), the security material shall be obtained by matching *SequenceNumber* to the sequence number of any key in the *nwkSecurityMaterialSet* attribute in the NIB.

c If *KeyIdentifier* is ‘02’ (that is, 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 sub-clause 4.6.3 for the key-transport key.

d If *KeyIdentifier* is ‘03’ (that is, 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 sub-clause 4.6.3 for the key-load key.

4 If 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 subfield of the frame control field of *ApsHeader* indicates an APS data frame, then *SecLevel* shall be set to the *nwkSecurityLevel* attribute in the NIB. Otherwise *SecLevel* shall be set to 7 (ENC-MIC-128). Overwrite the security level subfield of the security control field in the *AuxillaryHeader* with the value of *SecLevel*.

6 Execute the CCM* mode decryption and authentication checking operation, as specified in sub-clause A.3, with the following instantiations:

a The parameter *M* shall obtained from Table 4.30 corresponding to the security level from step 5;

b The bit string *Key* shall be the key obtained from step 3;

c 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 sub-clause 4.6.2.2);

d Parse the octet string *SecuredPayload* as $Payload_1 \| Payload_2$, where the right-most 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;

e 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.

9 If the sequence number of the received frame belongs to a newer entry in the *nwkSecurityMaterialSet,* and the source address of the packet is the trust center, then the *nwkActiveKeySeqNumber* may be set to received sequence number. If the security material associated with the *SourceAddress* of the incoming frame cannot be obtained from the attribute in the AIB, then security processing shall fail and no further security processing shall be done on this frame.

4.5.2 Key-establishment Services

The APSME provides services that allow two devices to mutually establish a link key. Initial trust information (for example, a master key) must be installed in each device prior to running the key establishment protocol (see sub-clause 4.5.3 for mechanisms to provision initial trust information).

4.5.2.1 APSME-ESTABLISH-KEY.request

The APSME-ESTABLISH-KEY request primitive is used for initiating a key-establishment protocol. This primitive can be used when there is a need to securely communicate with another device.

One device will act as an initiator device, and another device will act as the responder. The initiator shall start the key-establishment protocol by issuing:

- The APSME-ESTABLISH-KEY.request with parameters indicating the address of the responder device;
- An indication of which key-establishment protocol to use (that is, SKKE direct or indirect).

4.5.2.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface

```
APSME-ESTABLISH-KEY.request          {
                                     ResponderAddress,
                                     UseParent,
                                     ResponderParentAddress,
                                     KeyEstablishmentMethod
                                     }
```

Table 4.5 specifies the parameters for the APSME-ESTABLISH-KEY.request primitive.

Table 4.5 APSME-ESTABLISH-KEY.request Parameters

Parameter Name	Type	Valid Range	Description
Responder-Address	Device Address	Any valid 64-bit address	The extended 64-bit address of the responder device
UseParent	Boolean	TRUE | FALSE	This parameter indicates if the responder's parent shall be used to forward messages between the initiator and responder devices: TRUE: Use parent FALSE: Do not use parent
Responder-ParentAddress	Device Address	Any valid 64-bit address	If the *UseParent* is TRUE, then *ResponderParentAddress* parameter shall contain the extended 64-bit address of the responder's parent device; Otherwise, this parameter is not used and need not be set
KeyEstablishment-Method	Integer	0x00 - 0x03	The requested key-establishment method shall be one of the following: 0x00 = SKKE method 0x01-0x03: reserved

4.5.2.1.2 When Generated

A higher layer on an initiator device shall generate this primitive when it requires a link key to be established with a responder device. If the initiator device wishes to use the responder's parent as a liaison (for NWK security purposes), it shall set the *UseParent* parameter to TRUE and shall set the *ResponderParentAddress* parameter to the 64-bit extended address of the responder's parent.

4.5.2.1.3 Effect on Receipt

The receipt of an APSME-ESTABLISH_KEY.request primitive, with the KeyEstablishmentMethod parameter equal to SKKE, shall cause the APSME to execute the SKKE protocol, as described in sub-clause 4.5.2.6. The local APSME shall act as the initiator of this protocol, the APSME indicated by the *ResponderAddress* parameter shall act as the responder of this protocol, and the *UseParent* parameter will control whether the messages are sent indirectly via the responder's parent device given by the *ResponderParentAddress* parameter.

4.5.2.2 APSME-ESTABLISH-KEY.confirm

This primitive is issued to the ZDO upon completion or failure of a key-establishment protocol.

4.5.2.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-ESTABLISH-KEY.confirm            {
                                       Address,
                                       Status
                                       }
```

Table 4.6 specifies the parameters of the APSME-ESTABLISH-KEY.confirm primitive. Table 4.10 gives a description of some codes that can be returned in the *Status* parameter of this primitive. In addition to these codes, if, when sending one of the protocol messages, an NLDE-DATA.confirm primitive with a *Status* parameter set to a value other than SUCCESS is issued, the *Status* parameter of the APSME-ESTABLISH-KEY.confirm primitive shall be set to that received from the NWK layer.

Table 4.6 APSME-ESTABLISH-KEY.confirm Parameters

Name	Type	Valid Range	Description
Address	Device Address	Any valid 64-bit address	The extended 64-bit address of the device with which the key-establishment protocol was executed
Status	Enumeration	Value given by Table 4.10 or any status value returned from the NLDE-DATA.confirm primitive	This parameter indicates the final status of the key-establishment protocol

4.5.2.2.2 When Generated

The APSME in both the responder and initiator devices shall issue this primitive to the ZDO upon completion of a key-establishment protocol.

4.5.2.2.3 Effect on Receipt

If key establishment is successful, the AIB of the initiator and responder shall be updated with the new link key and the initiator shall be able to securely communicate with the responder. If the key establishment was not successful, then the AIB shall not be changed.

4.5.2.3 APSME-ESTABLISH-KEY.indication

The APSME in the responder shall issue this primitive to its ZDO when it receives an initial key-establishment message (for example, an SKKE-1 frame) from an initiator.

4.5.2.3.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-ESTABLISH-KEY.indication          {
                                        InitiatorAddress,
                                        KeyEstablishmentMethod
                                        }
```

Table 4.7 specifies the parameters of the APSME-ESTABLISH-KEY.indication primitive.

Table 4.7 APSME-ESTABLISH-KEY.indication Parameters

Name	Type	Valid Range	Description
InitiatorAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the initiator device
KeyEstablishmentMethod	Integer	0x00 - 0x03	The requested key-establishment method shall be one of the following: 0x00 = SKKE method 0x01-0x03: reserved

4.5.2.3.2 When Generated

The APSME in the responder device shall issue this primitive to the ZDO when a request to start a key-establishment protocol (for example, an SKKE-1 frame) is received from an initiator and a master key associated with the initiator device is present in the AIB.

4.5.2.3.3 Effect on Receipt

Upon receiving the APSME-ESTABLISH-KEY.indication primitive, the ZDO may use the *KeyEstablishmentMethod* and *InitiatorAddress* parameters to determine whether to establish a key with the initiator. The ZDO shall respond using the APSME-ESTABLISH-KEY.response primitive.

4.5.2.4 APSME-ESTABLISH-KEY.response

The ZDO of the responder device shall use the APSME-ESTABLISH-KEY.response primitive to respond to an APSME-ESTABLISH-KEY.indication primitive. The ZDO determines whether to continue with the key establishment or halt it. This decision is indicated in the Accept parameter of the APSME-ESTABLISH-KEY.response primitive.

