IEEE P802.15

Wireless Personal Area Networks

Project IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs)

Title Intel and ETRI merged proposal text

Date SubmittedOct 2009

Source R. Roberts, B. Sadeghi, P. Gopalakrishnan, M. Walma (Intel); S.Y. Chang, M.H. Son (ETRI)

E-mail: sychang@ecs.csus.edu; richard.d.roberts@intel.com

Intel&ETRI Version of IEEE802.15.7 Draft D0

IEEE Standard for
Information technology—
Telecommunications and information exchange between systems—
Local and metropolitan area networks—
Specific requirements—

Part 15.7: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)

1. Overview

Visible Light Communications (VLC) is envisioned to be used in a variety of applications generally falling into one of the following topology classifications: peer-to-peer, where peers may be fixed, mobile or vehicular mounted, and infrastructure-to-mobile/vehicular. For VLC in infrastructure topologies, it is essential to support communications features that coexist without interruption to the primary use of LEDs for lighting. In many cases, this lighting will be diffuse with overlapping footprints of coverage.

This document defines a standard for a visible light communications WPAN (VAN).

1.1 Scope

The scope of this standard is to define the physical layer (PHY) and medium access control (MAC) sublayer specifications for visible light communications wireless connectivity with fixed, portable, and moving devices operating in the personal operating space (POS).

1.2 Purpose

This document provides an international standard for a visible light communications PHY and MAC to satisfy an evolutionary set of industrial and consumer requirements for wireless personal area network (WPAN) communications.

2. References

3. Definitions

4. Acronyms and abbreviations

5. General Description

5.1 (Informative) Modulation Domain Spectrum

It will help the reader of this specification to understand such concepts as VLC CCA by thinking in the modulation domain. The “modulation domain” is based upon the premise (at the time of the writing of this specification) that VLC receivers are photodetector based and hence basically the receiver non-coherently detects the envelope of the lightwave carrier. The modulation domain is defined as what we observe at the output of the photodetector. So when the standard mentions detecting a carrier, the reference point for detecting said carrier is at the photodetector output, which was modulated on the lightwave carrier. That is, CCA is not detecting the presence of “light” but rather detecting the presence of modulation on a lightwave carrier (i.e. modulation domain). The following figure tries to better illustrate this concept.

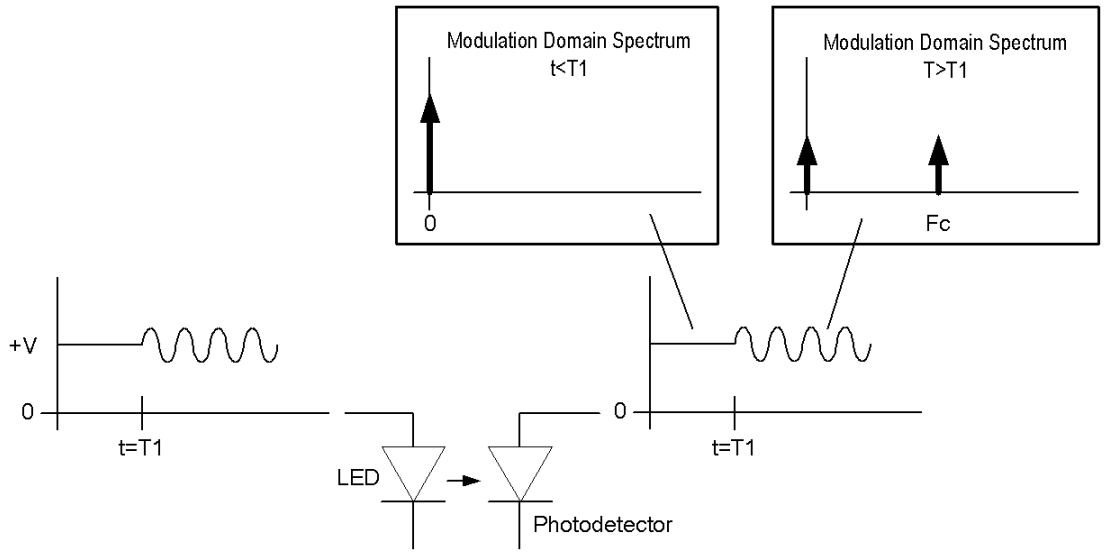


Figure 1—Illustration of Modulation Domain Spectrum

In Figure X we can see that the LED is "always on". When we say we do CCA we are implying that we want to detect when the modulation is applied, in our example at t=T1. As observed at the output of the photodetector, prior to t=T1 the spectrum is all at DC, while after t=T1 the spectrum is split between DC and the carrier frequency. So in this example CCA is not detecting if the LED is illuminated - it is illuminated all the time - rather the CCA is detecting when the modulation of interest is applied.

5.2 Introduction

An VAN is a simple, low-cost communication network that allows wireless connectivity in applications with limited power and relaxed throughput requirements. The main objectives of an VAN are ease of installation, reliable data transfer, short-range operation, extremely low cost, and a reasonable battery life, while maintaining a simple and flexible protocol.

Some of the characteristics of an VAN are as follows:

- Star or peer-to-peer operation
- Allocated 16-bit short or 64-bit extended addresses
- Optional allocation of guaranteed time slots (GTSs)
- Random access with optional collision avoidance.
- Fully acknowledged protocol for transfer reliability
- Link quality indication (LQI)

Two different device types can participate in an IEEE 802.15.7 network; a full-function device (FFD) and a reduced-function device (RFD). The FFD can operate in two modes serving as a personal area network (VAN) coordinator or a device. An FFD can talk to RFDs or other FFDs, while an RFD can talk only to an FFD or in a broadcast-only mode. 15.7.

5.3 Components of the IEEE 802.15.7 VAN

A system conforming to this standard consists of several components. The most basic is the device. A device may be an RFD or an FFD. Two or more devices within a POS communicating on the same physical channel constitute a VAN. To form a network VAN shall include at least one FFD, operating as the VAN -coordinator.

An IEEE 802.15.7 network is part of the VAN family of standards although the coverage of the network may extend beyond the POS, which typically defines the VPAN.

A well-defined coverage area does not exist for wireless media because propagation characteristics are dynamic and uncertain. Small changes in position or direction may result in drastic differences in the signal strength or quality of the communication link. These effects occur whether a device is stationary or mobile, as moving objects may impact station-to-station propagation.

5.4 Network topologies

Depending on the application requirements, an IEEE 802.15.7 VAN may operate in either of two topologies: the star topology or the peer-to-peer topology. Both are shown in Figure 1. In the star topology the communication is established between devices and a single central controller, called the VAN coordinator. A device typically has some associated application and is either the initiation point or the termination point for network communications. A VAN coordinator may also have a specific application, but it can be used to initiate, terminate, or route communication around the network. The VAN coordinator is the primary controller of the VAN. All devices operating on a network of either topology shall have unique 64-bit addresses. This address may be used for direct communication within the VAN, or a short address may be allocated by the VAN coordinator when the device associates and used instead. The VAN coordinator might often be mains powered, while the devices will most likely be battery powered. Applications that benefit from a star topology include home automation, personal computer (PC) peripherals, toys and games, and personal health care.

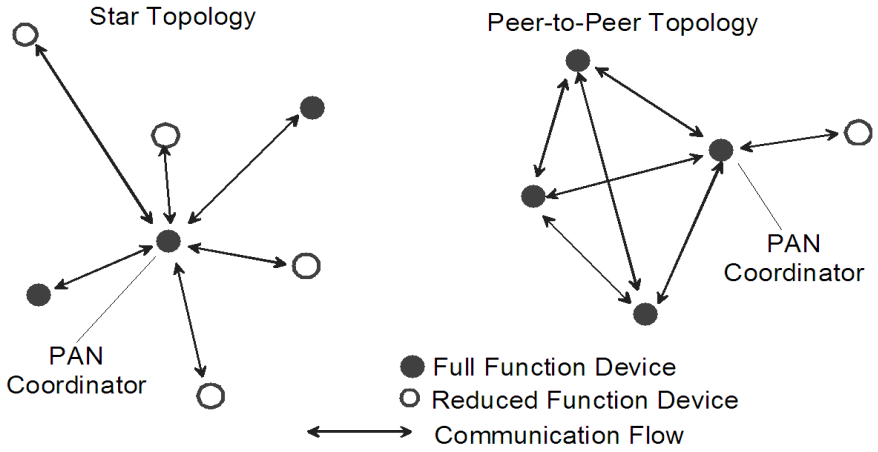


Figure 2—Star and peer-to-peer topology examples

The peer-to-peer topology also has a VAN coordinator; however, it differs from the star topology in that any device may communicate with any other device as long as they are in range of one another. A peer-to-peer network can be ad hoc, self-organizing, and self-healing. It may also allow multiple hops to route messages

from any device to any other device on the network. Such functions can be added at the higher layer, but are not part of this standard.

Each independent VAN selects a unique identifier. This VAN identifier allows communication between devices within a network using short addresses.. The mechanism by which identifiers are chosen is outside the scope of this standard.

The network formation is performed by the higher layer, which is not part of this standard. However, 5.3.1 and 5.3.2 provide a brief overview on how each supported topology can be formed. Apart from the two topologies, IEEE 802.15.7 devices may also operate in a broadcast only mode without being part of a network, i.e., without being associated to any device or having any devices associated to them.

5.4.1 Star network formation

The basic structure of a star network is illustrated in Figure 1. After an FFD is activated, it can establish its own network and become the VAN coordinator. All star networks operate independently from all other star networks currently in operation. This is achieved by choosing a VAN identifier that is not currently used by any other network within the radio sphere of influence. Once the VAN identifier is chosen, the VAN coordinator allows other devices, potentially both FFDs and RFDs, to join its network. The higher layer can use the procedures described in 7.5.2 and 7.5.3 to form a star network.

5.4.2 Peer-to-peer network formation

In a peer-to-peer topology, each device is capable of communicating with any other device within its radio sphere of influence. One device is nominated as the VAN coordinator, for instance, by virtue of being the first device to communicate on the channel. Further network structures are constructed out of the peer-to-peer topology and it is possible to impose topological restrictions on the formation of the network.

5.5 Architecture

The IEEE 802.15.7 architecture is defined in terms of a number of blocks in order to simplify the standard. These blocks are called layers. Each layer is responsible for one part of the standard and offers services to the higher layers. The layout of the blocks is based on the open systems interconnection (OSI) seven-layer model (see ISO/IEC 7498-1:1994 [B13]).

The interfaces between the layers serve to define the logical links that are described in this standard.

An VAN device comprises a PHY, which contains the radio frequency (RF) transceiver along with its low-level control mechanism, and a MAC sublayer that provides access to the physical channel for all types of transfer. Figure 3 shows these blocks in a graphical representation, which are described in more detail in 5.4.1 and 5.4.2

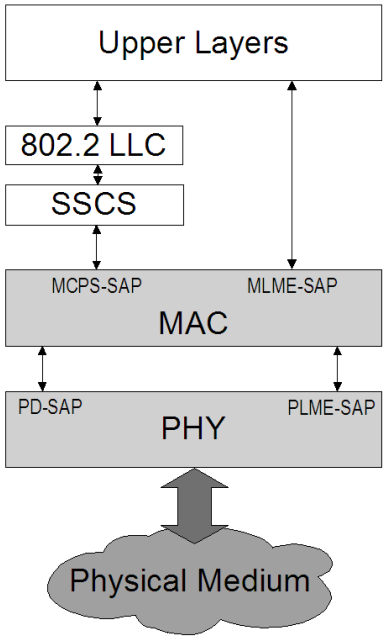


Figure 3—LR-WVAN device architecture

The upper layers, shown in Figure 3, consist of a network layer, which provides network configuration, manipulation, and message routing, and an application layer, which provides the intended function of the device. The definition of these upper layers is outside the scope of this standard. An IEEE 802.2 Type 1 logical link control (LLC) can access the MAC sublayer through the service-specific convergence sublayer (SSCS), defined in Annex A. The VAN architecture can be implemented either as embedded devices or as devices requiring the support of an external device such as a PC.