4.5.2.4.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-ESTABLISH-KEY.response          {
                                      InitiatorAddress,
                                      Accept
                                      }
```

Table 4.8 specifies the parameters of the APSME-ESTABLISH-KEY.response primitive

Table 4.8 APSME-ESTABLISH-KEY.response Parameters

Name	Type	Valid Range	Description
InitiatorAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the device that initiated key establishment

Accept	Boolean	TRUE | FALSE	This parameter indicates the response to an initiator's request to execute a key-establishment protocol. The response shall be either:
			TRUE = Accept
			FALSE = Reject

4.5.2.4.2 When Generated

The APSME-ESTABLISH-KEY.response primitive shall be generated by the ZDO and provided to the APSME following a request from an initiator device to start a key-establishment protocol (that is, after receipt of an APSME-ESTABLISH-KEY.indication). This primitive provides the responder's ZDO with an opportunity to determine whether to accept or reject a request to establish a key with a given initiator.

4.5.2.4.3 Effect on Receipt

If the *Accept* parameter is TRUE, then the APSME of the *responder* will attempt to execute the key establishment protocol indicated by the *KeyEstablishmentMethod* parameter. If *KeyEstablishmentMethod* is equal to SKKE, the APSME shall execute the SKKE protocol, described in sub-clause 4.5.2.6. The local APSME shall act as the responder of this protocol and the APSME indicated by the *InitiatorAddress* parameter shall act as the initiator of this protocol. If the *Accept* parameter is FALSE, the local APSME shall halt and erase all intermediate data pertaining to the pending key-establishment protocol.

4.5.2.5 Data Service Message Sequence Chart

Figure 4.5 illustrates the sequence of primitives necessary for a successful key establishment between two devices.

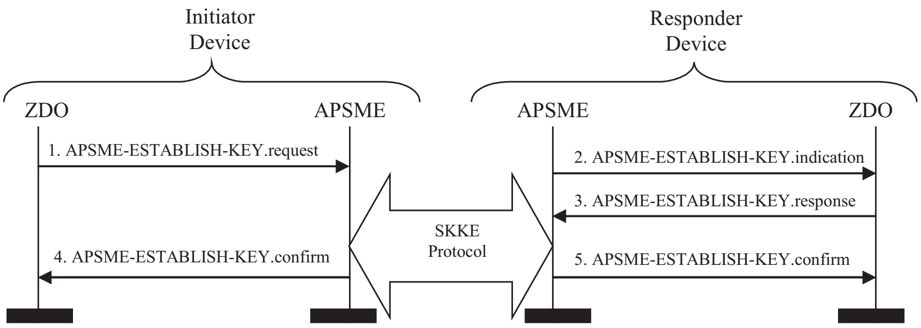

Initiator Device
Responder Device
ZDO
APSME
APSME
ZDO
1. APSME-ESTABLISH-KEY.request
2. APSME-ESTABLISH-KEY.indication
3. APSME-ESTABLISH-KEY.response
SKKE Protocol
4. APSME-ESTABLISH-KEY.confirm
5. APSME-ESTABLISH-KEY.confirm

Figure 4.5 Sequence Chart for Successful APSME-ESTABLISH-KEY Primitives

4.5.2.6 The SKKE Protocol

The APSME on the initiator and responder execute the symmetric-key key-agreement scheme instantiated in sub-clause B.2.1 and specified in B.1. The shared key (as specified in B.1 prerequisite step 2), shall be the master key shared between the initiator and responder devices as obtained from the appropriate master key element in the *DeviceKeyPairSet* attribute in the AIB. The messages sent during the scheme specified in B.1 shall be assigned to the frame names given in Table 4.9. The formats for these SKKE frames are given in sub-clause 4.5.8.1. The initiator device is responsible for sending the SKKE-1 and SKKE-3 frames and the responder device is responsible for sending the SKKE-2 and SKKE-4 frames. Additionally, if the *UseParent* parameter to the APSME-ESTABLISH-KEY.request primitive is TRUE, the responder device's parent (as indicated by the *ResponderParentAddress* parameter to the APSME-

ESTABLISH-KEY.request primitive) shall act as a liaison and forward messages between the initiator and responder devices.

During the key-establishment scheme, if the responder or initiator device detects any error condition listed in Table 4.10, the scheme shall be aborted and the local APSME shall issue the APSME-ESTABLISH-KEY.confirm primitive with the *Status* parameter set as indicated in Table 4.10. If no error conditions occur (that is, the key-agreement scheme outputs 'valid'), then the initiator and responder shall consider the derived key (that is, *KeyData*) as their newly shared link key. Both the initiator and responder shall update or add this link key to their AIB, set the corresponding incoming and outgoing frame counts to zero, and issue the APSME-ESTABLISH-KEY.confirm primitive with the *Status* parameter set to SUCCESS.

Table 4.9 Mapping of Frame Names to Symmetric-key Key Agreement Scheme Messages

Frame Name	Description	Reference
SKKE-1	Sent by initiator during action step 1 (B.7.1)	4.5.2.6.2
SKKE-2	Sent by responder during action step 2 (B.7.2)	4.5.2.6.3
SKKE-3	Sent by initiator during action step 8 (B.7.1)	4.5.2.6.4
SKKE-4	Sent by responder during action step 9 (B.7.2)	4.5.2.6.5

Table 4.10 Mapping of Symmetric-key Key Agreement Error Conditions to Status Codes

Status Description	Status Code	Value
No errors occur	SUCCESS	0x00
An invalid parameter was input to one of the key establishment primitives	INVALID_PARAMETER	0x01
No master key is available	NO_MASTER_KEY	0x02
Challenge is invalid: Initiator during action step 3 (B.7.1) Responder during action step 3 (B.7.2)	INVALID_CHALLENGE	0x03
SKG outputs invalid: Initiator during action step 4 (B.7.1) Responder during action step 3 (B.7.2)	INVALID_SKG	0x04
MAC transformation outputs invalid: Initiator during action step 8 (B.7.1) Responder during action step 10 (B.7.2	INVALID_MAC	0x05
Tag checking transformation outputs invalid: Initiator during action step 12 (B.7.1) Responder during action step 8 (B.7.2)	INVALID_KEY	0x06
Either the initiator or responder waits for an expected incoming message for time greater than the *apsSecurityTimeOutPeriod* attribute of the AIB	TIMEOUT	0x07
Either the initiator or responder receives an SKKE frame out of order	BAD_FRAME	0x08

4.5.2.6.1 Generating and Sending the Initial SKKE-1 Frame

The SKKE protocol begins with the initiator device sending an SKKE-1 frame. The SKKE-1 command frame shall be constructed as specified in sub-clause 4.5.8.1.

If the *UseParent* parameter to the APSME-ESTABLISH-KEY.request primitive is FALSE, the initiator device shall begin the protocol by sending this SKKE-1 frame directly to the responder device (as indicated by the *ResponderAddress* parameter to the APSME-ESTABLISH-KEY.request primitive). Otherwise, the initiator device shall begin the protocol by sending this SKKE-1 frame to the responder device's parent (as indicated by the *ResponderParentAddress* parameter to the APSME-ESTABLISH-KEY.request primitive). The SKKE-1 frame shall be sent using the NLDE-DATA.request primitive with NWK layer security set to the default NWK layer security level.

4.5.2.6.2 On Receipt of the SKKE-1 Frame

If the responder address field of the SKKE-1 frame does not equal the local device address, the APSME shall perform the following steps:

1. If the device given by the responder address field is not a child of the local device, the SKKE-1 frame shall be discarded.
2. Otherwise, the APSME of the local device shall send the SKKE-1 frame to the responder device using the NLDE-DATA.request primitive with:
   - The *DestAddr* parameter set to the 16-bit address corresponding to the 64-bit address in the responder address field of the SKKE-1 frame;
   - The *DiscoverRoute* parameter set to 0x01;
   - The *SecurityEnable* parameter set to FALSE.
3. Otherwise, the APSME shall perform the following steps:
   - i If the device does not have a master key corresponding to the initiator address field, the SKKE-1 frame shall be discarded and the APSME-ESTABLISH-KEY.confirm primitive shall be issued with the *Status* parameter set to NO_MASTER_KEY (see Table 4.10). The APSME should halt processing for this SKKE protocol.
   - ii Otherwise, the APSME shall issue an APSME-ESTABLISH-KEY.indication primitive with the *InitiatorAddress* parameter set to the initiator address field of the SKKE-1 frame and the *KeyEstablishmentMethod* parameter set to 0 (that is, the SKKE protocol).
   - iii After issuing the APSME-ESTABLISH-KEY.indication primitive, and upon receipt of the corresponding APSME-ESTABLISH-KEY.response primitive, the APSME shall evaluate the *InitiatorAddress* and *Accept* parameters of the received APSME-ESTABLISH-KEY.response primitive. If the *InitiatorAddress* parameter is set to the initiator address of the SKKE-1 frame and the *Accept* parameter set to FALSE, the APSME shall halt the SKKE protocol and discard the SKKE-1 frame.
   - iv Otherwise, it shall construct an SKKE-2 frame as specified in sub-clause 4.5.8.1. If the source of the SKKE-1 frame indicates the same device as the initiator address field of the SKKE-1 frame, the device shall send this SKKE-2 frame directly to the initiator device using the NLDE-DATA.request primitive, with the *DestAddr* parameter set to the source of the SKKE-1 frame, the *DiscoverRoute* parameter set to 0x01, and the *SecurityEnable* parameter set to TRUE. Otherwise, the device shall send the SKKE-2 frame to its parent using the NLDE-DATA.request primitive, with the *DiscoverRoute* parameter set to 0x01, and the *SecurityEnable* parameter set to FALSE.[18]

4.5.2.6.3 On Receipt of the SKKE-2 Frame

If the initiator address field of the SKKE-2 frame does not equal the local device address, the APSME shall perform the following steps:

1. If the device given by the responder address field is not a child of the local device, the SKKE-2 frame shall be discarded.
2. Otherwise, the device shall send the SKKE-2 to the initiator device using the NLDE-DATA.request primitive with NWK layer set to the default level.

Otherwise, the device shall construct an SKKE-3 frame as specified in sub-clause 4.5.8.1. If the source of the SKKE-2 frame is the same as the responder address field of the SKKE-2 frame, the device shall send this SKKE-3 frame directly to the responder device. Otherwise, the device shall send the SKKE-3 frame to the responder's parent. The SKKE-3 frame shall be sent using the NLDE-DATA.request primitive with NWK layer security set to the default NWK layer security level.

4.5.2.6.4 On Receipt of the SKKE-3 Frame

If the responder address field of the SKKE-3 frame does not equal the local device address, the APSME shall perform the following steps:

1. If the device given by the responder address field is not a child of the local device, the SKKE-3 frame shall be discarded;
2. Otherwise, the device shall send the SKKE-3 to the responder device using the NLDE-DATA.request primitive with NWK layer security disabled;
3. Otherwise, the device shall process the SKKE-3 data field and if the protocol was not a success it shall issue an APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set to the initiator's address and the *Status* parameter set appropriately;
4. If, from the device's perspective, the protocol was a success, the device shall construct an SKKE-4 frame as specified in sub-clause 4.5.8.1. If the source of the SKKE-3 frame is the same as the initiator address field of the SKKE-3 frame, the device shall send this SKKE-4 frame directly to the initiator device using the NLDE-DATA.request primitive with NWK layer security set to the default level. Otherwise, the device shall send the SKKE-4 frame to its parent using the NLDE-DATA.request primitive with NWK layer security disabled. Finally, the device shall issue an APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set the initiator's address and the *Status* parameter set to success.

18. CCB #437

4.5.2.6.5 On Receipt of the SKKE-4 Frame

If the initiator address field of the SKKE-4 frame does not equal the local device address, the APSME shall perform the following steps:

1. If the device given by the responder address field is not a child of the local device, the SKKE-4 frame shall be discarded.
2. Otherwise, the APSME of the local device shall send the SKKE-4 to the initiator device using the NLDE-DATA.request primitive with NWK layer set to the default level.

Otherwise, the APSME shall process the SKKE-4 frame and issue an APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set the responder's address and the *Status* parameter set appropriately.

4.5.3 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 Tables 4.12 to 4.15.

4.5.3.1 APSME-TRANSPORT-KEY.request

The APSME-TRANSPORT-KEY.request primitive is used for transporting a key to another device.

4.5.3.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-TRANSPORT-KEY.request         {
                                    DestAddress,
                                    KeyType,
                                    TransportKeyData
                                    }
```

Table 4.11 specifies the parameters for the APSME-TRANSPORT-KEY.request primitive.

Table 4.11 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
KeyType	Integer	0x00 – 0x03	Identifies the type of key material that should be transported; see Table 4.12
TransportKeyData	Variable	Variable	The key being transported along with identification and usage parameters. The type of this parameter depends on the *KeyType* parameter as follows: *KeyType* = 0x00 see Table 4.12 *KeyType* = 0x01 see Table 4.13 *KeyType* = 0x02 see Table 4.14 *KeyType* = 0x03 see Table 4.15

Table 4.12 *KeyType* Parameter of The Transport-Key Primitive

Enumeration	Value	Description
Trust-center master key	0x00	Indicates the key is a master key which is used to set up link keys between the trust center and another device
Network key	0x01	Indicates the key is a Network key
Application master key	0x02	Indicates the key is a master key which is used to set up link keys between two devices
Application link key	0x03	Indicates the key is a link key which is used as a basis of security between two devices

Table 4.13 *TransportKeyData* Parameter for a Trust Center Master Key

Parameter Name	Type	Valid Range	Description
ParentAddress	Device address	Any valid 64-bit address	The extended 64-bit address of the parent of the destination device given by the *DestAddress* parameter
TrustCenter-MasterKey	Set of 16 octets	Variable	The trust center master key

Table 4.14 ***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 4.15 ***TransportKeyData* Parameter for an Application Master or 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 master key
Initiator	Boolean	TRUE | FALSE	This parameter indicates if the destination device of this master key requested it: TRUE: If the destination requested the key FALSE: Otherwise
Key	Set of 16 octets	Variable	The master or link key (as indicated by the *KeyType* parameter)

4.5.3.1.2 When Generated

The ZDO on an initiator device shall generate this primitive when it requires a key to be transported to a responder device.

4.5.3.1.3 Effect on Receipt

The receipt of an APSME-TRANSPORT-KEY.request primitive shall cause the APSME to create a transport-key command packet (see sub-clause 4.5.8.2)

If the *KeyType* parameter is 0x00 (that is, trust center master 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 sub-clause 4.5.1.1. Then, if security processing succeeds, sent to the device specified by the *ParentAddress* sub-parameter of the *TransportKeyData* parameter by issuing a NLDE-DATA.request primitive.

If the *KeyType* parameter is 0x01 (that is, 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 sub-clause 4.5.1.1 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 *KeyType* parameter is 0x02 or 0x03 (that is, an application master or 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 4.5.1.1 and then, if security processing succeeds, sent to the device specified by the *DestinationAddress* parameter by issuing a NLDE-DATA.request primitive.

4.5.3.2 APSME-TRANSPORT-KEY.indication

The APSME-TRANSPORT-KEY.indication primitive is used to inform the ZDO of the receipt of keying material.

4.5.3.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-TRANSPORT-KEY.indication          {
                                        SrcAddress,
                                        KeyType,
                                        TransportKeyData
                                        }
```

Table 4.16 specifies the parameters of the APSME-TRANSPORT-KEY.indication primitive.

Table 4.16 APSME-TRANSPORT-KEY.indication Parameters

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
KeyType	Octet	0x00 – 0x03	Identifies the type of key material that was be transported; see Table 4.12
TransportKeyData	Variable	Variable	The key that was transported along with identification and usage parameters; the type of this parameter depends on the *KeyType* parameter as follows: *KeyType* = 0x00 see Table 4.17 *KeyType* = 0x01 see Table 4.18 *KeyType* = 0x02 see Table 4.19 *KeyType* = 0x03 see Table 4.20

Table 4.17 TransportKeyData Parameter for a Trust Center Master Key

Parameter Name	Type	Valid Range	Description
TrustCenter-MasterKey	Set of 16 octets	Variable	The trust center master key

Table 4.18 *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

4.5.3.2.2 When Generated

The APSME shall generate this primitive when it receives a transport-key command that is successfully decrypted and authenticated, as specified in sub-clause 4.5.1.2, that has the key type field set to 2 or 3 (that is, application link or master key).

Alternatively, the APSME shall generate this primitive when it receives a transport-key command that is successfully decrypted and authenticated, (as specified in sub-clause 4.5.1.2,) that has the key type field set to 0 or 1, (that is, a trust center master key or Network key) and the destination address sub-field of the key descriptor field is equal to the local address.

4.5.3.2.3 Effect on Receipt

Upon receipt of this primitive, the ZDO is informed of the receipt of the keying material.

4.5.3.3 Upon Receipt of a Transport-key Command

Upon receipt of a transport-key command, the APSME shall execute security processing as specified in sub-clause 4.5.1.2 and then check the key type sub-field.

If the key type field is set to 2 or 3 (that is, application link or master 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), the *KeyType* 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 0.

If the key type field is set to 0 or 1 (that is, trust center master key or NWK key) and the destination address field is equal to the local address, 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 KeyType 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 1), the *KeySeqNumber* sub-parameter shall be set to the sequence number field.

If the key type field is set to 0 or 1 (that is, trust center master key or NWK 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 an unsecured transport-key command, the APSME shall check the key type sub-field. If the key type field is set to 0 (that is, a trust center master key), the destination address field is equal to the local address, and the device does not have a trust center master key and address (that is, the *apsTrustCenterAddress* in the AIB), then the APSME shall issue the APSME-TRANSPORT-KEY.indication primitive. Also, if the key type field is set to 1 (that is, Network key), the destination address field is equal to the local address, and the device does not have a Network key, then the APSME shall issue the APSME-TRANSPORT-KEY.indication primitive. If an APSME-TRANSPORT-KEY.indication primitive is issued, the SrcAddress parameter shall be set to the source address field of the key-transport command, and the KeyType parameter shall be 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 1), the KeySeqNumber sub-parameter shall be set to the sequence number field.

4.5.4 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).

4.5.4.1 APSME-UPDATE-DEVICE.request

The ZDO 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).

4.5.4.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-UPDATE-DEVICE.request          {
                                     DestAddress,
                                     DeviceAddress,
                                     Status,
                                     DeviceShortAddress
                                     }
```

Table 4.19 specifies the parameters for the APSME-UPDATE-DEVICE.request primitive.

Table 4.19 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 – 0x08	Indicates the updated status of the device given by the *DeviceAddress* parameter: 0x00: device secured join 0x01: device unsecured join 0x02: device left 0x03-0x08 reserved
DeviceShortAddress	Network address	0x0000 - 0xffff	The 16-bit network address of the device whose status is being updated

4.5.4.1.2 When Generated

The ZDO (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). Effect on receipt

4.5.4.1.3 Effect on Receipt

Upon receipt of the APSME-UPDATE-DEVICE.request primitive the device shall first create an update-device command frame (see sub-clause 4.5.8.4). The device address field of this command frame shall be set to the *DeviceAddress* parameter and 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 sub-clause 4.5.1.1 and then, if security processing succeeds, sent to the device specified by the *DestAddress* parameter by issuing a NLDE-DATA.request primitive.

4.5.4.2 APSME-UPDATE-DEVICE.indication

The APSME shall issue this primitive to inform the ZDO that it received an update-device command frame.