5.5.1 PHY

The PHY layer supports two PHY types:

- PHY TYPE 1 - 10 kbps to 100 kbps using amplitude modulation

- PHY TYPE 2 - 1 Mbps to 10 Mbps using amplitude modulation

5.5.1.1 PHY TYPE 1

This PHY type is intended for applications that trade-off Eb/No for range, such as vehicular communications and outdoor information broadcast. This mode uses Manchester encoded OOK (or VPM if it is shown to be better) modulation at a chipping rate of 200 kcps. The burst data rate is 100 kbps without additional coding. Additional coding is used to achieve data rates of 75 kbps, 50 kbps, and 25 kbps. The basic link establishment rate is 25 kbps.

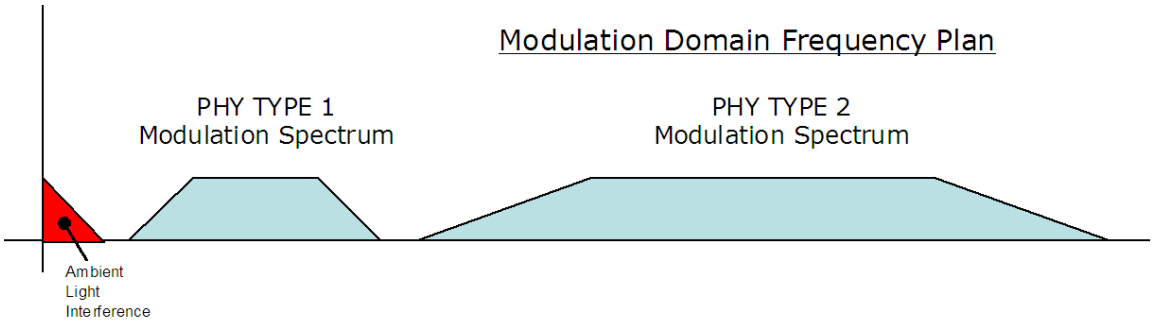
5.5.1.2 PHY TYPE 2

This PHY type is intended for applications that trade-off range for Eb/No, such as indoor information broadcast and VLAN. This mode uses Manchester encoded OOK modulation (or VPM if it is shown to be better) at a chipping rate of 20 Mcps. The burst data rate is 10 Mbps without additional coding. Additional coding is used to achieve a data rates of 5 Mbps and 1 Mbps. The basic link establishment rate is 1 Mbps.

5.5.1.3 Coexistence Between PHY TYPE 1 and PHY TYPE 2

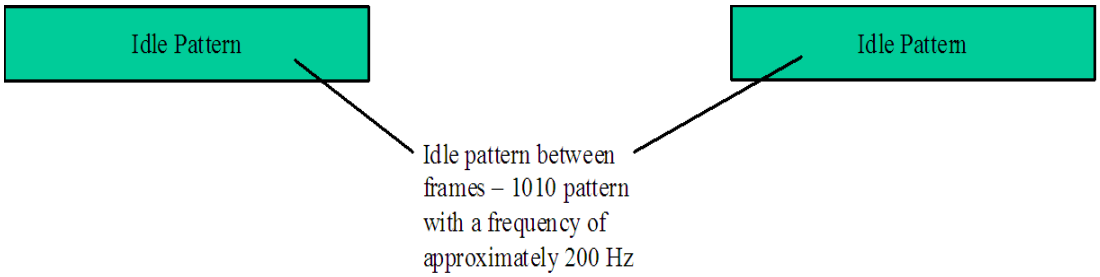
5.5.1.3.1 PHY TYPES 1 and 2

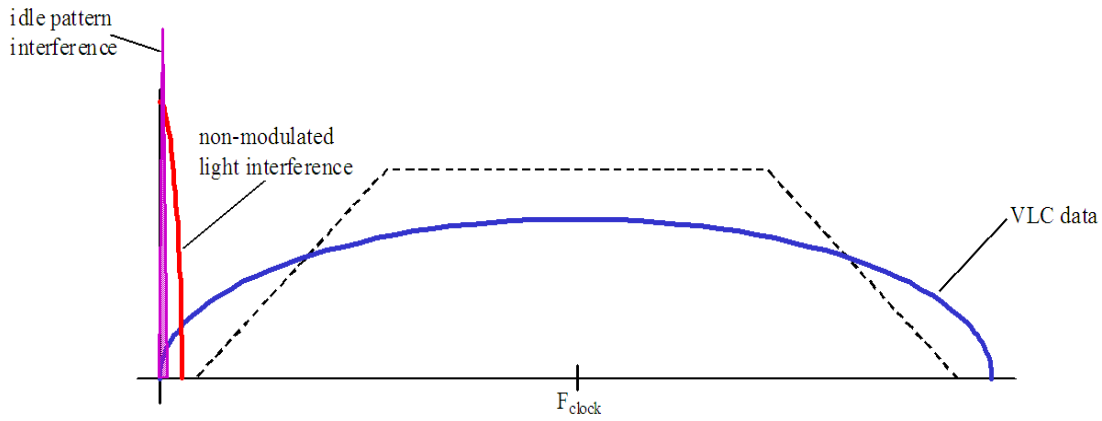
PHY types 1 and 2 occupy different spectral regions in the modulation domain spectrum which hence gives them FDM as a coexistence mechanism.



5.5.1.4 Frame Flicker Compensation

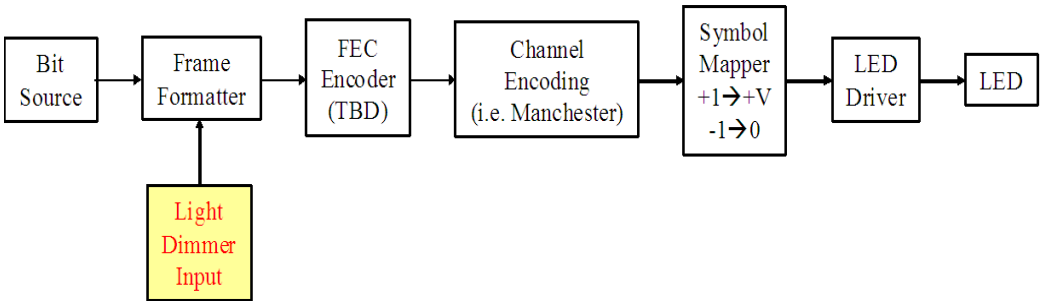
During the packet transfer time the average optical power decreases by 3 dB. To maintance a constant output power between packets an idle pattern is sent with a repetition rate that is signficantly less than the chipping rate. Exact repetition rate is TBD. Because of the reduced repetition rate the modulation domain spectral energy is shifted below the spectrum used by PHY Type 1. Frame flicker compensation is used by both PHY types.



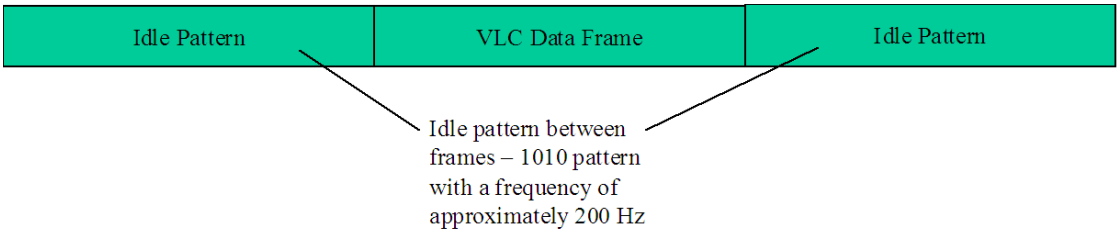


5.5.1.5 Accommodation of Light Dimmers

The information in regards to light dimming requirements is intercepted by the VLC modulator as shown below.



OFF time is inserted into either the idle pattern or into the data frame, as shown below, to reduce the average intensity of the light.



The impact on the data rate is the data rate decreases as the light gets dimmer.

5.5.2 MAC sublayer

The MAC sublayer provides two services: the MAC data service and the MAC management service interfacing to the MAC sublayer management entity (MLME) service access point (SAP) (known as MLME-

SAP). The MAC data service enables the transmission and reception of MAC protocol data units (MPDUs) across the PHY data service.

The features of the MAC sublayer are beacon management, channel access, GTS management, frame validation, acknowledged frame delivery, association, and disassociation. In addition, the MAC sublayer provides hooks for implementing application-appropriate security mechanisms.

Clause 7 contains the specifications for the MAC sublayer.

5.6 Functional overview

A brief overview of the general functions of a VAN is given in 5.5.1 through 5.6.8 and includes information on the superframe structure, the data transfer model, the frame structure, robustness, power consumption considerations, precision ranging, and security.

5.6.1 Superframe structure

This standard allows the optional use of a superframe structure. The format of the superframe is defined by the coordinator. The superframe is bounded by network beacons sent by the coordinator [see Figure 4a)] and is divided into 16 equally sized slots. Optionally, the superframe can have an active and an inactive portion [see Figure 4b)]. During the inactive portion, the coordinator may enter a low-power mode. The beacon frame is transmitted in the first slot of each superframe. If a coordinator does not wish to use a superframe structure, it will turn off the beacon transmissions. The beacons are used to synchronize the attached devices, to identify the VAN, and to describe the structure of the superframes. Any device wishing to communicate during the contention access period (CAP) between two beacons competes with other devices using slotted random access .

All transactions are completed by the time of the next network beacon.

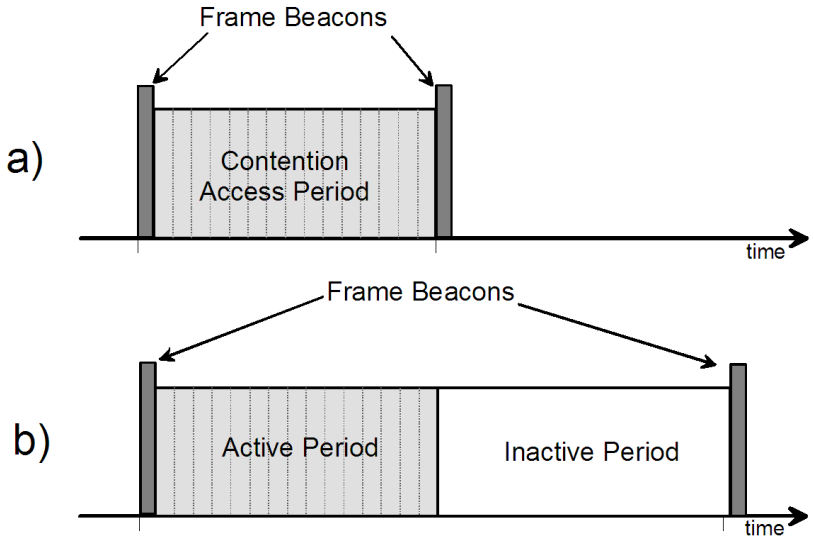


Figure 4—Superframe structure without GTSs

For low-latency applications or applications requiring specific data bandwidth, the VAN coordinator may dedicate portions of the active superframe to that application. These portions are called guaranteed time slots

(GTSs). The GTSs form the contention-free period (CFP), which always appears at the end of the active superframe starting at a slot boundary immediately following the CAP, as shown in Figure 5. The VAN coordinator may allocate up to seven of these GTSs, and a GTS may occupy more than one slot period. However, a sufficient portion of the CAP remains for contention-based access of other networked devices or new devices wishing to join the network. All contention-based transactions is completed before the CFP begins. Also each device transmitting in a GTS ensures that its transaction is complete before the time of the next GTS or the end of the CFP. More information on the superframe structure can be found in 7.5.1.1

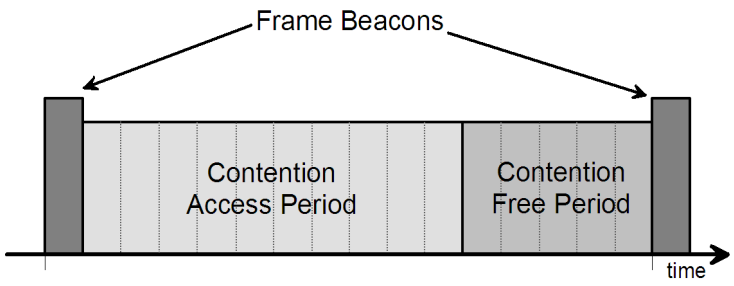


Figure 5—Superframe structure with GTSs

5.6.2 Data transfer model

Three types of data transfer transactions exist. The first one is the data transfer to a coordinator in which a device transmits the data. The second transaction is the data transfer from a coordinator in which the device receives the data. The third transaction is the data transfer between two peer devices. In star topology, only two of these transactions are used because data may be exchanged only between the coordinator and a device. In a peer-to-peer topology, data may be exchanged between any two devices on the network; consequently all three transactions may be used in this topology.