4.5.4.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-UPDATE-DEVICE.indication          {
                                        SrcAddress,
                                        DeviceAddress,
                                        Status,
                                        DeviceShortAddress
                                        }
```

Table 4.20 specifies the parameters for the APSME-UPDATE-DEVICE.indication primitive.

Table 4.20 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	Octet	0x00 – 0xFF	Indicates the updated status of the device given by the *DeviceAddress* parameter 0x00: device secured join. 0x01: device unsecured join. 0x02: device left. 0x03-0xFF reserved.
DeviceShortAddress	Network address	0x0000 - 0xffff	The 16-bit network address of the device whose status is being updated

4.5.4.2.2 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 sub-clause 4.5.1.2.

4.5.4.2.3 Effect on Receipt

Upon receipt of the APSME-UPDATE-DEVICE.indication primitive the ZDO will be informed that the device referenced by the *DeviceAddress* parameter has undergone a status update according to the *Status* parameter.

4.5.5 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.

4.5.5.1 APSME-REMOVE-DEVICE.request

The ZDO 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 fails to authenticate properly.

4.5.5.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-REMOVE-DEVICE.request          {
                                     ParentAddress,
                                     ChildAddress
                                     }
```

Table 4.21 specifies the parameters for the APSME-REMOVE-DEVICE.request primitive.

Table 4.21 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
ChildAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the child device that is requested to be removed

4.5.5.1.2 When Generated

The ZDO (for example, on a trust) 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 *ChildAddress* parameter).

4.5.5.1.3 Effect on Receipt

Upon receipt of the APSME-REMOVE-DEVICE.request primitive the device shall first create a remove-device command frame (see sub-clause 4.5.8.4). The child address field of this command frame shall be set to the *ChildAddress* parameter. This command frame shall be security protected as specified in sub-clause 4.5.1.1 and then, if security processing succeeds, sent to the device specified by the *ParentAddress* parameter by issuing a NLDE-DATA.request primitive.

4.5.5.2 APSME-REMOVE-DEVICE.indication

The APSME shall issue this primitive to inform the ZDO that it received a remove-device command frame.

4.5.5.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-REMOVE-DEVICE.indication          {
                                        SrcAddress,
                                        ChildAddress
                                        }
```

Table 4.22 specifies the parameters for the APSME-REMOVE-DEVICE.indication primitive.

Table 4.22 APSME-REMOVE-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 requesting that a child device be removed
ChildAddress	Device Address	Any valid 64-bit address	The extended 64-bit address of the child device that is requested to be removed

4.5.5.2.2 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 sub-clause 4.5.1.2.

4.5.5.2.3 Effect on Receipt

Upon receipt of the APSME-REMOVE-DEVICE.indication primitive the ZDO shall be informed that the device referenced by the *SrcAddress* parameter is requesting that the child device referenced by the *ChildAddress* parameter be removed from the network.

4.5.6 Request Key Services

The APSME provides services that allow a device to request the current Network key or a master key from another device (for example, its trust center).

4.5.6.1 APSME-REQUEST-KEY.request

This primitive allows the ZDO to request either the current Network key or a new end-to-end application master key.

4.5.6.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-REQUEST-KEY.request        {
                                 DestAddress,
                                 KeyType,
                                 PartnerAddress
                                 }
```

Table 4.23 specifies the parameters for the APSME-REQUEST-KEY.request primitive.

Table 4.23 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
KeyType	Octet	0x00-0xFF	The type of key being requested: 0x01 = Network key 0x02 = Application key 0x00 and 0x03-0xFF = Reserved
PartnerAddress	Device Address	Any valid 64-bit address	In the case that *KeyType* 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

4.5.6.1.2 When Generated

The ZDO of a device shall generate the APSME-REQUEST-KEY.request primitive when it requires either the current Network key or a new end-to-end application master key.

4.5.6.1.3 Effect on Receipt

Upon receipt of the APSME-REQUEST-KEY.request primitive the device shall first create a request-key command frame (see sub-clause 4.5.8.6). The key type field of this command frame shall be set to the same value as the *KeyType* parameter. If the *KeyType* parameter is 0x02 (that is, an application 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 sub-clause 4.5.1.1 and then, if security processing succeeds, sent to the device specified by the *DestAddress* parameter by issuing a NLDE-DATA.request primitive.

4.5.6.2 APSME-REQUEST-KEY.indication

The APSME shall issue this primitive to inform the ZDO that it received a request-key command frame.

4.5.6.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-REQUEST-KEY.indication          {
                                      SrcAddress,
                                      KeyType,
                                      PartnerAddress
                                      }
```

Table 4.24 specifies the parameters for the APSME-REQUEST-KEY.indication primitive.

Table 4.24 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
KeyType	Octet	0x00-0xFF	The type of key being requested: 0x01 = Network key 0x02 = Application key 0x00 and 0x03-0xFF = Reserved
PartnerAddress	Device Address	Any valid 64-bit address	In the case that *KeyType* 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

4.5.6.2.2 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 sub-clause 4.5.1.2.

4.5.6.2.3 Effect on Receipt

Upon receipt of the APSME-REQUEST-KEY.indication primitive the ZDO shall be informed that the device referenced by the *SrcAddress* parameter is requesting a key. The type of key being requested shall be indicated by the *KeyType* parameter and if the *KeyType* parameter is 0x02 (that is, an application key), the *PartnerAddress* parameter shall indicate a partner device that shall receive the same key as the device requesting the key (that is, the device indicated by the *SrcAddress* parameter).

4.5.7 Switch Key Services

The APSME provides services that allow a device (for example, a trust center) to inform another device that it should switch to a new active Network key.

4.5.7.1 APSME-SWITCH-KEY.request

This primitive allows a device (for example, the trust center) to request that another device switch to a new active Network key.

4.5.7.1.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-SWITCH-KEY.request                {
                                        DestAddress,
                                        KeySeqNumber
                                        }
```

Table 4.25 specifies the parameters for the APSME-SWITCH-KEY.request primitive.

Table 4.25 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
KeySeqNumber	Octet	0x00-0xFF	A sequence number assigned to a Network key by the trust center and used to distinguish Network keys

4.5.7.1.2 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 to switch to a new active Network key.

4.5.7.1.3 Effect on Receipt

Upon receipt of the APSME-SWITCH-KEY.request primitive the device shall first create a switch-key command frame (see sub-clause 4.5.8.7). The sequence number field of this command frame shall be set to the same value as the *KeySeqNumber* parameter.

This command frame shall be security protected as specified in sub-clause 4.5.1.1 and then, if security processing succeeds, sent to the device specified by the *DestAddress* parameter by issuing a NLDE-DATA.request primitive.

4.5.7.2 APSME-SWITCH-KEY.indication

The APSME shall issue this primitive to inform the ZDO that it received a switch-key command frame.

4.5.7.2.1 Semantics of the Service Primitive

This primitive shall provide the following interface:

```
APSME-SWITCH-KEY.indication          {
                                     SrcAddress,
                                     KeySeqNumber
                                     }
```

Table 4.26 specifies the parameters for the APSME-SWITCH-KEY.indication primitive.

Table 4.26 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

4.5.7.2.2 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 sub-clause 4.5.1.2.

4.5.7.2.3 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.

4.5.7.3 Secured APDU Frame

The APS layer frame format consists of APS header and APS payload fields (see Figure 2.4.) 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 sub-clause 4.6.1. 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.

Octets: Variable	**5 or 6**	**Variable**	
Original APS Header ([B7], Clause 7.1)	Auxiliary frame header	Encrypted Payload	Encrypted Message Integrity Code (MIC)
		Secure frame payload = Output of CCM*	
Full APS header		Secured APS payload	

Figure 4.6 Secured APS Layer Frame Format

4.5.8 Command Frames

The APS layer command frame formats are given in this clause.

Command identifier values are shown in Table 4.27.

Table 4.27 Command Identifier Values

Command Identifier	Value
APS_CMD_SKKE_1	0X01
APS_CMD_SKKE_2	0X02
APS_CMD_SKKE_3	0X02
APS_CMD_SKKE_4	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

4.5.8.1 Key-establishment Commands

The APS command frames used during key establishment is specified in this clause. The optional fields of the APS header portion of the general APS frame format shall not be present.

The generic SKKE command frame shall be formatted as illustrated in Figure 4.7.

Octets: 1	**1**	**8**	**8**	**16**
Frame control	Command identifier	Initiator Address	Responder Address	Data
APS Header	Payload			

Figure 4.7 Generic SKKE Frame Command Format

4.5.8.1.1 Command Identifier Field

The command identifier field shall indicate the APS command type. For SKKE frames, the command identifier shall indicate either an SKKE-1, SKKE-2, SKKE-3, or SKKE-4 frame, depending on the frame type (see Table 4.27).

4.5.8.1.2 Initiator Address Field

The initiator address field shall be the 64-bit extended address of the device that acts as the initiator in the key-establishment protocol.

4.5.8.1.3 Responder Address Field

The responder address field shall be the 64-bit extended address of the device that acts as the responder in the key-establishment protocol.

4.5.8.1.4 Data Field

The content of the data field depends on the command identifier field (that is, SKKE-1, SKKE-2, SKKE-3, or SKKE-4). Clauses 4.5.8.1.4.1 through 4.5.8.1.4.4 describe the content of the data field for each command type.

4.5.8.1.4.1 SKKE-1 Frame

The data field shall be the octet representation of the challenge *QEU* generated by the initiator during action step 1 of sub-clause B.7.1.

4.5.8.1.4.2 SKKE-2 Frame

The data field shall be the octet representation of the challenge *QEV* generated by the responder during action step 2 of sub-clause B.7.2.

4.5.8.1.4.3 SKKE-3 Frame

The data field shall be the octet representation of the string $MacTag_2$ generated by the initiator during action step 1 of sub-clause B.7.1.

4.5.8.1.4.4 SKKE-4 Frame

The data field shall be the octet representation of the string $MacTag_1$ generated by the responder during action step 10 of sub-clause B.7.2.

4.5.8.2 Transport-key Commands

The transport-key command frame shall be formatted as illustrated in Figure 4.8. The optional fields of the APS header portion of the general APS frame format shall not be present.

Octets: 1	**1**	**1**	**Variable**
Frame control	APS command identifier	Key Type	Key descriptor
APS Header	Payload		

Figure 4.8 Transport-Key Command Frame

4.5.8.3 Command Identifier Field

This field is 8-bits in length shall be set to indicate that this is a transport-key command frame (see Table 4.27).

4.5.8.3.1 Key Type 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 4.12.

4.5.8.3.2 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 key type field – see sub-clause 4.5.8.3.1) and shall be set to one of the formats described in the following subsections.

4.5.8.3.2.1 Trust Center Master Key Descriptor Field

If the key type field is set to 0, the key descriptor field shall be formatted as shown in Figure 4.9.

Octets: 16	**8**	**8**
Key	Destination address	Source address

Figure 4.9 Trust Center Master Key Descriptor Field in Transport-Key Command

- The key sub-field shall contain the master key that should be used to set up link keys with the trust center;
- The destination address sub-field shall contain the address of the device which should use this master key;
- The source address sub-field shall contain the address of the device (for example, the trust center) which originally sent this master key.

4.5.8.3.2.2 Network Key Descriptor Field

If the key type field is set to 1, this field shall be formatted as shown in Figure 4.10.

Octets: 16	**1**	**8**	**8**
Key	Sequence number	Destination address	Source address

Figure 4.10 Network Key Descriptor Field in Transport-Key Command

- 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 sub-field 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.

4.5.8.3.2.3 Application Master and Link Key Descriptor Field

If the key type field is set to 2 or 3, this field shall be formatted as shown in Figure 4.11.

Octets: 16	**8**	**1**
Key	Partner address	Initiator flag

Figure 4.11 Application Master Key Descriptor in Transport-Key Command

- The key sub-field shall contain a master or link key that is shared with the device identified in the partner address field;
- The partner address sub-field shall contain the address of the other device that was sent this link or master 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.

4.5.8.4 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 4.12.

Octets: 1	**1**	**8**	**2**	**1**
Frame control	Command identifier	Device Address	Device short address	Status
APS Header	Payload			

Figure 4.12 Update-Device Command Frame Format

4.5.8.4.1 Command Identifier Field

The command identifier field shall indicate the APS command type update-device (see Tables 4.27).

4.5.8.4.2 Device Address Field

The device address field shall be the 64-bit extended address of the device whose status is being updated.

4.5.8.4.3 Device Short Address Field

The device short address field shall be the 16-bit network address of the device whose status is being updated.

4.5.8.4.4 Status Field

The status field shall be assigned a value as described for the Status parameter in Table 4.19.

4.5.8.5 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 4.13.

Octets: 1	**1**	**8**
Frame control	Command identifier	Child address
APS Header	Payload	

Figure 4.13 Remove-Device Command Frame Format

4.5.8.5.1 Command Identifier Field

The command identifier field shall indicate the APS command type remove-device (see Table 4.27).

4.5.8.5.2 Child Address Field

The child address field shall be the 64-bit extended address of the device that is requested to be removed from the network.

4.5.8.6 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 4.14

Octets: 1	**1**	**1**	**0/8**
Frame control	Command identifier	Key type	Partner address
APS Header	Payload		

Figure 4.14 Request-Key Command Frame Format

4.5.8.6.1 Command Identifier Field

The command identifier field shall indicate the APS command type request-key (see Table 4.27).

4.5.8.6.2 Key Type Field

The key type field shall be set to 1 when the Network key is being requested and shall be set to 2 when an application key is being requested.

4.5.8.6.3 Partner Address Field

When the key type 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 key-type field is 1 (that is, a Network key), the partner address field will not be present.

4.5.8.7 Switch-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 switch-key command frame shall be formatted as illustrated in Figure 4.15.

Octets: 1	**1**	**1**
Frame control	Command identifier	Sequence number
APS Header	Payload	

Figure 4.15 Switch-key Command Frame Format

4.5.8.7.1 Command Identifier Field

The command identifier field shall indicate the APS command type switch-key (see Table 4.27).

4.5.8.7.2 Sequence Number Field

The sequence number field shall contain the sequence number identifying the Network key to make active.

4.5.9 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 Tables 4.28 and 4.29.

Table 4.28 AIB Security Attributes

Attribute	Identifier	Type	Range	Description	Default
apsDeviceKeyPairSet	0xaa	Set of Key-Pair Descriptor entries. See Table 4.29	Variable	A set of key-pair descriptors containing master and link key pairs shared with other devices	-
apsTrustCenterAddress	0xab	Device address	Any valid 64-bit address	Identifies the address of the device's trust center	-
apsSecurityTimeOutPeriod	0xac	Integer	0x0000-0xFFFF	The period of time a device will wait for an expected security protocol frame (in milliseconds)	1000

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	-
MasterKey	Set of 16 octets	-	The actual value of the master key	-
LinkKey	Set of 16 octets	-	The actual value of the link key	-
OutgoingFrame-Counter	Set of 4 octets	0x00000000-0xFFFFFFFF	Unique identifier of the key originating with the device indicated by *KeySrcAddress*	0x00000000
IncomingFrame-Counter	Set of 4 octets	0x00000000-0xFFFFFFFF	Incoming frame counter value corresponding to *DeviceAddress*	0x00000000

4.6 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 sub-clause 4.6.1. The MAC, NWK, and APS layers shall use the security parameters specified in sub-clause 4.6.2. 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 (left-most and least significant) to k-1 (right-most 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.

4.6.1 Auxiliary Frame Header Format

The auxiliary frame header, as illustrated by Figure 4.16, shall include a security control field and a frame counter field, and may include a sender address field and key sequence number field.

Octets: 1	4	0/8	0/1
Security control	Frame Counter	Source Address	Key Sequence Number

Figure 4.16 Auxiliary Frame Header Format

4.6.1.1 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 4.17.

Bit: 0-2	3-4	5	6-7
Security level	Key identifier	Extended Nonce	Reserved

Figure 4.17 Security Control Field Format

4.6.1.1.1 Security Level Sub-field

The security level identifier indicates how an outgoing frame is to be secured, respectively, how an incoming frame purportedly has been secured: it 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 4.30. 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 MAC, NWK, and APS layers

Security Level Identifier	**Security Level Sub-field** (Table 4.17)	**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)

Table 4.30 Security Levels Available to the MAC, NWK, and APS layers

Security Level Identifier	Security Level Sub-field (Table 4.17)	Security Attributes	Data Encryption	Frame Integrity (length M of MIC, in Number of Octets)
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)

4.6.1.1.2 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 4.31.

Table 4.31 Encoding for the Key Identifier Sub-field

Key Identifier	Key Identifier Sub-field (Table 4.17)	Description
0x00	'00'	A link key
0x01	'01'	A Network key
0x02	'10'	A key-transport key
0x03	'11'	A key-load key

4.6.1.1.3 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.

4.6.1.2 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.

4.6.1.3 Counter Field

The counter field is used to provide for frame freshness and to prevent processing of duplicate frames.

4.6.1.4 Key Sequence Number Field

The key sequence number field shall only be present when the key identifier sub-field of the security control field is 1 (that is, the Network key). When present, the key sequence number field shall indicate the key sequence number of the Network key used to secure the frame.

4.6.2 Security Parameters

This sub-clause specifies the parameters used for the CCM* security operations.

4.6.2.1 CCM* Mode of Operation and Parameters

Applying security to a MAC, NWK, or APS frame on a particular security level corresponds to a particular instantiation of the AES-CCM* mode of operation as specified in sub-clause B.1.2. The AES-CCM* mode of operation is an extension of the AES-CCM mode that is used in the 802.15.4-2003 MAC specification and provides capabilities for authentication, encryption, or both.

The nonce shall be formatted as specified in sub-clause 4.6.2.2.

Table 4.30 gives the relationship between the security level subfield of the security control field (Table 4.17), the security level identifier, and the CCM* encryption/authentication properties used for these operations.

4.6.2.2 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 4.18 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 sub-clause 4.6.1) of the frame being processed. The nonce's source address field shall be set to the extended 64-bit MAC 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 MAC address of the device originating security protection of the frame shall correspond to the auxiliary header's source address field (as defined in sub-clause 4.6.1) of the frame being processed

Octets: 8	**4**	**1**
Source address	Frame counter	Security control

Figure 4.18 CCM* Nonce

4.6.3 Cryptographic Key Hierarchy

The link key established between two (or more) devices via one of the key-establishment schemes specified in sub-clause 4.5.2 (or transport-key commands specified in sub-clause 4.5.3) 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 sub-clause B.1.4 under the link key with as input string the 1-octet string ‘0x00’.
2. *Key-Load Key.* This key is the outcome of executing the specialized keyed hash function specified in sub-clause B.1.4 under the link key with as input string the 1-octet string ‘0x02’.
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 Network key and associated outgoing and incoming frame counters.

4.6.4 Implementation Guidelines (Informative)

This clause provides general guidelines that should be followed to ensure a secure implementation.

4.6.4.1 Random Number Generator

A ZigBee device implementing the key-establishment (that is, see sub-clause 4.2.4.1) security service may need 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. The general recommendation is that the random number generation shall meet the random number tests specified in FIPS140-2 [B12]. 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 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.

4.6.4.2 Security Implementation

To avoid "bugs" that an attacker can use to his advantage, it is crucial that security be well implemented and tested. It is also desirable that the security implementation does not need to be re-certified for every application. Security services should be implemented and tested by security experts and should not be re-implemented or modified for different applications.

4.6.4.3 Conformance

Conformance shall be defined by the profile inheriting from this specification. Correct implementation of selected cryptographic protocols should be verified as part of the ZigBee certification process. This verification shall include known value tests: an implementation must show that given particular parameters, it will correctly compute the corresponding results.

4.7 Functional Description

This sub-clause provides detailed descriptions of how the security services shall be used in a ZigBee network. A description of the ZigBee coordinator's security initialization responsibilities is given in sub-clause 4.7.1. A brief description of the trust center application is given in sub-clause 4.7.2. Detailed security procedures are given in sub-clause 4.7.3.

4.7.1 ZigBee Coordinator

The ZigBee coordinator shall configure the security level of the network by setting the *NwkSecurityLevel* attribute in the NWK layer PIB table. If the *NwkSecurityLevel* attribute is set to zero, the network will be unsecured, otherwise it will be secured.

The ZigBee coordinator shall configure the address of the trust center by setting the AIB attribute *apsTrustCenterAddress*. The default value of this address is the ZigBee coordinator's address itself, otherwise, the ZigBee coordinator may designate an alternate trust center.

4.7.2 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 be configured to operate in either commercial or residential mode and may be used to help establish end-to-end application keys either by sending out link keys directly (that is, key-escrow capability) or by sending out master keys. These keys shall be generated at random.

4.7.2.1 Commercial Mode

The commercial mode of the trust center is designed for high-security commercial applications. In this mode, the trust center shall maintain a list of devices, master keys, link keys, and Network keys that it needs to control and enforce the policies of Network key updates and network admittance. In this mode, the memory required for the trust center grows with the number of devices in the network and the *nwkAllFresh* attribute in the NIB shall be set to TRUE.

4.7.2.2 Residential Mode

The residential mode of the trust center is designed for low-security residential applications. In this mode, the trust center may maintain a list of devices, master keys, or link keys with all the devices in the network; however, it shall maintain the Network key and controls policies of network admittance. In this mode, the memory required for the trust center does not grow with the number of devices in the network and the *nwkAllFresh* attribute in the NIB shall be set to FALSE.

4.7.3 Security Procedures

This sub-clause 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.

4.7.3.1 Joining a Secured Network

Figure 4.19 shows an example message sequence chart ensuing from when a joiner device communicates with a router device to join a secured network.

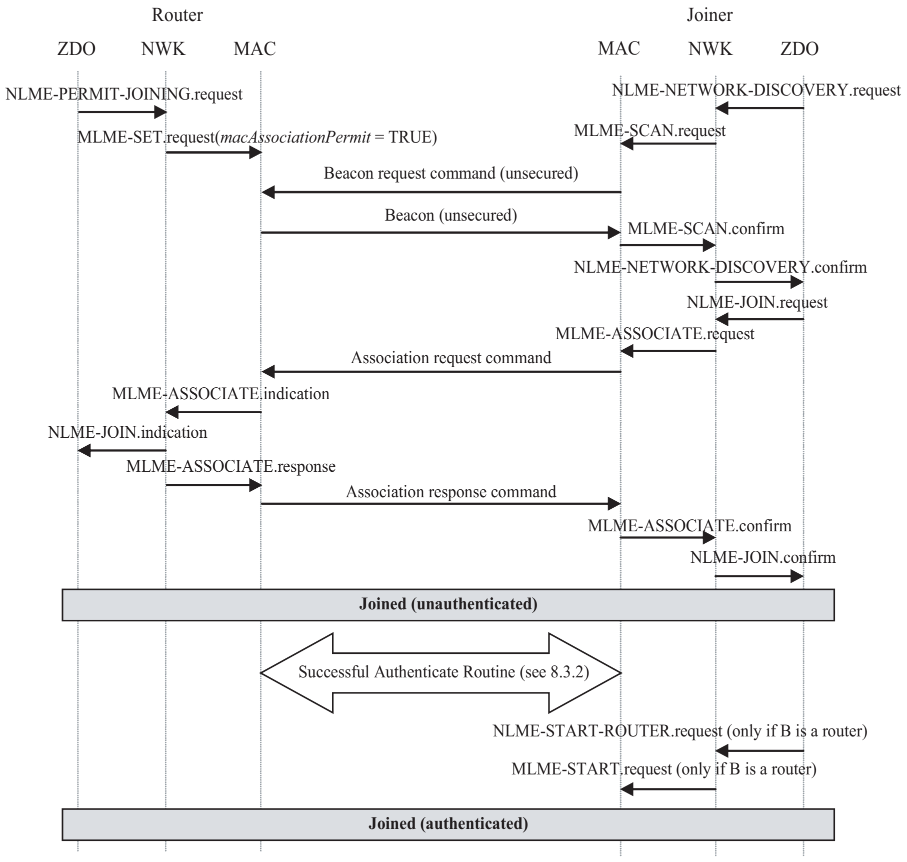

Router
Joiner
ZDO
NWK
MAC
MAC
NWK
ZDO
NLME-PERMIT-JOINING.request
NLME-NETWORK-DISCOVERY.request
MLME-SET.request(*macAssociationPermit* = TRUE)
MLME-SCAN.request
Beacon request command (unsecured)
Beacon (unsecured)
MLME-SCAN.confirm
NLME-NETWORK-DISCOVERY.confirm
NLME-JOIN.request
MLME-ASSOCIATE.request
Association request command
MLME-ASSOCIATE.indication
NLME-JOIN.indication
MLME-ASSOCIATE.response
Association response command
MLME-ASSOCIATE.confirm
NLME-JOIN.confirm
Joined (unauthenticated)
Successful Authenticate Routine (see 8.3.2)
NLME-START-ROUTER.request (only if B is a router)
MLME-START.request (only if B is a router)
Joined (authenticated)

Figure 4.19 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 will issue an NLME-NETWORK-DISCOVERY.confirm primitive. The *NetworkList* parameter of this primitive will indicate all of the nearby PANs along with their *nwkSecurityLevel* and *nwkSecureAllFrames* attributes. In Figure 4.19, 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 (for example, based on the security attributes received in NLME-NETWORK-DISCOVERY.confirm primitive) 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 4.19, the NLME-JOIN.request primitive causes an association request command to be sent to the router.

Upon receipt of the association request command, the router shall issue an MLME-ASSOCIATE.indication primitive with the *SecurityUse* parameter set to TRUE or FALSE, depending on whether or not the association request command was secured. 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 whether the Network key was used to secure the association request command. The router will also issue an MLME-ASSOCIATE.response primitive with the *SecurityEnable* parameter set to TRUE or FALSE, depending on whether or not the association request command was secured or not, respectively. This primitive will cause an association response command to be sent to the joiner.

Upon receipt of the association response command, the joiner shall issue the NLME-JOIN.confirm primitive The joiner is now declared "joined, but unauthenticated" to the network. The authentication routine (see sub-clause 4.7.3.2) shall follow.

If the joiner is not a router, it is declared "joined and authenticated" immediately following the successful completion of the authentication routine.

If the joiner is a router, it is declared "joined and authenticated" only after the successful completion of the authentication 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 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 authentication routine.

4.7.3.2 Authentication

Once a device joins a secured network and is declared "joined but unauthenticated", it must be authenticated as specified in this sub-clause.

4.7.3.2.1 Router Operation

If the router is not the trust center, it shall begin the authentication 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 to 0x00 (that is, secured join) if the newly joined device secured the associate request command. Otherwise, the *Status* parameter shall be set to 0x01 (that is, unsecured join).

If the router is the trust center, it shall begin the authentication procedure by simply operating as a trust center.