The mechanisms for each transfer type depend on whether the network supports the transmission of beacons. A beacon-enabled VAN is used in networks that either require synchronization or support for low-latency devices, such as PC peripherals. If the network does not need synchronization or support for low latency devices, it can elect not to use the beacon for normal transfers. However, the beacon is still required for network discovery. The structure of the frames used for the data transfer is specified in 7.2.

5.6.3 Data transfer to a coordinator

When a device wishes to transfer data to a coordinator in a beacon-enabled VAN, it first listens for the network beacon. When the beacon is found, the device synchronizes to the superframe structure. At the appropriate time, the device transmits its data frame, using slotted random access to the coordinator. The coordinator may acknowledge the successful reception of the data by transmitting an optional acknowledgment frame. This sequence is summarized in Figure 6.

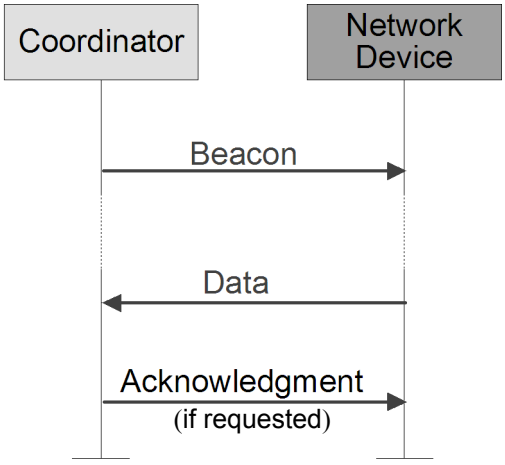


Figure 6—Communication to a coordinator in a beacon-enabled VAN

When a device wishes to transfer data in a nonbeacon-enabled VAN, it simply transmits its data frame, using unslotted random access, to the coordinator. The coordinator acknowledges the successful reception of the data by transmitting an optional acknowledgment frame. The transaction is now complete. This sequence is summarized in Figure 7.

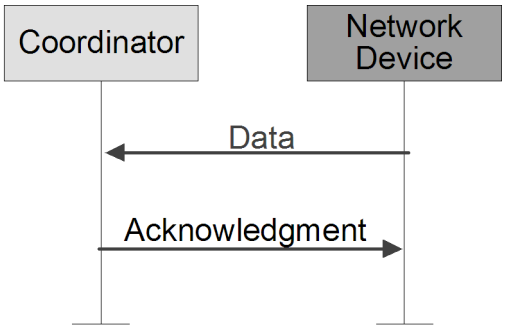


Figure 7—Communication to a coordinator in a nonbeacon-enabled VAN

5.6.3.1 Data transfer from a coordinator

When the coordinator wishes to transfer data to a device in a beacon-enabled VAN, it indicates in the network beacon that the data message is pending. The device periodically listens to the network beacon and, if a message is pending, transmits a MAC command requesting the data, using slotted random access . The coordinator acknowledges the successful reception of the data request by transmitting an acknowledgment frame. The pending data frame is then sent using slotted random access , or, if possible, immediately after the acknowledgment (see 7.5.6.3). The device may acknowledge the successful reception of the data by transmitting an optional acknowledgment frame. The transaction is now complete. Upon successful completion of the data transaction, the message is removed from the list of pending messages in the beacon. This sequence is summarized in Figure 8.

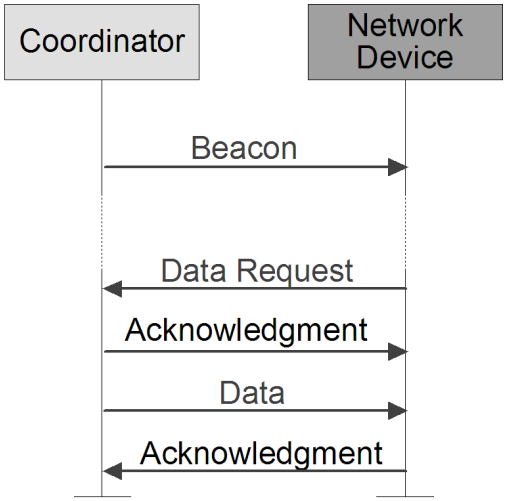


Figure 8—Communication from a coordinator a beacon-enabled VAN

When a coordinator wishes to transfer data to a device in a nonbeacon-enabled VAN, it stores the data for the appropriate device to make contact and request the data. A device may make contact by transmitting a MAC command requesting the data, using unslotted random access to its coordinator at an application-defined rate. The coordinator acknowledges the successful reception of the data request by transmitting an acknowledgment frame. If a data frame is pending, the coordinator transmits the data frame, using unslotted random access to the device. If a data frame is not pending, the coordinator indicates this fact either in the acknowledgment frame following the data request or in a data frame with a zero-length payload (see 7.5.6.3). If requested, the device acknowledges the successful reception of the data frame by transmitting an acknowledgment frame. This sequence is summarized in Figure 9.

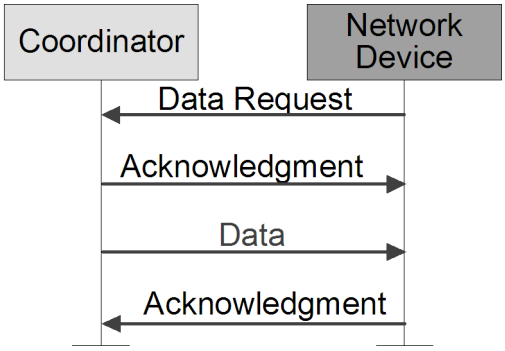


Figure 9—Communication from a coordinator in a nonbeacon-enabled VAN

5.6.3.2 Peer-to-peer data transfers

In a peer-to-peer VAN, every device may communicate with every other device in its radio sphere of influence. In order to do this effectively, the devices wishing to communicate will need to either receive constantly or synchronize with each other. In the former case, the device can simply transmit its data

usingrandom access . In the latter case, other measures need to be taken in order to achieve synchronization. Such measures are beyond the scope of this standard.

5.6.4 Frame structure

The frame structures have been designed to keep the complexity to a minimum while at the same time making them sufficiently robust for transmission on a noisy channel. Each successive protocol layer adds to the structure with layer-specific headers and footers. This standard defines four frame structures:

- A beacon frame, used by a coordinator to transmit beacons
- A data frame, used for all transfers of data
- An acknowledgment frame, used for confirming successful frame reception
- A MAC command frame, used for handling all MAC peer entity control transfers

The structure of each of the four frame types is described in 5.5.3.1 through 5.5.3.4. The diagrams in these subclauses illustrate the fields that are added by each layer of the protocol.

5.6.4.1 Beacon frame

Figure 10 shows the structure of the beacon frame, which originates from within the MAC sublayer. A coordinator can transmit network beacons in a beacon-enabled VAN. The MAC payload contains the superframe specification, GTS fields, pending address fields, and beacon payload (see 7.2.2.1). The MAC payload is prefixed with a MAC header (MHR) and appended with a MAC footer (MFR). The MHR contains the MAC Frame Control field, beacon sequence number (BSN), addressing fields, and optionally the auxiliary security header. The MFR contains a 16-bit frame check sequence (FCS). The MHR, MAC payload, and MFR together form the MAC beacon frame (i.e., MPDU).

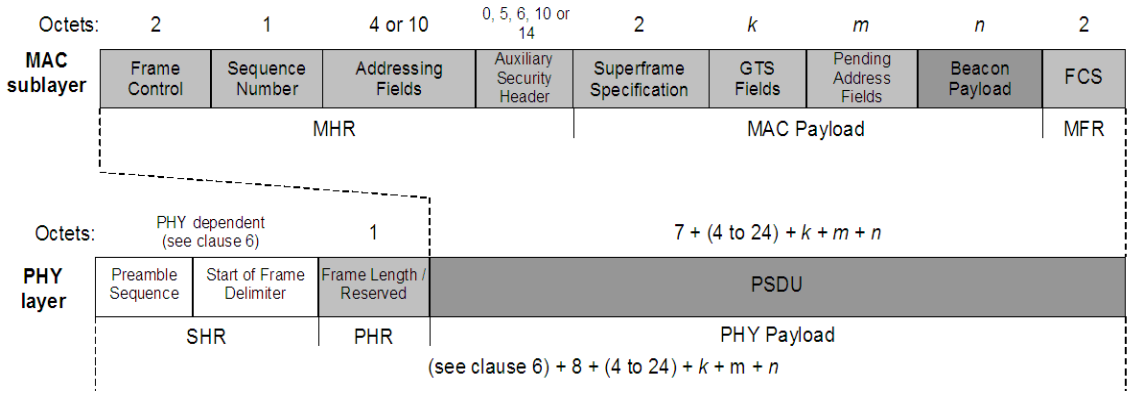


Figure 10—Schematic view of the beacon frame and the PHY packet

The MAC beacon frame is then passed to the PHY as the PHY service data unit (PSDU), which becomes the PHY payload. The PHY payload is prefixed with a synchronization header (SHR), containing the Preamble Sequence and Start-of-Frame Delimiter (SFD) fields, and a PHY header (PHR) containing the length of the PHY payload in octets. The SHR, PHR, and PHY payload together form the PHY packet (i.e., PPDU).

5.6.4.2 Data frame

Figure 11 shows the structure of the data frame, which originates from the upper layers.

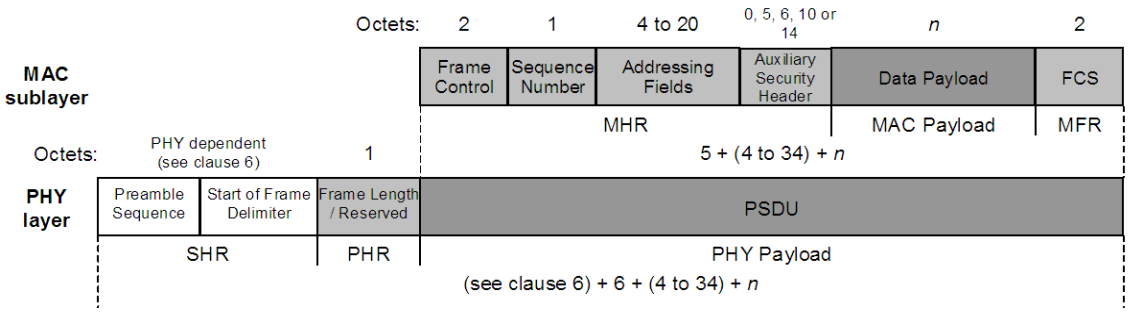


Figure 11—Schematic view of the data frame and the PHY packet

The data payload is passed to the MAC sublayer and is referred to as the MAC service data unit (MSDU). The MAC payload is prefixed with an MHR and appended with an MFR. The MHR contains the Frame Control field, data sequence number (DSN), addressing fields, and optionally the auxiliary security header. The MFR is composed of a 16-bit FCS. The MHR, MAC payload, and MFR together form the MAC data frame, (i.e., MPDU).

The MPDU is passed to the PHY as the PSDU, which becomes the PHY payload. The PHY payload is prefixed with an SHR, containing the Preamble Sequence and SFD fields, and a PHR containing the length of the PHY payload in octets. The preamble sequence and the data SFD enable the receiver to achieve symbol synchronization. The SHR, PHR, and PHY payload together form the PHY packet, (i.e., PPDU).

5.6.4.3 Acknowledgment frame

Figure 12 shows the structure of the acknowledgment frame, which originates from within the MAC sublayer. The MAC acknowledgment frame is constructed from an MHR and an MFR; it has no MAC payload. The MHR contains the MAC Frame Control field and DSN. The MFR is composed of a 16-bit FCS. The MHR and MFR together form the MAC acknowledgment frame (i.e., MPDU).

The MPDU is passed to the PHY as the PSDU, which becomes the PHY payload. The PHY payload is prefixed with the SHR, containing the Preamble Sequence and SFD fields, and the PHR containing the length of the PHY payload in octets. The SHR, PHR, and PHY payload together form the PHY packet, (i.e., PPDU).

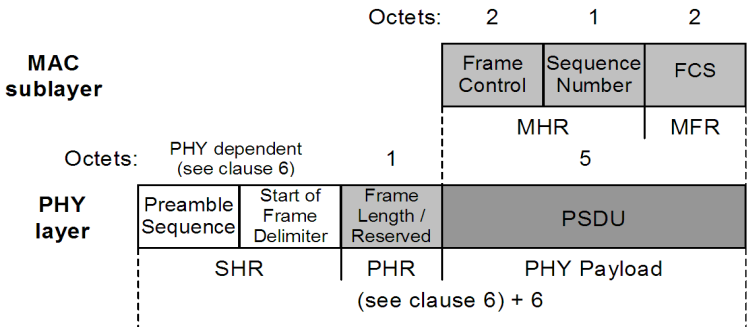


Figure 12—Schematic view of the acknowledgment frame and the PHY packet

5.6.4.4 MAC command frame

Figure 13 shows the structure of the MAC command frame, which originates from within the MAC sublayer. The MAC payload contains the Command Type field and the command payload (see 7.2.2.4). The MAC payload is prefixed with an MHR and appended with an MFR. The MHR contains the MAC Frame Control field, DSN, addressing fields, and optionally the auxiliary security header. The MFR contains a 16-bit FCS. The MHR, MAC payload, and MFR together form the MAC command frame, (i.e., MPDU).

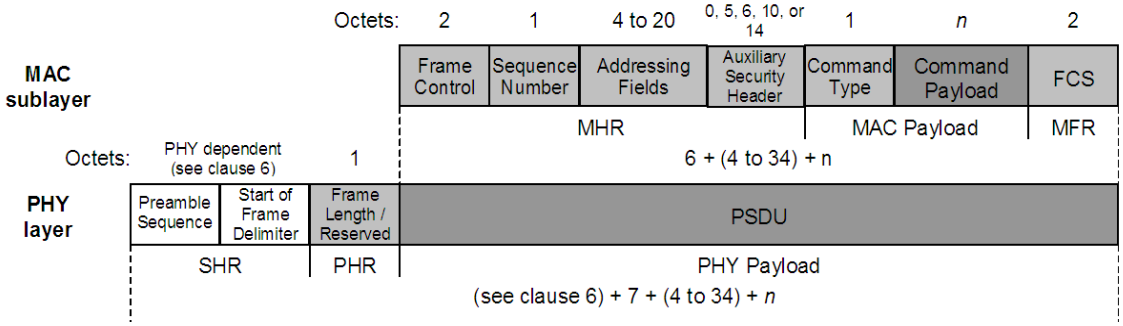


Figure 13—Schematic view of the MAC command frame and the PHY packet

The MPDU is then passed to the PHY as the PSDU, which becomes the PHY payload. The PHY payload is prefixed with an SHR, containing the Preamble Sequence and SFD fields, and a PHR containing the length of the PHY payload in octets. The preamble sequence enables the receiver to achieve symbol synchronization. The SHR, PHR, and PHY payload together form the PHY packet, (i.e., PPDU).

5.6.5 Improving probability of successful delivery

The IEEE 802.15.7 VAN employs various mechanisms to improve the probability of successful data transmission. These mechanisms are the random access, frame acknowledgment, and data verification and are briefly discussed in 5.5.4.1 through 5.5.4.4.

5.6.5.1 Random access mechanism

The IEEE 802.15.7 VAN uses two types of channel access mechanism, depending on the network configuration. Nonbeacon-enabled PANs use an unslotted random channel access mechanism, as described in 7.5.1.

Each time a device wishes to transmit data frames or MAC commands, it waits for a random period. Following the random backoff, the device transmits its data. If the optional carrier sense mechanism is active and the channel is found to be busy following the random backoff, the device waits for another random period before trying to access the channel again. Acknowledgment frames are sent without using a random access mechanism.

Beacon-enabled PANs use a slotted random channel access mechanism, where the backoff slots are aligned with the start of the beacon transmission. The backoff slots of all devices within one VAN are aligned to the VAN coordinator. Each time a device wishes to transmit data frames during the CAP, it locates the boundary of the next backoff slot and then waits for a random number of backoff slotsIf theoptional collision avoidance mechanism is active and the channel is busy, following this random backoff, the device waits for another random number of backoff slots before trying to access the channel again. If the channel is idle or the optional carrier sense mechanism is not active, the device begins transmitting on the next available backoff slot boundary. Acknowledgment and beacon frames are sent without using a random access mechanism.

5.6.5.2 Frame acknowledgment

A successful reception and validation of a data or MAC command frame can be optionally confirmed with an acknowledgment, as described in 7.5.6.4. If the receiving device is unable to handle the received data frame for any reason, the message is not acknowledged.

If the originator does not receive an acknowledgment after some period, it assumes that the transmission was unsuccessful and retries the frame transmission. If an acknowledgment is still not received after several retries, the originator can choose either to terminate the transaction or to try again. When the acknowledgment is not required, the originator assumes the transmission was successful.

5.6.5.3 Data verification

In order to detect bit errors, an FCS mechanism employing a 16-bit International Telecommunication Union-Telecommunication Standardization Sector (ITU-T) cyclic redundancy check (CRC) is used to detect errors in every frame.

The FCS mechanism is discussed in 7.2.1.9.

5.6.6 Power consumption considerations

TBD

5.6.7 Security

<editor's note: this section and the security mechanism needs to be reviewed for its applicability to VLC>

From a security perspective, wireless ad hoc networks are no different from any other wireless network. They are vulnerable to passive eavesdropping attacks and potentially even active tampering because physical access to the wire is not required to participate in communications. The very nature of ad hoc networks and their cost objectives impose additional security constraints, which perhaps make these networks the most difficult environments to secure. Devices are low-cost and have limited capabilities in terms of computing power, available storage, and power drain; and it cannot always be assumed they have a trusted computing base nor a high-quality random number generator aboard. Communications cannot rely on the online availability of a fixed infrastructure and might involve short-term relationships between devices that may never have communicated before. These constraints might severely limit the choice of cryptographic algorithms and protocols and would influence the design of the security architecture because the establishment and maintenance of trust relationships between devices need to be addressed with care. In addition, battery lifetime and cost constraints put severe limits on the security overhead these networks can tolerate, some-

thing that is of far less concern with higher bandwidth networks. Most of these security architectural elements can be implemented at higher layers and may, therefore, be considered to be outside the scope of this standard.

The cryptographic mechanism in this standard is based on symmetric-key cryptography and uses keys that are provided by higher layer processes. The establishment and maintenance of these keys are outside the scope of this standard. The mechanism assumes a secure implementation of cryptographic operations and secure and authentic storage of keying material.

The cryptographic mechanism provides particular combinations of the following security services:

- Data confidentiality: Assurance that transmitted information is only disclosed to parties for which it is intended.

- Data authenticity: Assurance of the source of transmitted information (and, hereby, that information was not modified in transit).

- Replay protection: Assurance that duplicate information is detected.

The actual frame protection provided can be adapted on a frame-by-frame basis and allows for varying levels of data authenticity (to minimize security overhead in transmitted frames where required) and for optional data confidentiality. When nontrivial protection is required, replay protection is always provided.

Cryptographic frame protection may use a key shared between two peer devices (link key) or a key shared among a group of devices (group key), thus allowing some flexibility and application-specific tradeoffs between key storage and key maintenance costs versus the cryptographic protection provided. If a group key is used for peer-to-peer communication, protection is provided only against outsider devices and not against potential malicious devices in the key-sharing group.

For more detailed information on the cryptographic security mechanisms used for protected MAC frames following this standard, refer to Clause 7.

5.7 Concept of primitives

This subclause provides a brief overview of the concept of service primitives. Refer to IEEE Std 802.2-1998 for more detailed information.

The services of a layer are the capabilities it offers to the user in the next higher layer or sublayer by building its functions on the services of the next lower layer. This concept is illustrated in Figure 14, showing the service hierarchy and the relationship of the two correspondent N-users and their associated N-layer (or sublayer) peer protocol entities.

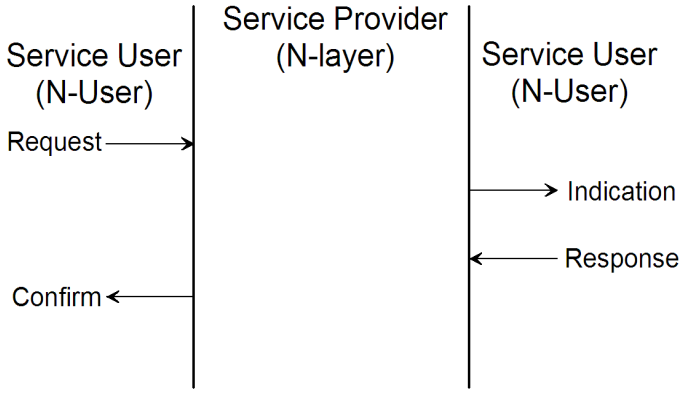


Figure 14—Service primitives

The services are specified by describing the information flow between the N-user and the N-layer. This information flow is modeled by discrete, instantaneous events, which characterize the provision of a service. Each event consists of passing a service primitive from one layer to the other through a layer SAP associated with an N-user. Service primitives convey the required information by providing a particular service. These service primitives are an abstraction because they specify only the provided service rather than the means by which it is provided. This definition is independent of any other interface implementation.

Services are specified by describing the service primitives and parameters that characterize it. A service may have one or more related primitives that constitute the activity that is related to that particular service. Each service primitive may have zero or more parameters that convey the information required to provide the service.

A primitive can be one of four generic types:

— Request: The request primitive is passed from the N-user to the N-layer to request that a service is initiated.

— Indication: The indication primitive is passed from the N-layer to the N-user to indicate an internal N-layer event that is significant to the N-user. This event may be logically related to a remote service request, or it may be caused by an N-layer internal event.

— Response: The response primitive is passed from the N-user to the N-layer to complete a procedure previously invoked by an indication primitive.

— Confirm: The confirm primitive is passed from the N-layer to the N-user to convey the results of one or more associated previous service requests.

6. PHY sublayer specification

This clause specifies three PHY options for IEEE 802.15.7. The PHY is responsible for the following tasks:

— Activation and deactivation of the VL transceiver

— LQI for received packets

— Channel selection

— Data transmission and reception

Constants and attributes that are specified and maintained by the PHY are written in the text of this clause in italics. Constants have a general prefix of “a”, e.g., aMaxPHYPacketSize, and are listed in Table X. Attributes have a general prefix of “phy”, e.g., phyCurrentChannel, and are listed in Table X.

6.1 General requirements and definitions

This subclause specifies requirements that are common to both of the IEEE 802.15.7 PHYs.

6.1.1 Operating frequency range

A compliant device shall operate in one or several visible light frequency bands as shown in table X using the modulation and spreading formats summarized in Table Y.

Frequency band (nm)		Spectral width (nm)	Color	Proposed Code
380	450	70	pB	000
450	510	60	B, BG	001
510	560	50	G	010
560	600	40	yG,gY,	011
600	650	50	rO	100
650	710	60	R	101
710	780	70	R	110
			Reserved	111

This standard is intended to conform with established regulations in Europe, Japan, Canada, and the United States. The regulatory documents listed below are for information only and are subject to change and revisions at any time. IEEE 802.15.4 devices shall also comply with specific regional legislation. Additional regulatory information is provided in Annex F.

Canada:

Europe:

Japan:

Korea:

United States:

6.1.2 Operating modes

A compliant IEEE802.15.7 PHY shall implement at least one of the following PHY modes.