4.7.3.2.2 Trust Center Operation

The trust center role in the authentication 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 at least five factors:

- Whether the trust center decides to allow the new device to join the network (for example, the trust center is in a mode that allows new devices to join);
- Whether the trust center is operating in residential or commercial mode (see sub-clause 4.7.2.1 and sub-clause 4.7.2.2, respectively);
- If in residential mode, whether the device is joining unsecured or secured (that is, as indicated by the *Status* sub-field of the update-device command);
- If in commercial mode, whether the trust center has a master key corresponding to the newly joined device;
- The *nwkSecureAllFrames* parameter of the NIB.

If, at any time during the authentication procedure, the trust center decides not to allow the new device to join the network (for example, a policy decision or a failed 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 shall 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 unauthenticated) device. If the trust center is the router of the newly joined device, it shall remove the device from the network by issuing the NLME-LEAVE.request primitive with the DeviceAddress parameter set to the address of the joined (but unauthenticated) device, the Rejoin parameter set to 0, the SilentLeave parameter set to 1 and the ReuseAddress parameter set to 1. Other fields are defined by the stack profile.[19]

4.7.3.2.2.1 Residential Mode

After being activated for the authentication procedure the trust center 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 *KeyType* parameter to 0x01 (that is, Network key).

19. CCB #676

If the joining device already has the Network key (that is, the *Status* sub-field of the update-device command is 0x00), the *TransportKeyData* sub-parameters shall be set as follows: the *KeySeqNumber* sub-parameter shall be set to 0, the *NetworkKey* sub-parameter shall be set to all zeros, and the *UseParent* sub-parameter shall be set to FALSE.

Otherwise, the *KeySeqNumber* sub-parameter shall be set to the sequence count value for this Network key, the *NetworkKey* sub-parameter shall be set to 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.

In the case of a joining device that is not pre-configured with a Network key, the issuance of this transport-key primitive will cause the Network key to be sent unsecured from the router to the newly joined device — security is assumed to be present here via non-cryptographic means — such as only sending this key once, at low power, immediately after external input to both router and joiner, that can guarantee secrecy and authenticity. If the joining device did not receive the key within *apsSecurityTimeOutPeriod*, it shall reset and may choose to start the joining procedure again.

4.7.3.2.2.2 Commercial Mode

After being activated for the authentication procedure, the trust center operation in commercial mode depends on if the device joining the network is preconfigured with a trust center master key.

If the trust center does not already share a master key with the newly joined device, it shall send the device a master key by issuing the APSME-TRANSPORT-KEY.request primitive with the *DestAddress* parameter set to the address of the newly joined device, and the *KeyType* parameter to 0x00 (that is, trust center master key). The *TransportKeyData* sub-parameters shall be set as follows: the *TrustCenterMasterKey* sub-parameter shall be set to trust center master key, and the *ParentAddress* sub-parameter shall set to the address of the local device if the trust center is the router. Otherwise, the *ParentAddress* sub-parameter shall set to the address of the router originating the update-device command. The issuance of this primitive will cause the master key to be sent unsecured from the router to the newly joined device—security is assumed to be present here via non-cryptographic means, such as only sending this key once, at low power, immediately after external input to both router and joiner, etc.

The trust center shall initiate the establishment of a link key by issuing the APSME-ESTABLISH-KEY.request primitive with the *ResponderAddress* parameter set to the address of the newly joined device and the *KeyEstablishmentMethod* set to 0x00 (that is, SKKE). Additionally, if the *nwkSecureAllFrames* parameter of the NIB is FALSE or the trust center is the router, the *UseParent* parameter shall be set to FALSE. Otherwise, the *UseParent* parameter shall be set to TRUE and the *ResponderParentAddress* parameter shall be set to the address of the router originating the update-device command.

Upon receipt of the corresponding APSME-ESTABLISH-KEY.confirm primitive with *Status* equal to 0x00 (that is, success), the trust center shall send the new device the 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 *KeyType* parameter to 0x01 (that is, Network key). The *TransportKeyData* sub-parameters shall be set as follows:

- The *KeySeqNumber* sub-parameter shall be set to the sequence count value for this Network key;
- The *NetworkKey* sub-parameter shall be set to Network key;
- The *UseParent* sub-parameter shall be set to FALSE.

4.7.3.2.3 Joining Device Operation

After successfully associating to a secured network, the joining device shall participate in the authentication procedure described in this sub-clause. Following a successful authentication procedure, the joining device shall set the *nwkSecurityLevel* and *nwkSecureAllFrames* attributes in the NIB to the values indicated in the beacon from the router.

A joined and authenticated device in a secured network with *nwkSecureAllFrames* equal to TRUE shall always apply NWK layer security to outgoing (incoming) frames unless the frame is destined for (originated from) a newly joined but unauthenticated child. No such restrictions exist if *nwkSecureAllFrames* is equal to FALSE.

The joining device's participation in the authentication procedure depends on the state of the device. There are three possible initial states to consider:

- Preconfigured with a Network key (that is, residential mode)
- Preconfigured with a trust center master key and address (that is, commercial mode)
- Not preconfigured (that is, undetermined mode – either residential or commercial mode)

In a secured network, if the device does not become authenticated within a preconfigured amount of time, it shall leave the network.

4.7.3.2.3.1 Preconfigured Network Key

If the joining device was preconfigured with just a Network key (and the association was successful), it shall set the outgoing frame counter for this key to zero, and empty the incoming frame counter set for this key, and wait to receive a dummy (all zero) Network key from the trust center. Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x01 (that is, the 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 authenticated and shall enter the normal operating state for residential mode.

4.7.3.2.3.2 Preconfigured Trust Center Key

If the joining device is preconfigured with a trust center master key and address (that is, the *apsTrustCenterAddress* attribute in the AIB) it shall wait to establish a link key and receive a Network key from the trust center. Therefore, upon receipt of the APSME-ESTABLISH-KEY.indication primitive with the *InitiatorAddress* parameter set to the trust center's address and the *KeyEstablishmentMethod* parameter set to SKKE, the joining device shall respond with the APSME-ESTABLISH-KEY.response primitive with the *InitiatorAddress* parameter set to the trust center's address and the *Accept* parameter set to TRUE. After receipt of the APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set to the trust center's address and the *Status* parameter set to 0x00 (that is, success), the joining device shall expect to receive the Network key. Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with SourceAddress parameter set to the trust center's address and with the KeyType parameter set to 0x01 (that is, the Network key), the joining device shall use the data in the TransportKeyData parameter for configuring the Network key. All incoming frame counters and the outgoing frame counter of the Network key shall be set to 0.[20] The joining device is now considered authenticated and shall enter the normal operating state for commercial mode. If the joining device did not receive the APSME-ESTABLISH-KEY.indication primitive with the *InitiatorAddress* parameter set to the trust center's address and the *KeyEstablishmentMethod* parameter set to SKKE within *apsSecurityTimeOutPeriod*, it shall reset and may choose to start the joining procedure again.

4.7.3.2.3.3 Not Preconfigured

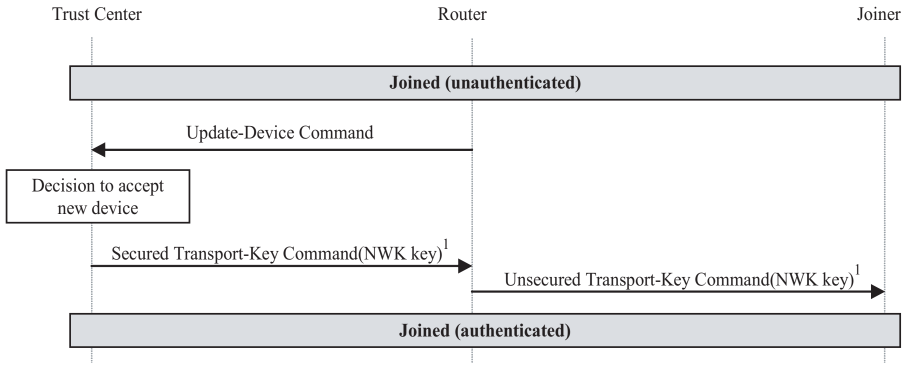
If the joining device is not preconfigured with a Network key nor a trust center master key and address (that is, the *apsTrustCenterAddress* attribute in the AIB) it shall wait to receive either an unsecured trust center master key or a Network key. Implementers should note that transmission of an unsecured key represents a security risk and that if security is a concern, keys should be preconfigured – preferable via an out-of-band mechanism that guarantees secrecy and authenticity. If the joining device did not receive either of the keys within *apsSecurityTimeOutPeriod*, it shall reset and may choose to start the joining procedure again.

20. CCB #671

Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x01 (that is, the Network key), the joining device shall make the data in the *TransportKeyData* parameter its active Network key and 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 authenticated and shall enter the normal operating state for residential mode.

Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x00 (that is, the trust center master key), the joining device shall make its trust center master key the data in the *TransportKeyData* parameter and the *apsTrustCenterAddress* attribute in its AIB the *SrcAddress* parameter. Next, upon receipt of the APSME-ESTABLISH-KEY.indication primitive with the *InitiatorAddress* parameter set to the trust center's address and the *KeyEstablishmentMethod* parameter set to SKKE, the joining device shall respond with the APSME-ESTABLISH-KEY.response primitive with the *InitiatorAddress* parameter set to the trust center's address and the *Accept* parameter set to TRUE. After receipt of the APSME-ESTABLISH-KEY.confirm primitive with the *Address* parameter set to the trust center's address and the *Status* parameter set to 0x00 (that is, success), the joining device shall expect to receive the Network key. Upon receipt of the APSME-TRANSPORT-KEY.indication primitive with SourceAddress parameter set to the trust center's address and with the KeyType parameter set to 0x01 (that is, the Network key), the joining device shall use the data in the TransportKeyData parameter for configuring the Network key. All incoming frame counters and the outgoing frame counter of the Network key shall be set to 0.[21] The joining device is now considered authenticated and shall enter the normal operating state for commercial mode.

21. CCB #671


Trust Center
Router
Joiner
Joined (unauthenticated)
Update-Device Command
Decision to accept new device
Secured Transport-Key Command(NWK key)1
Unsecured Transport-Key Command(NWK key)1
Joined (authenticated)

Note:
1. The trust center sends a dummy all-zero NWK key if the joiner securely joined using a preconfigured network key.

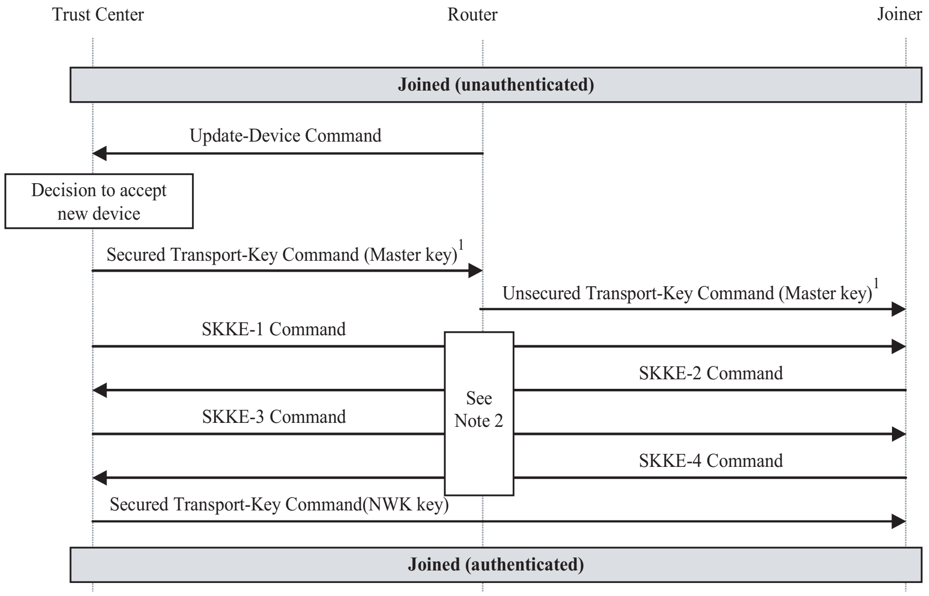
Figure 4.20 Example Residential-Mode Authentication Procedure

4.7.3.2.4 Message Sequence Charts

Figure 4.20 and 4.21 give example message sequence charts for the authentication procedure when the router and trust center are separate devices operating in residential or commercial mode, respectively.

In Figure 4.20 the update-device and transport-key commands communicated between the trust center and the router shall be secured at the APS layer based on the Network key; If the *nwkSecureAllFrames* NIB attribute is TRUE, it is also secured at the NWK layer with the Network key. The transport-key command sent from the router to joiner shall not be secured.

In Figure 4.21, the update-device and transport-key commands communicated between the trust center, and the router shall be secured at the APS layer based on the trust center link key; If the *nwkSecureAllFrames* NIB attribute is TRUE, it is also secured at the NWK layer with the Network key. The transport-key command sent from the router to joiner shall not be secured. The SKKE commands shall be sent using the router as a liaison when the *nwkSecureAllFrames* NIB attribute is TRUE, such that SKKE commands between the trust center and router shall be secured at the NWK layer with the Network key and commands between the router and joiner shall not be secured. Otherwise, the SKKE commands shall be unsecured between the trust center and joiner. The final transport-key communicated between the trust center and the joiner shall be secured at the APS layer based on the trust center link key and, if the *nwkSecureAllFrames* NIB attribute is TRUE, also secured at the NWK layer with the Network key.