Table 1—PHY Operating Modes

PHY TYPE	MCS Identifier	Spreading Parameters			Data Parameters	
		Inner Code Rate	Outer Code Rate	Modulation	Chip/ Sample Rate	Bit Rate
TYPE 1	3	1/2	1/4	OOK (or VPM)	200 kcps	25 kbps
	4	1/2	1/2	OOK (or VPM)	200 kcps	50 kbps
	5	1/2	none	OOK (or VPM)	200 kcps	100 kbps
TYPE 2	6	1/2	1/10	OOK (or VPM)	20 Mcps	1 Mbps
	7	1/2	1/2	OOK (or VPM)	20 Mcps	5 Mbps
	8	1/2	none	OOK (or VPM)	20 Mcps	10 Mbps

6.1.3 Channel assignments and numbering

TBD

6.1.4 Optical power measurement

TBD

6.1.5 Transmit power

The maximum transmit power shall conform with local regulations. Refer to Annex F for additional information on regulatory limits. A compliant device shall have its nominal transmit power level indicated by its PHY parameter, phyTransmitPower (see 6.4).

6.1.6 Out-of-band spurious emission

TBD - does this even make sense for VLC?

6.1.7 Receiver sensitivity definitions

The definitions in Table 2 are referenced by subclauses elsewhere in this standard regarding receiver sensitivity.

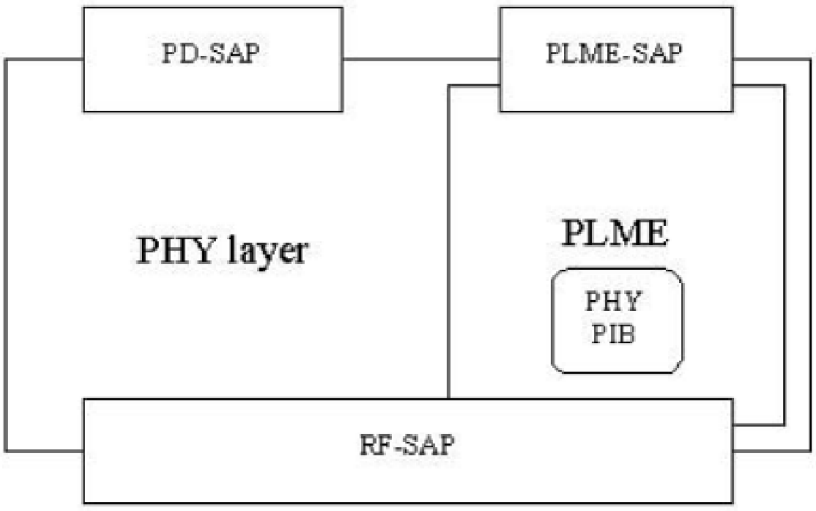
Table 2—Receiver sensitivity definitions

Term	Definition of term	Conditions
Packet error rate (PER)	Average fraction of transmitted packets that are not detected correctly.	- Average measured over random PSDU data.
Receiver sensitivity	Threshold input signal power that yields a specified PER.	- PSDU length = 20 octets - PER <1% - Optical power measured at photodetector - Interference not present

6.2 PHY service specifications

The PHY provides an interface between the MAC sublayer and the physical radio channel, via the RF firmware and RF hardware. The PHY conceptually includes a management entity called the PLME. This entity provides the layer management service interfaces through which layer management functions may be invoked. The PLME is also responsible for maintaining a database of managed objects pertaining to the PHY. This database is referred to as the PHY PAN information base (PIB).

Figure X depicts the components and interfaces of the PHY.



The PHY provides two services, accessed through two SAPs: the PHY data service, accessed through the PHY data SAP (PD-SAP), and the PHY management service, accessed through the PLME's SAP (PLME-SAP).

6.2.1 PHY data service

The PD-SAP supports the transport of MPDUs between peer MAC sublayer entities. Table 1 lists the primitives supported by the PD-SAP. These primitives are discussed in the subclauses referenced in the table.

Table 3—PD-SAP primitives

PD-SAP primitive	Request	Confirm	Indication
PD-DAtA	**6.x.x.x**	**6.x.x.x**	**6.x.x.x**

6.2.1.1 PD-DATA.request

The PD-DATA.request primitive requests the transfer of an MPDU (i.e., PSDU) from the MAC sublayer to the local PHY entity.

6.2.1.1.1 Semantics of the service primitive

The semantics of the PD-DATA.request primitive is as follows:

PD-DATA.request (

psduLength,

psdu

)

Table 2 specifies the parameters for the PD-DATA.request primitive.

Table 4—PD-DATA.request parameters

Name	Type	Valid range	Description
psduLength	Unsigned Integer	<=aMaxPHYPacketSize	The number of octets in the PSDU to be transmitted by the PHY entity.
psdu	Set of octets	-	The set of octets forming the PSDU to be transmitted by the PHY entity.

6.2.1.1.2 When generated

The PD-DATA.request primitive is generated by a local MAC sublayer entity and issued to its PHY entity to request the transmission of an MPDU.

6.2.1.1.3 Effect on receipt

The receipt of the PD-DATA.request primitive by the PHY entity will cause the transmission of the supplied PSDU. Provided the transmitter is enabled (TX_ON state), the PHY will first construct a PPDU, containing the supplied PSDU, and then transmit the PPDU. When the PHY entity has completed the transmission, it will issue the PD-DATA.confirm primitive with a status of SUCCESS.

If the PD-DATA.request primitive is received while the receiver is enabled (RX_ON state) or if the transceiver is disabled (TRX_OFF state), the PHY entity will issue the PD-DATA.confirm primitive with a status of RX_ON or TRX_OFF, respectively.

6.2.1.2 PD-DATA.confirm

The PD-DATA.confirm primitive confirms the end of the transmission of an MPDU (i.e., PSDU) from a local MAC sublayer entity to a peer MAC sublayer entity.

6.2.1.2.1 Semantics of the service primitive

The semantics of the PD-DATA.confirm primitive is as follows:

```
PD-DATA.confirm     (
                    status
                    )
```

Table 3 specifies the parameters for the PD-DATA.confirm primitive.

Table 5—PD-DATA.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	SUCCESS, RX_ON, or TRX_OFF	The result of the request to transmit a packet

6.2.1.2.2 When generated

The PD-DATA.confirm primitive is generated by the PHY entity and issued to its MAC sublayer entity in response to a PD-DATA.request primitive. The PD-DATA.confirm primitive will return a status of either SUCCESS, indicating that the request to transmit was successful, or an error code of RX_ON or TRX_OFF. The reasons for these status values are fully described in 6.2.1.1.3.

6.2.1.2.3 Effect on receipt

On receipt of the PD-DATA.confirm primitive, the MAC sublayer entity is notified of the result of its request to transmit. If the transmission attempt was successful, the status parameter is set to SUCCESS. Otherwise, the status parameter will indicate the error.

6.2.1.3 PD-DATA.indication

The PD-DATA.indication primitive indicates the transfer of an MPDU (i.e., PSDU) from the PHY to the local MAC sublayer entity.

6.2.1.3.1 Semantics of the service primitive

The semantics of the PD-DATA.indication primitive is as follows:

```
PD-DATA.indication          (
                            psduLength,
                            psdu,
                            ppduLinkQuality
                            )
```

Table 6 specifies the parameters for the PD-DATA.indication primitive.

Table 6—PD-DATA.indication parameters

Name	Type	Valid	Description
psduLength	Unsigned Integer	<=aMaxPHYPacketSize	The number of octets contained in the PSDU received by the PHY entity.
psdu	Set of octets	-	The set of octets forming the PSDU received by the PHY entity.
ppduLinkQuality	Integer	0x00-0 x ff	Link quality (LQ) value measured during reception of the PPDU (see 6.x.x).

6.2.1.3.2 When generated

The PD-DATA.indication primitive is generated by the PHY entity and issued to its MAC sublayer entity to transfer a received PSDU. This primitive will not be generated if the received psduLength field is zero or greater than aMaxPHYPacketSize.

6.2.1.3.3 Effect on receipt

On receipt of the PD-DATA.indication primitive, the MAC sublayer is notified of the arrival of an MPDU across the PHY data service.

6.2.2 PHY management service

The PLME-SAP allows the transport of management commands between the MLME and the PLME. Table 5 lists the primitives supported by the PLME-SAP. These primitives are discussed in the clauses referenced in the table.

Table 7—PLME-SAP primitives

PLME-SAP primitive	Request	Confirm
PLME-CCA	6.x.x.x	6.x.x.x
PLME-GET	6.x.x.x	6.x.x.x
PLME-SET-TRX-STATE	6.x.x.x	6.x.x.x
PLME-SET	6.x.x.x	6.x.x.x
PLME-DIMMER	6.x.x.x	6.x.x.x

6.2.2.1 PLME-CCA.request

The PLME-CCA.request primitive requests that the PLME perform a CCA as defined in 6.7.9.

6.2.2.1.1 Semantics of the service primitive

The semantics of the PLME-CCA.request primitive is as follows:

PLME-CCA.request ()

There are no parameters associated with the PLME-CCA.request primitive.

6.2.2.1.2 When generated

The PLME-CCA.request primitive is generated by the MLME and issued to its PLME whenever the CSMACA algorithm requires an assessment of the channel.

6.2.2.1.3 Effect on receipt

If the receiver is enabled on receipt of the PLME-CCA.request primitive, the PLME will cause the PHY to perform a CCA. When the PHY has completed the CCA, the PLME will issue the PLME-CCA.confirm primitive with a status of either BUSY or IDLE, depending on the result of the CCA.

If the PLME-CCA.request primitive is received while the transceiver is disabled (TRX_OFF state) or if the transmitter is enabled (TX_ON state), the PLME will issue the PLME-CCA.confirm primitive with a status of TRX_OFF or TX_ON, respectively.

6.2.2.2 PLME-CCA.confirm

The PLME-CCA.confirm primitive reports the results of a CCA.

6.2.2.2.1 Semantics of the service primitive

The semantics of the PLME-CCA.confirm primitive is as follows:

```
PLME-CCA.confirm        (
                        status
                        )
```

Table 8 specifies the parameters for the PLME-CCA.confirm primitive.

Table 8—PLME-CCA.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	TRX_OFF, TX_ON, BUSY, or IDLE	The result of the request to perform a CCA.

6.2.2.2.2 When generated

The PLME-CCA.confirm primitive is generated by the PLME and issued to its MLME in response to a PLME-CCA.request primitive. The PLME-CCA.confirm primitive will return a status of either BUSY or IDLE, indicating a successful CCA, or an error code of TRX_OFF or TX_ON. The reasons for these status values are fully described in 6.2.2.1.3.

6.2.2.2.3 Effect on receipt

On receipt of the PLME-CCA.confirm primitive, the MLME is notified of the results of the CCA. If the CCA attempt was successful, the status parameter is set to either BUSY or IDLE. Otherwise, the status parameter will indicate the error.

6.2.2.3 PLME-GET.confirm

The PLME-GET.confirm primitive reports the results of an information request from the PHY PIB.

6.2.2.3.1 Semantics of the service primitive

The semantics of the PLME-GET.confirm primitive is as follows:

```
PLME-GET.confirm        (
                        status,
                        PIBAttribute,
```

PIBAttributeValue

)

Table 11 specifies the parameters for the PLME-GET.confirm primitive.

Table 9—PLME-GET.confirm parameters

Name	Type	Valid range	Description
Status	Enumeration	SUCCESS or UNSUPPORTED_ATTRIBUTE	The result of the request for PHY PIB attribute information.
PIBAttribute	Enumeration	See Table X	The identifier of the PHY PIB attribute to get.
PIBAttributeValue	Various	Attribute specific	The value of the indicated PHY PIB attribute to get.

6.2.2.3.2 When generated

The PLME-GET.confirm primitive is generated by the PLME and issued to its MLME in response to a PLME-GET.request primitive. The PLME-GET.confirm primitive will return a status of either SUCCESS, indicating that the request to read a PHY PIB attribute was successful, or an error code of UNSUPPORTED_ATTRIBUTE. The reasons for these status values are fully described in subclause 6.2.2.5.3.

6.2.2.3.3 Effect on receipt

On receipt of the PLME-GET.confirm primitive, the MLME is notified of the results of its request to read a PHY PIB attribute. If the request to read a PHY PIB attribute was successful, the status parameter is set to SUCCESS. Otherwise, the status parameter will indicate the error.