Notes:
1. The trust center does not send a master key if it already shares one with the joiner device (i.e., the pre-configured situation)
2. SKKE commands shall be sent using the router as a liaison when the *nwkSecureAllFrame* NIB attribute is TRUE (i.e., these commands will be secured between the trust center and router at the NWK layer, but not between the router and joiner).

Figure 4.21 Example Commercial-Mode Authentication Procedure

4.7.3.3 Network Key Update

The trust center and network device shall follow the procedures described in this sub-clause when updating the Network key.

4.7.3.3.1 Trust Center Operation

When operating in residential mode, the trust center may update the Network key for all devices on the network. To update the Network key, the trust center shall first send the new Network key by issuing the APSME-TRANSPORT-KEY.request primitive with the DestAddress parameter set to the broadcast address and the KeyType parameter set to 0x01 (that is, Network key). The TransportKeyData sub-parameters shall be set as follows:

- The *KeySeqNumber* sub-parameter shall be set to the sequence count value for this Network key;
- The *NetworkKey* sub-parameter shall be set to 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 with 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.

When operating in commercial mode, the trust center shall maintain a list of all devices in the network. To update the Network key, the trust center shall first send the new Network key to each device on this list and then ask each device to switch to this new key. The new Network key shall be sent to a device on the list by issuing the APSME-TRANSPORT-KEY.request primitive with the *DestAddress* parameter set to the address of the device on the list and the *KeyType* parameter set to 0x01 (that is, Network key). The *TransportKeyData* sub-parameters shall be set as follows:

The *KeySeqNumber* sub-parameter shall be set to the sequence count value for this Network key;

The *NetworkKey* sub-parameter shall be set to the 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 shall ask a device to switch to this new key by issuing the APSME-SWITCH-KEY.request primitive with the *DestAddress* parameter set to the address of the device on the list and the *KeySeqNumber* parameter set to the sequence count value for the updated Network key.

4.7.3.3.2 Network Device Operation

When in the normal operating state and upon receipt of a APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x01 (that is, 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 and if the device is capable of storing an alternate Network key, 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 Network key shall be set to 0.[22]

When in the normal operating state and upon receipt of a APSME-SWITCH-KEY.indication primitive, a device shall switch its active Network key to the one

22. CCB #671

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).

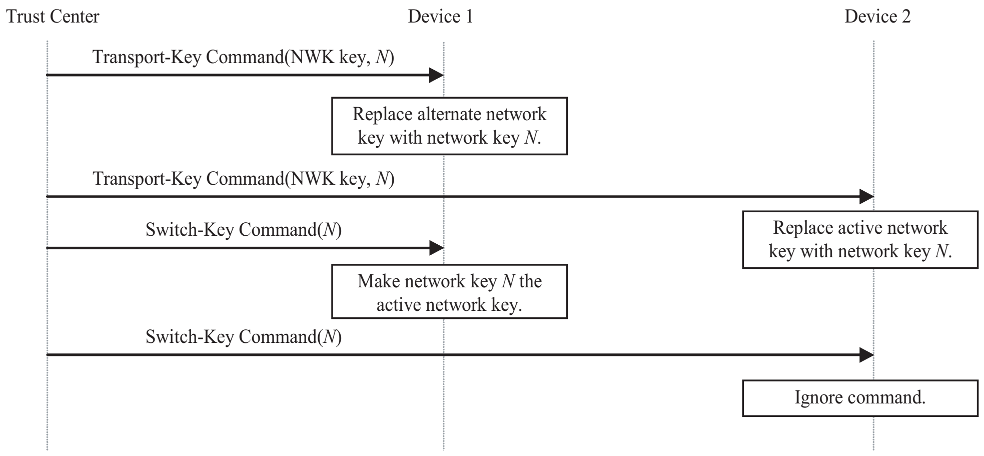

Trust Center
Device 1
Device 2
Transport-Key Command(NWK key, *N*)
Replace alternate network key with network key *N*.
Transport-Key Command(NWK key, *N*)
Replace active network key with network key *N*.
Switch-Key Command(*N*)
Make network key *N* the active network key.
Switch-Key Command(*N*)
Ignore command.

Figure 4.22 Example Network Key-Update Procedure

4.7.3.3.3 Message Sequence Chart

An example of a successful Network key-update procedure for two devices is shown in Figure 4.22. In this example, the trust center sends the Network key with sequence number *N* to devices 1 and 2. In this example, device 1 is an FD capable of storing two Network keys, an active and alternate, and device 2 is an RFD that can store only a single Network key. Upon receipt of the transport-key command, device 1 replaces its alternate Network key with the new Network key; however device 2 must replace its active Network key with the new key. Next, upon receipt of the switch-key command, device 1 makes the new Network key the active Network key; however device 2 has just one active Network key, so it ignores this command.

4.7.3.4 Network Key Recovery

A network device and trust center shall follow the procedures described in this sub-clause when recovering the Network key.

4.7.3.4.1 Network Device Operation

When in the normal operating state a network device shall request the current Network key by issuing the APSME-REQUEST-KEY.request primitive with the *DestAddress* parameter set to the trust center's address (as maintained in the *apsTrustCenterAddress* attribute of the AIB), the *KeyType* parameter set to 0x01 (that is, Network key), and the *PartnerAddress* parameter set to 0.

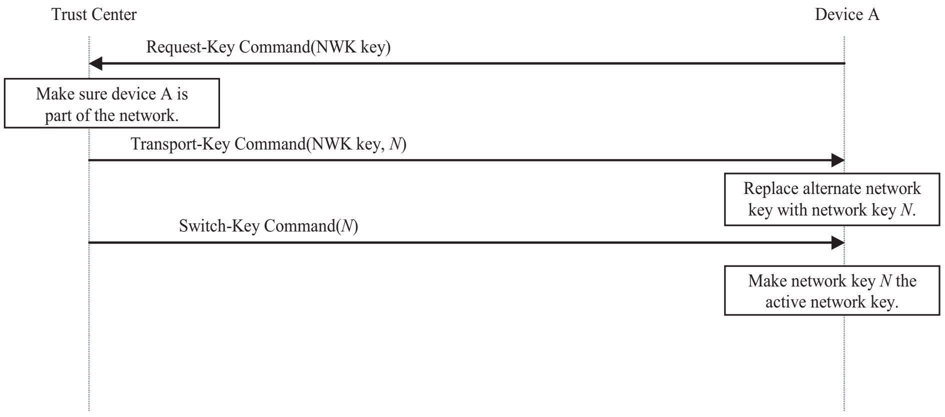

Trust Center
Device A
Request-Key Command(NWK key)
Make sure device A is part of the network.
Transport-Key Command(NWK key, *N*)
Replace alternate network key with network key *N*.
Switch-Key Command(*N*)
Make network key *N* the active network key.

Figure 4.23 Example Network Key-Recovery Procedure

4.7.3.4.2 Trust Center Operation

When operating in commercial mode and receipt of APSME-REQUEST-KEY.indication primitives with the *KeyType* parameter set to 0x01 (that is, Network key), the trust center shall determine whether the device indicated by the *SrcAddress* parameter is present on its list of all devices on the network. When operating in residential mode and receipt of APSME-REQUEST-KEY.indication primitive with the KeyType parameter set to 0x01 (that is, Network key), the trust center shall assume the device indicated by the SrcAdress parameter to be present on the list of devices on the network. If the device is present on this list, the trust center shall issue the APSME-TRANSPORT-KEY.request primitive with the *DestAddress* parameter set to the address of the device requesting the key and the *KeyType* parameter set to 0x01 (that is, Network key). The *TransportKeyData* sub-parameters shall be set as follows. The *KeySeqNumber* sub-parameter shall be set to the sequence count value for this Network key, the *NetworkKey* sub-parameter shall be set to the Network key, and the *UseParent* sub-parameter shall be set to FALSE. Next, the trust center shall ask a device to switch to this new key by issuing the APSME-SWITCH-KEY.request primitive with the *DestAddress* parameter set to the address of the device that requested the key and the *KeySeqNumber* parameter set to the sequence count value for the updated Network key.

4.7.3.4.3 Message Sequence Chart

An example of a successful Network key-recovery procedure is shown in Figure 4.23. In this example, the network device requests the current Network key from the trust center. The trust center responds with current key and then tells the device to switch to this key.

4.7.3.5 End-to-End Application Key Establishment

An initiator device, a trust center, and a responder device shall follow the procedures described in this sub-clause when establishing a link key for purposes of end-to-end application security between initiator and responder devices.

4.7.3.5.1 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 *DstDevice* parameter shall be set to the address of its trust center, the *KeyType* parameter shall be set to 0x02 (that is, application key), and the *PartnerAddress* parameter shall be set to the address of the responder device.

4.7.3.5.1.1 Upon Receipt of a Link Key

Upon receipt of an APSME-TRANSPORT-KEY.indication primitive with the *KeyType* 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 *DeviceKeyPairSet* 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.

4.7.3.5.1.2 Upon Receipt of a Master Key

Upon receipt of an APSME-TRANSPORT-KEY.indication primitive with the *KeyType* parameter set to 0x02 (that is, application master key), a device may accept the *TransportKeyData* parameters as a master key with the device indicated by the *PartnerAddress* sub-parameter only if the *SrcAddress* parameter is the same as the *apsTrustCenterAddress* attribute of the AIB. If accepted, the *DeviceKeyPairSet* attribute in AIB table will be updated. A key-pair descriptor 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 *MasterKey* element to the master key from the *TransportKeyData* parameter, and the *OutgoingFrameCounter* and *IncomingFrameCounter* elements to 0.

Next, if the *Initiator* sub-parameter of the *TransportKeyData* parameter of the APSME-TRANSPORT-KEY.indication primitive was TRUE, the device shall issue the APSME-ESTABLISH-KEY.request primitive. The *ResponderAddress* parameter shall be set to the *PartnerAddress* sub-parameter of the

TransportKeyData parameter, the *UseParent* parameter shall be set to FALSE, and the *KeyEstablishmentMethod* shall be set to 0x00 (that is, SKKE).

Upon receipt of the APSME-ESTABLISH-KEY.indication primitive, the responder device shall be informed that the initiator device wishes to establish a link key. If the responder decides to establish a link key, it shall issue the APSME-ESTABLISH-KEY.response primitive with the *InitiatorAddress* parameter set to the address of the initiator and the *Accept* parameter set to TRUE. Otherwise, it shall set the *Accept* parameter set to FALSE.

If the responder decided to set up a key with the initiator, the SKKE protocol will ensue and the APSME-ESTABLISH-KEY.confirm primitive will be issued to both the responder and initiator.

4.7.3.5.2 Trust Center Operation

Upon receipt of APSME-REQUEST-KEY.indication primitives with the *KeyType* parameter set to 0x02 (that is, application key), the trust center behavior depends on if it has been configured to send out application link keys or master keys.

The trust center shall issue two APSME-TRANSPORT-KEY.request primitives. If configured to send out application link keys the *KeyType* parameter shall be set to 0x03 (that is, application link key); otherwise, the *KeyType* parameter shall be set to 0x02 (that is, application master 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 (a master or 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, and the *Key* sub-parameter shall be set to *K*.

4.7.3.5.3 Message Sequence Chart

An example message sequence chart of the end-to-end application key establishment procedure is shown in Figure 4.24. The procedure begins with the transmission of the request-key command from the initiator to the trust center. Next, the trust center starts a time-out timer. For the duration of this timer (that is, until it expires), the trust center shall discard any new request-key commands for this pair of devices unless they are from the initiator.

The trust center shall now send transport-key commands containing the application link or master key to the initiator and responder devices. Only the initiator's transport-key command will have the *Initiator* field set to 1 (that is, TRUE), so if a master key was sent, only the initiator device will begin the key-establishment protocol by sending the SKKE-1 command. If the responder decides to accept establishing a key with the initiator, the SKKE protocol will progress via the exchange of the SKKE-2, SKKE-3, and SKKE-4 commands. 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.

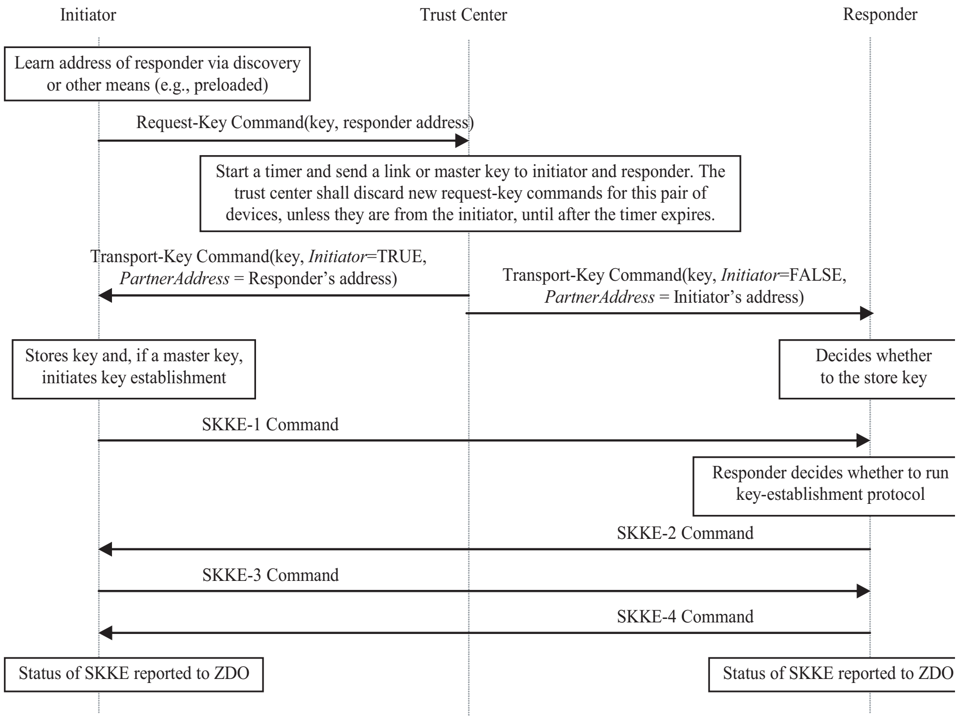

Initiator
Trust Center
Responder
Learn address of responder via discovery or other means (e.g., preloaded)
Request-Key Command(key, responder address)
Start a timer and send a link or master key to initiator and responder. The trust center shall discard new request-key commands for this pair of devices, unless they are from the initiator, until after the timer expires.
Transport-Key Command(key, Initiator=TRUE, PartnerAddress = Responder's address)
Transport-Key Command(key, Initiator=FALSE, PartnerAddress = Initiator's address)
Stores key and, if a master key, initiates key establishment
Decides whether to the store key
SKKE-1 Command
Responder decides whether to run key-establishment protocol
SKKE-2 Command
SKKE-3 Command
SKKE-4 Command
Status of SKKE reported to ZDO
Status of SKKE reported to ZDO

Figure 4.24 Example End-to-End Application Key Establishment Procedure

4.7.3.6 Network Leave

A device, its router, and the trust center shall follow the procedures described in this sub-clause when the device is to leave the network.

4.7.3.6.1 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 send 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. If operating in commercial mode, the trust center shall delete the leaving device from its list of network devices.

4.7.3.6.2 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, 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, the rejoin parameter set to 0 and the ReuseAddress parameter set to 1. If the device corresponding to the DeviceAddress parameter of the APSME-REMOVE-DEVICE.indication primitive is declared "joined but unauthenticated" (see sub-clause 4.7.3.2), the SilentLeave parameter shall be set to 1. Otherwise the the SilentLeave parameter shall be set to 0. Other fields are defined by the stack profile.[23]

The router shall ignore APSME-REMOVE-DEVICE.indication primitives with the SrcAddress parameter not equal to the apsTrustCenterAddress attribute of the AIB.[24]

- Upon receipt of an NLME-LEAVE.indication primitive with the *DeviceAddress* parameter set to one of its children, 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);

23. CCB #676
24. CCB #676

- 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 sub-clause 4.7.3.6.1).

4.7.3.6.3 Leaving Device Operation

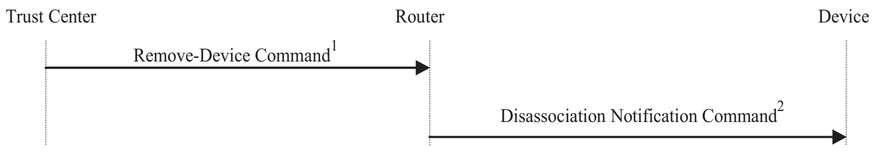
Devices are responsible for receiving and sending leave notification commands.

In a secured ZigBee network, leave[25] notification commands shall be secured with the Network key and sent with security enabled at the level indicated by the *nwkSecurityLevel* attribute in the NIB.

In a secured ZigBee network, leave[26] notification commands shall be received and processed only if secured with the Network key and received with security enabled at the level indicated by the *nwkSecurityLevel* attribute in the NIB.

4.7.3.6.4 Message Sequence Charts

Figure 4.25 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 Network key. Upon receipt of the remove-device command, a router shall send a disassociation notification command to the device to leave the network.



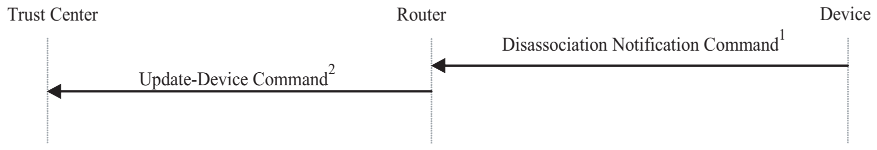
Note:
1. If a trust center wants a device to leave and if the trust center is not the router for that device, the trust center shall send the router a remove-device command with the address of the device it wishes to leave the network.
2. A router shall send a disassociation command to cause one of its children to leave the network.

Figure 4.25 Example Remove-Device Procedure

Figure 4.26 shows an example message sequence chart whereby a device notifies its router that it is disassociating from the network. In this example, the device sends a disassociation notification command (secured with the Network key) to its

25. CCB #676
26. CCB #676

router. The router then sends a device-update command to the trust center. In a secured network, the device-update command must be secured with the link key, if present, or the Network key.



Note:
1. A device leaving the network shall send a disassociation command to its router.
2. Upon receipt of a valid disassociation command, a router shall send an update-device command to the trust center to inform it that a device has left the network.

Figure 4.26 Example Device-Leave Procedure

4.7.3.7 Intra-PAN Portability

Devices shall follow the procedures described in this sub-clause as necessary to support portability, in conjunction with the mechanism described in 2.5.5.5.2.2.

At present, the security support for intra-PAN portability is minimized. In most cases security operation is not different from normal, as described in these sections.

4.7.3.7.1 Router Operation

This text describes the security operations for support of intra-PAN security which are to be carried out by the ZigBee Coordinator and by ZigBee Routers.

Following the steps described in 2.5.5.5.2.2, an orphaned 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 correctly formed ZigBee message to the new parent secured with the network key. 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 parent in the case that a message from an orphaned device fails security processing above that required by text elsewhere in this document.

Entity authentication of (re-)joined devices is neither specified nor required by this specification.

4.7.3.7.2 End-device Operation

Following the steps described in 2.5.5.5.2.2, an orphaned 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 to the new parent secured with the network key. Further, it shall verify that at least one ZigBee message is received from the new parent secured using the current network key. If this message successfully passes all the security processing steps described in this document, the device shall operate as a member of the network. If this protocol cannot be completed correctly, the device shall abandon its attempt to join this parent.

Entity authentication of the new parent is neither specified nor required by this specification.

As normal, when operating in residential mode the end device shall not accept an unsecured Network key from the trust center, except as required by this document when activated to join a new network. Accordingly power cycling of devices without persistent memory is not supported within this specification.

As normal, when operating in commercial mode the end device shall be configured to accept an updated network key from the trust center, even when not activated for the authentication procedure. In this case, the key shall be transported using the APSME-TRANSPORT-KEY.request primitive, and appropriate security shall be verified.

Note that a ZigBee router may also carry out an orphan scan as described in 2.5.5.5.2.2. In this case it shall, at this time, also follow the steps described in this sub-section.

4.7.3.7.3 Trust Center Operation

In this specification, there is no special additional role for the trust center above that of other routing devices.

As normal, when operating in residential mode the trust center shall not issue or reissue the Network key unsecured to orphaned devices, except when activated for the authentication procedure.

As normal, when operating in commercial mode the trust center may optionally issue or reissue the current or an updated network key to orphaned devices when not activated for the authentication procedure. In this case, the key shall be transported using the APSME-TRANSPORT-KEY.request primitive, and shall be secured appropriately. Initiation of such an update is beyond the scope of this specification.

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 with a 128-bit block size, such as AES-128 [B8]. The CCM* ideas can easily be extended to other block sizes, but this will require further definitions.

The CCM* mode coincides with the original CCM mode specification [B20] 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, [B21] and [B22], carries over to the CCM* mode described here. The design of the CCM* mode takes into account the results of [B23]; thus allowing it to be securely used in implementation environments for 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 is the 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 $0 \leq l(m)l < 28L$.

4 An octet string *a* of length *l*(*a*) octets, where $0 \leq l(a) < 2^{64}$.

The nonce *N* shall encode the potential values for *M* such that one can uniquely determine from *N* the actually used value of *M*. 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 $0 < l(a) < 2^{16}\text{-}2^8$, then *L(a)* is the 2-octets encoding of *l(a)*.
   c. If $2^{16}\text{-}2^8 \leq l(a) < 2^{32}$, 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 *l(a)* 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 || 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 || Adata || M || 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 sub-clause A.2.1 (step 4) and the authentication tag T that was established in sub-clause 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 follo33ws:

$Flags = Reserved \mid\mid Reserved \mid\mid 0 \mid\mid L$

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:

$A_i = Flags \mid\mid Nonce\ N \mid\mid Counter\ i,$ *for* $i=0, 1, 2, \ldots$

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 sub-clause A.2.2, step 1).

Parse the message *PlaintextData* as $M_1 \| \ldots \| M_t$, where each message block M_i is a 16-octet string.

The ciphertext blocks $C_1, \ldots, C_t$ are defined by:

$$C_i := E(Key, A_i) \oplus M_i \text{ for } i=1, 2, \ldots, t$$

The string *Ciphertext* is the result of omitting all but the leftmost *l(m)* octets of the string $C_1 \| \ldots \| C_t$

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 right-most 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 sub-clause A.2.3, with as inputs the data *CipherTextData* and the tag *U*.
4. Parse the output string resulting from applying this transformation as *m* || *T*, where the right-most 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 sub-clause A.2.1, with as inputs the string *a* and the octet string *m* that was established in sub-clause A.3.1 (step 4).
2. Use the authentication transformation in sub-clause A.2.2, with as input the message *AuthData*.
3. Compare the output tag *MACTag* resulting from this transformation with the tag *T* that was established in sub-clause 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 sub-clause A.2.3, with the following instantiations:

1. Each entity shall use the block-cipher E as specified in sub-clause 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 block-cipher based cryptographic hash function specified in Chapter 1, with the following instantiations:

1 Each entity shall use the block-cipher *E* as specified in section sub-clause B.1.1;

2 All integers and octets shall be represented as specified in Chapter 1.

The Matyas-Meyer-Oseas hash function (specified in Chapter 1) 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 sub-clause 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 sub-clause B.1.3.

B.1.5 Specialized Keyed Hash Function for Message Authentication

The specialized[27] keyed hash message authentication code used in this specification shall be the keyed hash message authentication code, as specified in sub-clause B.1.4.

27. This refers to a MAC scheme where the MAC function has the additional property that it is also pre-image and collision resistant for parties knowing the key (see also Remark 9.8 of [B18]). Such MAC functions allow key derivation in contexts where unilateral key control is undesirable.

B.1.6 Challenge Domain Parameters

The challenge domain parameters used in the specification shall be as specified in sub-clause B.3.1, with the following instantiation: (*minchallengelen, maxchallengelen*)=(128,128).

All challenges shall be validated using the challenge validation primitive as specified in Chapter 1.

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 as specified in Chapter 1, with the following instantiations:

1 Each entity shall be identified as specified in Chapter 1;

2 Each entity shall use the HMAC-scheme as specified in sub-clause B.1.4;

3 Each entity shall use the specialized HMAC-scheme as specified in sub-clause B.1.5;

4 Each entity shall use the cryptographic hash function as specified in sub-clause B.1.3.,

5 The parameter *keydatalen* shall have the same integer value as the key size parameter *keylen* as specified in sub-clause B.1.1;

6 The parameter *SharedData* shall be the empty string; parameter *shareddatalen* shall have the integer value 0;

7 The optional parameters $Text_1$ and $Text_2$ as specified in sub-clause B.7.1 and sub-clause B.7.2 shall both be the empty string.

8 Each entity shall use the challenge domain parameters as specified in sub-clause B.1.6.

9 All octets shall be represented as specified in section Chapter 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 determine 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 (e.g., 128-bit challenges). However, one may define the challenge domain parameters such that challenges of varying length might be accepted. The latter is useful in contexts where entities that wish to engage in cryptographic schemes might have a bad random number generator on-board. Allowing both entities that engage in a scheme to contribute sufficiently long inputs enables each of these to contribute sufficient entropy to the scheme at hand.

In this standard, challenge domain parameters will be shared by a number of entities using a scheme of 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 nonnegative 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

Challenge validation refers to the process of checking the length properties of a challenge. It is used to check whether a challenge to be used by a scheme in the standard has sufficient length (e.g., messages that are too short are discarded, due to insufficient entropy).

The challenge validation primitive is used in Chapter 1.

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* $\in$ [*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[28] MAC scheme shall have been 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:

$$MacTag = MAC_{MacKey}(MacData)$$

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 block-ciphers with a key size that is 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, we refer to Section 9.4.1 of [B18].

Prerequisites: The following are the prerequisites for the operation of Matyas-Meyer-Oseas hash function:

28. This refers to a MAC scheme where the MAC function has the additional property that it is also pre-image and collision resistant for parties knowing the key (see also Remark 9.8 of [B18]). Such MAC functions allow key derivation in contexts where unilateral key control is undesirable.

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 2^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 $0 \le l < 2^n$.

Actions: The hash value shall be derived as follows:

1. Pad the message M 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:

$$l+1+k \equiv 7n \ (\mathbf{mod}\ 8n) \qquad (1)$$

   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. Parse the padded message M' as $M_1 \| M_2 \| \ldots \| M_t$ where each message block M_i is an n-octet string.
3. The output $Hash_t$ is defined by

$$Hash_0 = 0^{8n};\ Hash_j = E(Hash_{j-1}, M_j) \oplus M_j \text{ for } j=1,\ldots,t \qquad (2)$$

   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 2^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 2^{16} bits. It is assumed that all hash function calls are on bit strings of length less than 2^n bits. Any scheme attempting to call the hash function on a bit string exceeding 2^n bits shall output 'invalid' and stop.