6.2.2.4 PLME-SET-TRX-STATE.request

The PLME-SET-TRX-STATE.request primitive requests that the PHY entity change the internal operating state of the transceiver. The transceiver will have three main states:

— Transceiver disabled (TRX_OFF).

— Transmitter enabled (TX_ON).

— Receiver enabled (RX_ON).

6.2.2.4.1 Semantics of the service primitive

The semantics of the PLME-SET-TRX-STATE.request primitive is as follows:

PLME-SET-TRX-STATE.request (

state

)

Table 12 specifies the parameters for the PLME-SET-TRX-STATE.request primitive.

Table 10—PLME-SET-TRX-STATE.request parameters

Name	Type	Valid range	Description
state	Enumeration	RX_ON, TRX_OFF, FORCE_TRX_OFF, or TX_ON	The new state in which to configure the transceiver.

6.2.2.4.2 When generated

The PLME-SET-TRX-STATE.request primitive is generated by the MLME and issued to its PLME when the current operational state of the receiver needs to be changed.

6.2.2.4.3 Effect on receipt

On receipt of the PLME-SET-TRX-STATE.request primitive, the PLME will cause the PHY to change to the requested state. If the state change is accepted, the PHY will issue the PLME-SET-TRX-STATE.confirm primitive with a status of SUCCESS. If this primitive requests a state that the transceiver is already configured, the PHY will issue the PLME-SET-TRX-STATE.confirm primitive with a status indicating the current state, i.e., RX_ON, TRX_OFF, or TX_ON. If this primitive is issued with RX_ON or TRX_OFF argument and the PHY is busy transmitting a PPDU, the PHY will issue the PLME-SET-TRXSTATE.confirm primitive with a status BUSY_TX and defer the state change till the end of transmission. If this primitive is issued with TX_ON or TRX_OFF argument and the PHY is in RX_ON state and has already received a valid SFD, the PHY will issue the PLME-SET-TRX-STATE.confirm primitive with a status BUSY_RX and defer the state change till the end of reception of the PPDU. If this primitive is issued with FORCE_TRX_OFF, the PHY will cause the PHY to go the TRX_OFF state irrespective of the state the PHY is in.

6.2.2.5 PLME-SET-TRX-STATE.confirm

The PLME-SET-TRX-STATE.confirm primitive reports the result of a request to change the internal operating state of the transceiver.

6.2.2.5.1 Semantics of the service primitive

The semantics of the PLME-SET-TRX-STATE.confirm primitive is as follows:

PLME-SET-TRX-STATE.confirm (

status

)

Table 13 specifies the parameters for the PLME-SET-TRX-STATE.confirm primitive.

Table 11—PLME-SET-TRX-STATE.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	SUCCESS, RX_ON, TRX_OFF, TX_ON, BUSY_RX, or BUSY_TX	The result of the request to change the state of the transceiver.

6.2.2.5.2 When generated

The PLME-SET-TRX-STATE.confirm primitive is generated by the PLME and issued to its MLME after attempting to change the internal operating state of the transceiver.

6.2.2.5.3 Effect on receipt

On receipt of the PLME-SET-TRX-STATE.confirm primitive, the MLME is notified of the result of its request to change the internal operating state of the transceiver. A status value of SUCCESS indicates that the internal operating state of the transceiver was accepted. A status value of RX_ON, TRX_OFF, or TX_ON indicates that the transceiver is already in the requested internal operating state. A status value of BUSY_TX is issued when the PHY is requested to change its state to RX_ON or TRX_OFF while transmitting. A status value of BUSY_RX is issued when the PHY is in RX_ON state, has already received a valid SFD, and is requested to change its state to TX_ON or TRX_OFF.

6.2.2.6 PLME-SET.request

The PLME-SET.request primitive attempts to set the indicated PHY PIB attribute to the given value.

6.2.2.6.1 Semantics of the service primitive

The semantics of the PLME-SET.request primitive is as follows:

```
PLME-SET.request        (
                        PIBAttribute
                        PIBAttributeValue
                        )
```

Table 14 specifies the parameters for the PLME-SET.request primitive.

6.2.2.6.2 When generated

The PLME-SET.request primitive is generated by the MLME and issued to its PLME to write the indicated PHY PIB attribute.

Table 12—PLME-SET.request parameters

Name	Type	Valid range	Description
PIBAttribute	Enumeration	See Table X	The identifier of the PIB attribute to set.
PIBAttributeValue	Various	Attribute specific	The value of the indicated PIB attribute to set.

6.2.2.6.3 Effect on receipt

On receipt of the PLME-SET.request primitive, the PLME will attempt to write the given value to the indicated PHY PIB attribute in its database. If the PIBAttribute parameter specifies an attribute that is not found in the database (see Table 19), the PLME will issue the PLME-SET.confirm primitive with a status of UNSUPPORTED_ATTRIBUTE. If the PIBAttibuteValue parameter specifies a value that is out of the valid range for the given attribute, the PLME will issue the PLME-SET.confirm primitive with a status of INVALID_PARAMETER.

If the requested PHY PIB attribute is successfully written, the PLME will issue the PLME-SET.confirm primitive with a status of SUCCESS.

6.2.2.6.4 PLME-SET.confirm

The PLME-SET.confirm primitive reports the results of the attempt to set a PIB attribute.

6.2.2.6.5 Semantics of the service primitive

The semantics of the PLME-SET.confirm primitive is as follows:

PLME-SET.confirm (

status,

PIBAttribute

)

Table 15 specifies the parameters for the PLME-SET.confirm primitive.

Table 13—PLME-SET.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	SUCCESS, UNSUPPORTED_ATTRIBUTE, or INVALID_PARAMETER	The status of the attempt to set the requested PIB attribute.
PIBAttribute	Enumeration	See Table X	The identifier of the PIB attribute being confirmed.

6.2.2.6.6 When generated

The PLME-SET.confirm primitive is generated by the PLME and issued to its MLME in response to a PLME-SET.request primitive. The PLME-SET.confirm primitive will return a status of either SUCCESS, indicating that the requested value was written to the indicated PHY PIB attribute, or an error code of UNSUPPORTED_ATTRIBUTE or INVALID_PARAMETER. The reasons for these status values are fully described in subclause 6.2.2.9.3.

6.2.2.6.7 Effect on receipt

On receipt of the PLME-SET.confirm primitive, the MLME is notified of the result of its request to set the value of a PHY PIB attribute. If the requested value was written to the indicated PHY PIB attribute, the status parameter is set to SUCCESS. Otherwise, the status parameter will indicate the error.

6.2.2.7 PLME-DIMMER.request

The PLME-DIMMER.request primitive requests that the PLME perform the dimmer function as defined in 6.7.7.

6.2.2.7.1 Semantics of the service primitive

The semantics of the PLME-DIMMER.request primitive is as follows:

```
PLME-DIMMER.request     (
                        PIBDim
                        PIBDimValue
                        )
```

Table 14 specifies the parameters for the PLME-DIMMER.request primitive.

Table 14—PLME-DIMMER.request parameters

Name	Type	Valid range	Description
PIBAttribute	Enumeration	See Table X	The identifier of the PIB attribute to set.
PIBAttributeValue	Various	Attribute specific	The value of the indicated PIB attribute to set.

6.2.2.7.2 When generated

The PLME-DIMMER.request primitive is generated by the MLME and issued to its PLME whenever the dimmer algorithm requires the required dimming value.

6.2.2.7.3 Effect on receipt

If the transmitter is enabled on receipt of the PLME-DIMMER.request primitive, the PLME will cause the PHY to perform the required dimming function. When the PHY has completed the required dimming, the PLME will issue the PLME-DIMMER.confirm primitive with the status of COMPLETE.

6.2.2.8 PLME-DIMMER.confirm

The PLME-DIMMER.confirm primitive reports the results of a dimming request.

6.2.2.8.1 Semantics of the service primitive

The semantics of the PLME-DIMMER.confirm primitive is as follows:

PLME-DIMMER.confirm (

status

)

Table 8 specifies the parameters for the PLME-DIMMER.confirm primitive.

Table 15—PLME-DIMMER.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	TRX_OFF, TX_ON, BUSY, or IDLE	The result of the request to perform a CCA.

6.2.2.8.2 When generated

The PLME-DIMMER.confirm primitive is generated by the PLME and issued to its MLME in response to a PLME-DIMMER.request primitive. The PLME-DIMMER.confirm primitive will return a status of either BUSY or IDLE, indicating a successful dimming. The reasons for these status values are fully described in 6.x.x.x.x.

6.2.2.8.3 Effect on receipt

On receipt of the PLME-DIMMER.confirm primitive, the MLME is notified of the results of the dimming. If the CCA attempt was successful, the status parameter is set to either BUSY or IDLE. Otherwise, the status parameter will indicate the error.

6.2.3 PHY enumerations description

Table 16 shows a description of the PHY enumeration values defined in the PHY specification.

Table 16—PHY enumerations description
Editor Note: these PHY enumerations have to be modified for VLC

Enumeration	Value	Description
BUSY	0x00	THe CCA attempt has detected a busy channel.
BUSY_RX	0x01	The transceiver is asked to change its state while receiving.
BUSY_TX	0x02	The transceiver is asked to change its state while transmitting.
FORCE_TRX_OFF	0x03	The transceiver is to be switched off.
IDLE	0x04	The CCA attempt has detected an idel channel.
INVALID_PARAMETER	0x05	A SET/GET request was issued with a parameter in the primitive that is out of the valid range.
RX_ON	0x06	The transceiver is in, or is to be configured into, the receiver enabled state.
SUCCESS	0x07	A SET/GET, an ED operation, or a transceiver state change was successful.
TRX_OFF	0x08	The transceiver is in, or is to be configured into, the transceiver disabled state.
TX_ON	0x09	The transceiver is in, or is to be configured into, the transmitter enabled state.
UNSUPPORTED_ATTRIBUTE	0x0a	A SET/GET request was issued with the identifier of an attribute that is not supported.

6.3 PPDU format

This clause specifies the format of the PPDU packet.

For convenience, the PPDU packet structure is presented so that the leftmost field as written in this standard shall be transmitted or received first. All multiple octet fields shall be transmitted or received least significant octet first and each octet shall be transmitted or received least significant bit (LSB) first. The same transmission order should apply to data fields transferred between the PHY and MAC sublayer.

Each PPDU packet consists of the following basic components:

— A SHR, which allows a receiving device to synchronize and lock onto the bit stream.

— A PHR, which contains frame length information.

— A variable length payload, which carries the MAC sublayer frame.

6.3.1 General packet format

The PPDU packet structure shall be formatted as illustrated in Figure 16.

Table 17—Format of the PPDU

Octets: 4	**1**	**1**		**variable**	**TBD**
Preamble	SFD	Frame length (7 bits)	Reserved (1 bit)	PSDU	Frame Check Sequence
SHR		PHR		PHY payload	FCS

6.3.1.1 Preamble field

The preamble field is used by the transceiver to obtain chip and symbol synchronization with an incoming message. The preamble field is TBD.

6.3.1.1.1 Multiple Preambles

Table 18—Preambles versus applications

Preamble	**Application**
P1	P2P
P2	VLAN
P3	IB
P4	VB

Ed. Note: It is assumed that preamble has more processing gain than the packet body - so what do we do when the SNR droops during packet body to prevent the BER from going bad? Also, is it necessary to tie applications to preamble number? Is the application list exhaustive?

6.3.1.2 SFD field

The SFD is an 8 bit field indicating the end of the synchronization (preamble) field and the start of the packet data. The SFD shall be formatted as illustrated in Figure 17.

The SFD is TBD.

6.3.1.3 Frame length field

The frame length field is 7 bits in length and specifies the total number of octets contained in the PSDU (i.e., PHY payload). It is a value between 0 and *aMaxPHYPacketSize* (see 6.x). Table 17 summarizes the type of payload versus the frame length value.

Table 19—Frame length values

Frame length values	Payload
0-4	Reserved
5	MPDU (Acknowledgment)
6-7	Reserved
8 to aMaxPHYPacketSize	MPDU

6.3.1.4 PSDU field

The PSDU field has a variable length and carries the data of the PHY packet. For all packet types of length five octets or greater than seven octets, the PSDU contains the MAC sublayer frame (i.e., MPDU).