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.

Figure B.1 illustrates the messaging involved in the use of the full symmetric-key key agreement with key confirmation scheme.

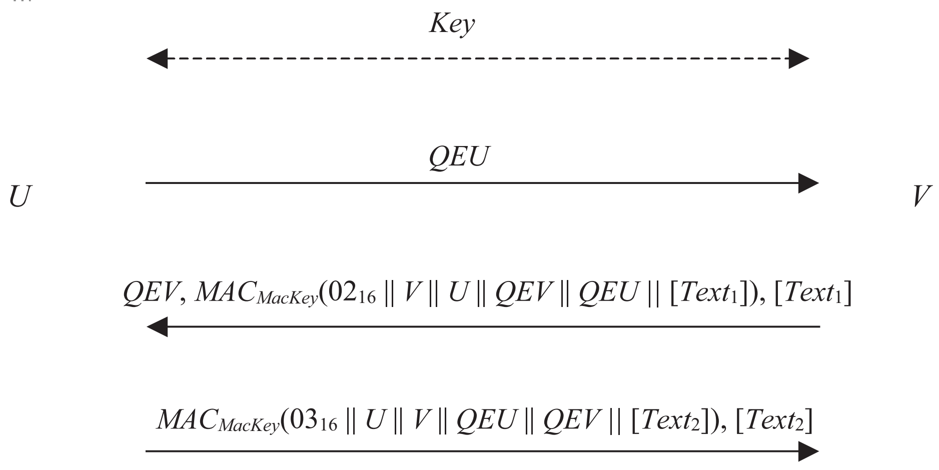

Key
QEU
U
V
QEV, $MAC_{MacKey}(02_{16} \| V \| U \| QEV \| QEU \| [Text_1])$, $[Text_1]$
$MAC_{MacKey}(03_{16} \| U \| V \| QEU \| QEV \| [Text_2])$, $[Text_2]$

Figure B.1 Symmetric-Key Authenticated Key Agreement Scheme

The scheme is 'asymmetric' so two transformations are specified. *U* uses the transformation specified in sub-clause B.7.1 to agree on keying data with *V* if *U* is the protocol's initiator, and *V* uses the transformation specified in sub-clause 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 merely 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 (e.g., 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[29] 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. (e.g., 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 in Section 5.3 of ANSI X9.63-2001 [B7], the challenge validation

29. This refers to a MAC function with the additional property that it is also pre-image and collision resistant for parties knowing the key (see also Remark 9.8 of [B18]. Specialized MAC functions allow key derivation in contexts where unilateral key control is undesirable.

primitive in sub-clause B.3.2, the SKG primitive in sub-clause 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 sub-clause B.3.2. If the validation primitive rejects the challenge, output ‘invalid’ and stop.

4 Use the SKG primitive in Chapter 1 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.

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 $Text_1$:

$MacData_1$ = 0216 || V || U || QEV’ || QEU || [Text1]

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]:

$$MacTag2 = MACMacKey(MacData2) \qquad (3)$$

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 in Section 5.3 of ANSI X9.63-2001 [B7], the challenge validation primitive in sub-clause B.3.2, the SKG primitive in Chapter 1, 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 sub-clause 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):

MacData2 = 0316 || U || V || QEU' || QEV || [Text2]

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 Chapter 1 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 $Text_1$:

MacData1 = 0216 || V || U || QEV || QEU' || [Text1]

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]:

$$MacTag_1 = MAC_{MacKey}(MacData_1)$$

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).

ANNEX

C

TEST VECTORS FOR CRYPTOGRAPHIC BUILDING BLOCKS

This annex provides sample test vectors for the ZigBee community, aimed at 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 sub-clause 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 sub-clause 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 tagging transformation as follows:

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 encryption transformation as follows:

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) ⊕ 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 sub-clause 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 right-most 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)	$P_i = AES(Key,A_i) \oplus C_i$
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 sub-clause C.3.1, with as inputs the string *a* and the octet string *m* that was established in sub-clause C.4.1 (step 7):

AuthData = 08 00 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 sub-clause C.3.2, with as input the message *AuthData* to compute the authentication tag *MACTag*:

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 sub-clause C.4.1 (step 9):
T = B9 D7 89 67 04 BC FA 20 = *MACTag*

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 C.5.

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 $Hash_1$ 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 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 $M_1 \parallel M_2$, 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.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 C.6.

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 C.6.3.

For test vectors, see clause C.6.

C.6.4 Symmetric-key Key Agreement Scheme

This annex provides sample test vectors for the symmetric-key key agreement scheme as specified in clause C.6.4.

Prerequisites: The following are the prerequisites for the use of the scheme:

1 The unique identifiers of the entities *U* and *V* to be used:

U's identifier: *U*=55 73 65 72 20 55 0D 0A

V's identifier: *V*=55 73 65 72 20 56 0D 0A

2 The key *Key* of length *keylen*=128 bits to be used:

Key = C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF

3 The optional parameter *SharedData* of length *shareddatalen*=48 bits to be used:

SharedData = D0 D1 D2 D3 D4 D5

C.6.5 Initiator Transformation

U obtains 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 The length *keydatalen* in bits of the keying data to be generated: *keydatalen*=128.

2 The optional bit string $Text_2$ to be used is not present, i.e., $Text_2 = \varepsilon$ (the empty string).

Actions: *U* derives keying data 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*.

```
QEU = 9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96
```

2 Then receive from *V* a challenge *QEV'* purportedly owned by *V*. If this value is not received, output 'invalid' and stop.

```
QEV' = BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53
```

3 Verify that *QEV'* is a valid challenge for the challenge domain parameters *D* as specified in sub-clause B.3.2. If the validation primitive rejects the challenge, output 'invalid' and stop.

4 Use the SKG primitive in clause 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.

a Form the bit string *MACData* = *U* || *V* || *QEU* || *QEV'*:

```
MACData = 55 73 65 72 20 55 0D 0A || 55 73 65 72 20 56 0D 0A ||
             9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96 ||
             BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53
```

b Calculate the *MACTag* for *MACData* under the key *Key* using the tagging transformation of the HMAC-Matyas-Meyer-Oseas MAC scheme:

$MACTag=MAC_{Key}(MACData)$ = 78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D

c Set *Z*=*MACTag*:

```
Z = 78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D
```

5 Use the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7] with the Matyas-Meyer-Oseas hash function to derive keying data *KKeyData* of length 256 bits from the shared secret value *Z* and the shared data *SharedData*:

a The hash values $Hash_1$, $Hash_2$ are computed as follows:

i	$Hash_i=H(X_i)$	X_i
1	E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50	78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D || 00 00 00 01 || D0 D1 D2 D3 D4 D5
2	72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F	78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D || 00 00 00 02 || D0 D1 D2 D3 D4 D5

b Set $KKeyData=Hash_1 \| Hash_2$:

```
KKeyData = E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50 ||
           72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F
```

6 Parse the leftmost 128 bits of *KKeyData* as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.

```
MacKey = E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50;

KeyData = 72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F
```

7 Form the bit string $MacData_1 = 02_{16} \| V \| U \| QEV' \| QEU \| [Text_1]$:

```
MacData1 = 02 || 55 73 65 72 20 56 0D 0A || 55 73 65 72 20 55 0D 0A ||
        BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53 ||
        9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96
```

8 Verify that $MacTag_1$' is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation specified in Section 5.7.2 of ANSI X9.63-2001 [B7]. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

a Calculate $MacTag_1=MAC_{MacKey}(MacData_1)$ = E6 C3 DE 1F E8 63 15 B9 E6 A0 2B 44 FF 63 D8 D0

b Verify that $MacTag_1=MacTag_1$'

9 Form the bit string $MacData_2 = 03_{16} \| U \| V \| QEU \| QEV' \| [Text_2]$:

```
MacData2 = 03 || 55 73 65 72 20 55 0D 0A || 55 73 65 72 20 56 0D 0A ||
           9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96 ||
           BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53
```

10 Calculate the tag $MacTag_2$ on $MacData_2$ under the key *MacKey* using the tagging transformation specified in Section 5.7.1 of ANSI X9.63-2001 [B7] with the HMAC-Matyas-Meyer-Oseas MAC scheme:

```
MacTag2 = MACMacKey (MacData2) = 66 36 8D 61 0F E0 0B 7F 06 3E
                                 74 C4 78 0A A3 6D
```

If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $MacTag_2$ and, if present, $Text_2$ to *V*.

11 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.

$Text_1$ = ε (the empty string);

12 $MacTag_1$' = E6 C3 DE 1F E8 63 15 B9 E6 A0 2B 44 FF 63 D8 D0

Output: output 'valid' and accept the 128-bit string *KeyData* = 72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F as the keying data shared with *V*.

C.6.6 Responder Transformation

V obtains 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*.

```
QEU' = 9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96
```

2 The length *keydatalen* in bits of the keying data to be generated: *keydatalen*=128

3 The optional bit string $Text_1$ to be used is not present, i.e., $Text_1$ = ε (the empty string).

Actions: *V* derives keying data as follows:

1 Verify that *QEU*' is a valid challenge for the challenge domain parameters *D* as specified in sub-clause 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*.

```
QEV = BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53
```

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.

```
Text2 = ε (the empty string);

MacTag2' = 66 36 8D 61 0F E0 0B 7F 06 3E 74 C4 78 0A A3 6D
```

4 Form the bit string $MacData_2 = 03_{16} \| U \| V \| QEU' \| QEV \| [Text_2]$:

```
MacData2 = 03 || 55 73 65 72 20 55 0D 0A || 55 73 65 72 20 56 0D 0A ||
           9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96 ||
           BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53
```

5 Verify that *$MacTag_2$’* is the valid tag on *$MacData_2$* under the key *MacKey* using the tag checking transformation specified in Section 5.7 ANSI X9.63-2001 [B7]. If the tag checking transformation outputs ‘invalid’, output ‘invalid’ and stop.

a Calculate $MacTag_2=MAC_{MacKey}(MacData_2)$ = 66 36 8D 61 0F E0 0B 7F 06 3E 74 C4 78 0A A3 6D

b Verify that $MacTag_2=MacTag_2$’

6 Use the SKG primitive in clause 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 *SharedKey*. If the SKG primitive outputs ‘invalid’, output ‘invalid’ and stop.

a Form the bit string *MACData=U* || *V* || *QEU’* || *QEV*:

```
MACData = 55 73 65 72 20 55 0D 0A || 55 73 65 72 20 56 0D 0A ||
          9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96 ||
          BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53
```

b Calculate the *MACTag* for *MACData* under the key *Key* using the tagging transformation of the HMAC-Matyas-Meyer-Oseas MAC scheme:

```
MACTag = MACKey (MACData) = 78 7C DE F6 80 13 12 CD 41 1B CD 62 14
                            91 F8 6D
```

c Set *Z=MACTag*:

```
Z = 78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D
```

7 Use the key derivation function in Section 5.6.3 of ANSI X9.63-2001 [B7] with the Matyas-Meyer-Oseas hash function to derive keying data *KKeyData* of length 256 bits from the shared secret value *Z* and the shared data *SharedData*:

a The hash values $Hash_1$, $Hash_2$ are computed as follows:

i	$Hash_i=H(X_i)$	X_i
1	E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50	78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D || 00 00 00 01 || D0 D1 D2 D3 D4 D5
2	72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F	78 7C DE F6 80 13 12 CD 41 1B CD 62 14 91 F8 6D || 00 00 00 02 || D0 D1 D2 D3 D4 D5

b Set $KKeyData=Hash_1 \| Hash_2$:

```
KKeyData = E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50 ||
           72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F
```

8 Parse the leftmost 128 bits of *KKeyData* as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.

```
MacKey = E4 D4 5F 76 2A 36 B7 99 F9 5E C2 C6 FD E9 A1 50;

KeyData = 72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F
```

9 Form the bit string $MacData_1 = 02_{16} \| V \| U \| QEV \| QEU' \| [Text_1]$:

```
MacData1 = 02 || 55 73 65 72 20 56 0D 0A || 55 73 65 72 20 55 0D 0A ||
               BF 14 DF 94 94 39 D2 CE 24 C9 09 53 B5 72 D6 53 ||
               9E 3F 0C 19 05 4B 05 44 D5 A7 17 62 0A F2 7D 96
```

10 Calculate the tag $MacTag_1$ for $MacData_1$ under the key *MacKey* using the tagging transformation specified in Section 5.7 of ANSI X9.63-2001 [B7] with the HMAC-Matyas-Meyer-Oseas MAC scheme:

```
MacTag1 = MACMacKey(MacData1)= E6 C3 DE 1F E8 63 15 B9 E6 A0 2B 44
                               63 D8 D0
```

11 If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send to *U*, if present the bit string $Text_1$, and $MacTag_1$.

Output: output 'valid' and accept the 128-bit string *KeyData* = 72 57 7D 02 CC E1 39 33 1A BF F4 0B C5 6E A3 7F as the keying data shared with *U*.

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 provided solutions to such problems, for use by the ZigBee Alliance.

D.1.3 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. Thus, for example, since the HC stack profile relies on a beaconless network, the platform compliance testing for the HS stack profile does not employ beaconing; the code to support regular beaconing, beacon track, and so on, may 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.

D.1.4 MAC Association

At association time, according to the IEEE 802.15.4 2003 specification, a number of frames, 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. Tables D.1–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			