6.3.1.5 Frame Check Sequence

Ed. Note - is this done here or at the MAC level? If done at the MAC level then this clause should be removed.

The frame shall be protected with a CCITT CRC-16 frame check sequence (FCS). The CCITT CRC-16 HCS shall be the ones complement of the remainder generated by the modulo-2 division of the protected frame by the polynomial

x16+x12+x5+1

The protected bits shall be processed in transmit order. All HCS calculations shall be made prior to data scrambling. A schematic of the processing is shown in Figure X.

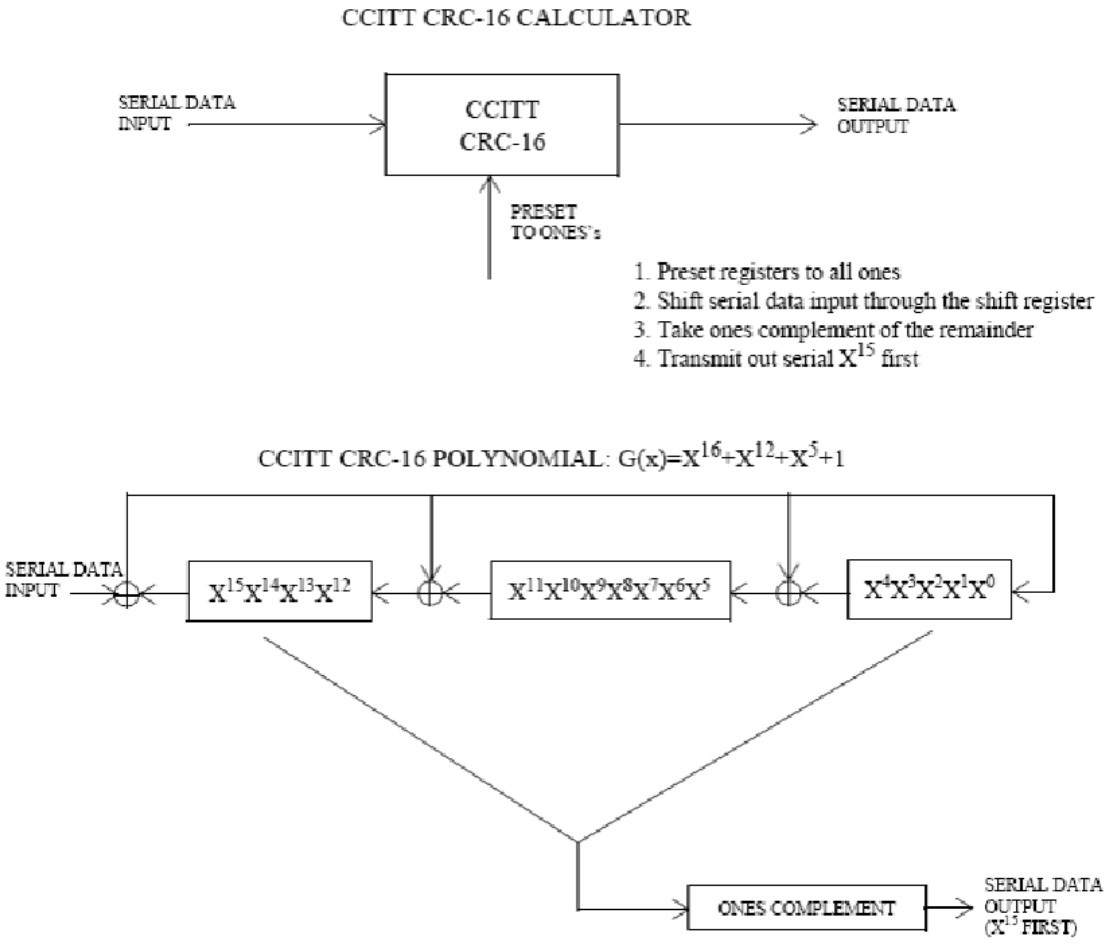


Figure 166—CCITT CRC-16 Implementation

As an example, consider the following 32-bit length sequence to be protected by the CRC-16

0101 0000 0000 0000 0000 0011 0000 0000

b0.....b31

The leftmost bit (b0) is transmitted first in time.

The ones complement HCS for this sequence would be the following:

0101 1011 0101 0111

b0.....b15

The leftmost bit (b0) is transmitted first in time. Bit b0 corresponds to X15 in the Figure 166.

An illustrative example of the CCITT CRC-16 HCS using the information from Figure 166 is shown in Figure X.

Data	CRC Registers msb lsb	
	1111111111111111	; Initialize preset to ones
0	1110111111011111	
1	1101111110111110	
0	1010111101011101	
1	0101111010111010	
0	1011110101110100	
0	0110101011001001	
0	1101010110010010	
0	1011101100000101	
0	0110011000101011	
0	1100110001010110	
0	1000100010001101	
0	0000000100111011	
0	0000001001110110	
0	0000010011101100	
0	0000100111011000	
0	0001001110110000	
0	0010011101100000	
0	0100111011000000	
0	1001110110000000	
0	0010101100100001	
0	0101011001000010	
0	1010110010000100	
1	0101100100001000	
1	1010001000110001	
0	0101010001000011	
0	1010100010000110	
0	0100000100101101	
0	1000001001011010	
0	0001010010010101	
0	0010100100101010	
0	0101001001010100	
0	1010010010101000	

Figure 167—Example of CRC calculation

The CRC-16 described in this subclause is the same one used in IEEE Std 802.11b™-1999 [B3].

6.4 PHY constants and PIB attributes

This subclause specifies the constants and attributes required by the PHY.

6.4.1 PHY constants

The constants that define the characteristics of the PHY are presented in Table 18. These constants are hardware dependent and cannot be changed during operation.

Table 20—PHY constants

Constant	Description	Value
aMaxPHYPacketSize	The maximum PSDU size (in octets) the PHY shall be able to receive.	127
aTurnaroundTime	RX-to-TX or TX-to-RX maximum turn-around time (see 6.x.x and 6.x.x)	12 symbol periods

6.4.2 PHY PIB attributes

The PHY PIB comprises the attributes required to manage the PHY of a device. Each of these attributes can be read or written using the PLME-GET.request and PLME-SET.request primitives, respectively. The attributes contained in the PHY PIB are presented in Table 19.

Table 21—PHY PIB attributes

Attribute	Identifier	Type	Range	Description
phyCurrentChannel	0x00	Integer	0-26	The RF channel to use for all following transmissions and receptions (see 6.x.x).
phyChannelsSupported	0x01	Bitmap	See description	The 5 most significant bits (MSBs) (b27,... , b31) of phyChannelsSupported shall be reserved and set to 0, and the 27 LSBs (b0, b1, ... b26) shall indicate the status (1=available, 0=unavailable) for each of the 27 valid channels (bk shall indicate the status of channel k as in 6.1.2).
phyCCAMode	0x02	Integer	1-3	The CCA mode (see 6.x.x)

6.5 PHY TYPE 1 specifications

The type 1 PHY is targetted towards applications requiring data rates operating at several tens of meters of range.

Table 22—PHY TYPE 1 Modes

	MCS identifier	Spreading parameters			Data parameters	
		Inner Code rate	Outer Code Rate	Modulation	Chip rate	Bit rate
TYPE 1	3	1/2 Rate **Manchester**	1/4 Rate	OOK	200 kcps	25 kbps
	4	1/2 Rate **Manchester**	1/2 Rate	OOK	200 kcps	50 kbps
	6	1/2 Rate **Manchester**	none	OOK	200 kcps	100 kbps

6.5.1 Data rate

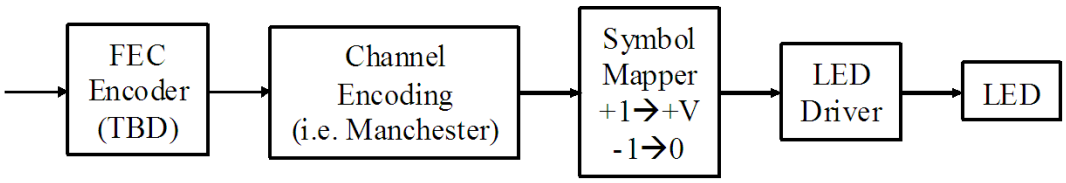
The LRP supports data rates of 25 kbps, 50 kbps and 100 kbps. The data rate of 25 kbps is used for link establishment. After the link is established other data rates may be utilized based upon the desired quality of service.

6.5.2 Modulation, forward error correction and DC balancing code

The modulation is OOK (or VPM if it is shown to be better) with an FEC outer code and Manchester Inner code. The 1/2 rate FEC outer code is TBD at this time. The 1/4 rate FEC outer code is TBD at this time.

6.5.2.1 Reference modulator diagram

A reference implementation is shown below.



6.5.2.2 Bit-to-symbol mapping

A logic level high applied to the light source shall result in a radiated high intensity. Likeswise, a logic level low applied to the light source shall result in a radiated low intensity.

6.5.2.3 Symbol-to-chip mapping

A bit value of logic high shall be represented by a high manchester chip followed by a low manchester chip.

6.5.2.3.1 Frame Flicker Compensation

TBD

6.6 PHY Type 2 specifications

The type 2 PHY is targetted towards applications requiring high data rates operating at several meters of range.

Table 23—PHY TYPE 2

	MCS identifier	Spreading parameters			Data parameters	
		Inner Code rate	Outer Code Rate	Modulation	Chip rate	Bit rate
Type 3	7	1/2 Rate **Manchester**	1/10 Rate	OOK	20 Mcps	1 Mbps
	8	1/2 Rate **Manchester**	1/2 Rate	OOK	20 Mcps	5 Mbps
	9	1/2 Rate **Manchester**	none	OOK	20 Mcps	10 Mbps

6.6.1 Data rate

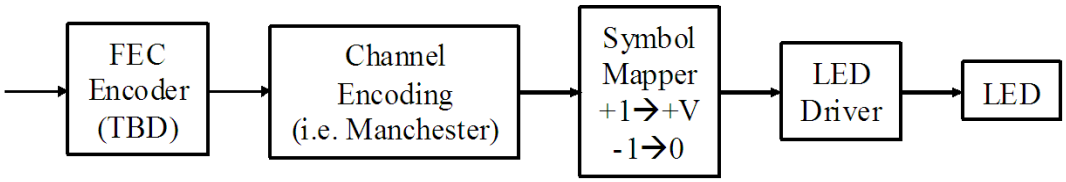
This PHY supports data rates of 1 Mbps, 5 Mbps and 10 Mbps using OOK modulation (or VPM if it is shown to be better). The OOK data rate of 5 Mbps is used for link establishment. After the link is established other data rates may be utilized based upon the desired quality of service.

6.6.2 Modulation, forward error correction and DC balancing code

The modulation is OOK with an FEC outer code and Manchester Inner code. The 1/2 rate FEC outer code is TBD at this time. The 1/10 rate FEC outer code is TBD at this time.

6.6.2.1 Reference modulator diagram

A reference implementation is shown below.



6.6.2.2 Bit-to-symbol mapping

A logic level high applied to the light source shall result in a radiated high intensity. Likeswise, a logic level low applied to the light source shall result in a radiated low intensity.

6.6.2.3 Symbol-to-chip mapping

A bit value of logic high shall be represented by a high manchester chip followed by a low manchester chip.

6.6.2.4 Frame Flicker Compensation

TBD

6.6.3 Frame Flicker Compensation

To prevent the LED from appearing “dimmer” during the packet frame transmission time, an idle pattern is sent between frames that has the same duty cycle as the modulated frame but the pulse repetition rate (exact repetition rate is TBD) is set much lower so as not to cause “in band” interference with any VLC modulation.

6.6.3.1 Filler Bit Pattern

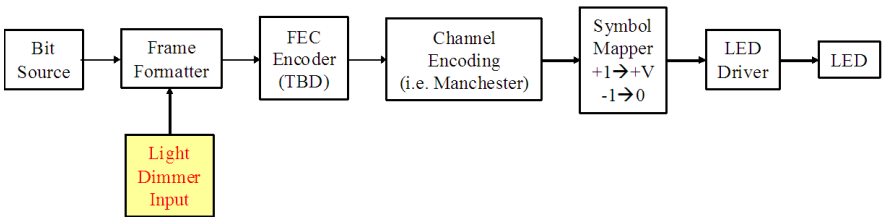
The bit pattern used for the filler sequence shall be a repetitive 1010. The ratio of ones and zeros shall be adjusted so as to make the required dimming as per clause X.X.X. The number of bits in the fillter pattern shall be an even number so as to maintain a DC balance on this sequence. The extinction ratio used for the transmission of the filler sequence shall be the same used for actual data packets.

6.6.3.2 Filler Bit Rate

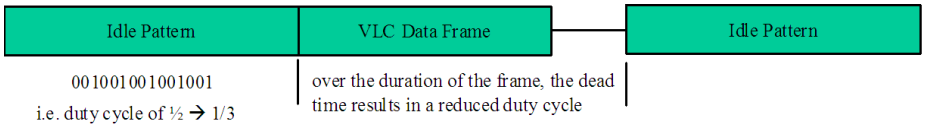
The filler sequence shall be sent with a bit rate that is approximately 200 bps.

6.6.4 Lighting Dimming

Light dimming is generally a MAC layer function; however, it does require that the PHY periodically turn off the light source. The details are shown in clause 7.X.X



OFF time is inserted into either the idle pattern or into the data frame, as shown below, to reduce the average intensity of the light.



6.6.4.1 Idle Pattern Extinction Ratio

The extinction ratio ideally should be 100% but practically it shall be as large as possible.

6.6.5 Device Discovery Sequence

For any of the operational modes an optional device discovery sequence can be sent to allow the receiver time to discover a VLC transmitting device. The device discovery sequence is part of the preamble as shown in clause 6.X.X and is sent at the discretion of the transmitting source. A transmitting device that chooses to transmit the discovery sequence implements this option by arbitrarily lengthening the preamble *[Ed. Note: is the discovery sequency length arbitrary? If not then how is the length determined and what should it be?]*. The device discovery sequence is terminated by the start of the SFD.

6.7 General radio specifications

The specifications in 6.7.1 through 6.7.9 apply to the PHY TYPES 1 and 2.

6.7.1 TX-to-RX turnaround time

The TX-to-RX turnaround time shall be less than *aTurnaroundTime* (see 6.4.1).

The TX-to-RX turnaround time shall be measured at the air interface from the trailing edge of the last transmitted symbol until the receiver is ready to begin the reception of the next PHY packet.

6.7.2 RX-to-TX turnaround time

The RX-to-TX turnaround time shall be less than aTurnaroundTime (see 6.4.1).

The RX-to-TX turnaround time shall be measured at the air interface from the trailing edge of the last chip (of the last symbol) of a received packet until the transmitter is ready to begin transmission of the resulting acknowledgment. Actual transmission start times are specified by the MAC sublayer (see 7.5.6.4.2).

6.7.3 Error-vector magnitude (EVM) definition

The modulation accuracy of an IEEE 802.15.7 transmitter is determined with an EVM measurement. In order to calculate the EVM measurement, a time record of N received complex chip values is captured. For each received complex chip, a decision is made about which complex chip value was transmitted. The ideal position of the chosen complex chip (the center of the decision box) is represented by the vector . The error vector is defined as the distance from this ideal position to the actual position of the received point (see Figure 22).

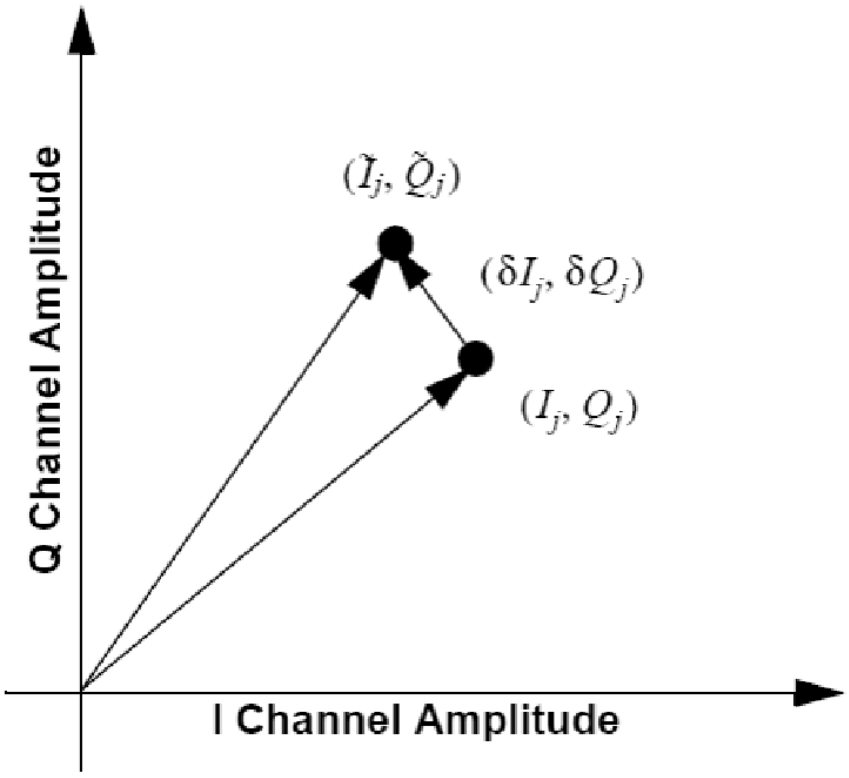


Figure 22—Error vector calculation

NEED TO FINISH THIS SECTION - ARE WE GOING TO SPECIFY AN ERROR VECTOR MAGNITUDE? ARE WE GOING TO DO THIS FOR BOTH OOK AND CCM?

6.7.3.1 EVM calculated values

An IEEE 802.15.7 transmitter shall have EVM values of less than TBD% when measured for 1000 chips. The error-vector measurement shall be made on baseband I and Q chips after recovery through a reference receiver system. The reference receiver shall perform carrier lock, symbol timing recovery, and amplitude adjustment while making the measurements.

6.7.4 Transmit frequency tolerance

The transmitted frequency tolerance shall be ± TBD ppm maximum.

6.7.5 Receiver maximum input level of desired signal

The receiver maximum input level is the maximum power level of the desired signal, in decibels relative to 1 mW, present at the input of the receiver for which the error rate criterion in 6.1.6 is met. An IEEE 802.15.7 receiver shall have a receiver maximum input level greater than or equal to –20 dBm.

6.7.6 LQI

The LQI measurement is a characterization of the strength and/or quality of a received packet. The measurement may be implemented using receiver ED, a signal-to-noise ratio estimation, or a combination of these methods. The use of the LQI result by the network or application layers is not specified in this standard.

The LQI measurement shall be performed for each received packet, and the result shall be reported to the MAC sublayer using PD-DATA.indication (see 6.2.1.3) as an integer ranging from 0x00 to 0xff. The minimum and maximum LQI values (0x00 and 0xff) should be associated with the lowest and highest quality IEEE 802.15.7 signals detectable by the receiver, and LQ values in between should be uniformly distributed between these two limits. At least eight unique values of LQ shall be used.

6.7.7 CCA

The IEEE 802.15.4 PHY may provide the capability to perform CCA according to at least one of the following three methods:

— CCA Mode 1: Energy above threshold. CCA may report a busy medium upon detecting any energy above the energy detect threshold.

— CCA Mode 2: Carrier sense only. CCA may report a busy medium only upon the detection of a signal with the modulation and spreading characteristics of IEEE 802.15.4. This signal may be above or below the energy detect threshold.

— CCA Mode 3: Carrier sense with energy above threshold. CCA may report a busy medium only upon the detection of a signal with the modulation and spreading characteristics of IEEE 802.15.4 with energy above the energy detect threshold.

For any of the CCA modes, if the PLME-CCA.request primitive (see 6.X.X.X) is received by the PHY during reception of a PPDU, CCA may report a busy medium. PPDU reception is considered to be in progress following detection of the SFD, and it remains in progress until the number of octets specified by the decoded PHR has been received.

A busy channel may be indicated by the PLME-CCA.confirm primitive (6.X.X.X) with a status of BUSY.

A clear channel may be indicated by the PLME-CCA.confirm primitive (6.X.X.X) with a status of IDLE.

The PHY PIB attribute phyCCAMode (see 6.X) may indicate the appropriate operation mode. The CCA parameters are subject to the following criteria:

a) The energy detect threshold may be at most TBD dB above the specified receiver sensitivity (see 6.X.X.X and 6.X.X.X).

b) The CCA detection time may be equal to TBD symbol periods.

7. MAC sublayer specification

This clause specifies the MAC sublayer of this standard. The MAC sublayer handles all access to the physical radio channel and is responsible for the following tasks:

- — Generating network beacons if the device is a coordinator
- — Synchronizing to network beacons
- — Supporting VAN association and disassociation

— Supporting device security

— Employing the random access mechanism for channel access

— Handling and maintaining the GTS mechanism

— Providing a reliable link between two peer MAC entities

Constants and attributes that are specified and maintained by the MAC sublayer are written in the text of this clause in italics. Constants have a general prefix of "a", e.g., *aBaseSlotDuration*, and are listed in Table 66 (see 7.5.1). Attributes have a general prefix of "mac", e.g., *macAckWaitDuration*, and are listed in Table 31 (see 7.5.2), while the security attributes are listed in Table 69 (see 7.7.1).

7.1 MAC sublayer service specification

The MAC sublayer provides an interface between the SSCS and the PHY. The MAC sublayer conceptually includes a management entity called the MLME. This entity provides the service interfaces through which layer management functions may be invoked. The MLME is also responsible for maintaining a database of managed objects pertaining to the MAC sublayer. This database is referred to as the MAC sublayer PIB.

Figure 15 depicts the components and interfaces of the MAC sublayer.

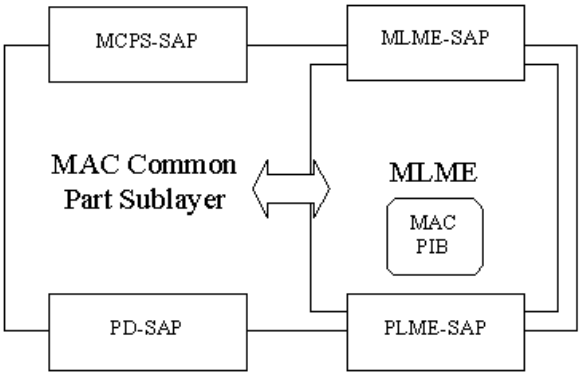


Figure 15—The MAC sublayer reference model

The MAC sublayer provides two services, accessed through two SAPs:

— The MAC data service, accessed through the MAC common part sublayer (MCPS) data SAP (MCPS-SAP), and

— The MAC management service, accessed through the MLME-SAP.

These two services provide the interface between the SSCS and the PHY, via the PD-SAP and PLME-SAP interfaces (see 6.2). In addition to these external interfaces, an implicit interface also exists between the MLME and the MCPS that allows the MLME to use the MAC data service.

7.1.1 MAC data service

The MCPS-SAP supports the transport of SSCS protocol data units (SPDUs) between peer SSCS entities. Table 24 lists the primitives supported by the MCPS-SAP. Primitives marked with a diamond (♦) are optional for an RFD. These primitives are discussed in the subclauses referenced in the table.

Table 24—MCPS-SAP primitives

MCPS-SAP primitive	Request	Confirm	Indication
MCPS-DATA	7.1.1.1	7.1.1.2	7.1.1.3
MCPS-PURGE	7.1.1.4♦	7.1.1.5♦	—

7.1.1.1 MCPS-DATA.request

The MCPS-DATA.request primitive requests the transfer of a data SPDU (i.e., MSDU) from a local SSCS entity to a single peer SSCS entity.

7.1.1.1.1 Semantics of the service primitive

The semantics of the MCPS-DATA.request primitive are as follows:

MCPS-DATA.request (
SrcAddrMode,
DstAddrMode,
DstVANId,
DstAddr,
msduLength,
msdu,
msduHandle,
TxOptions,
SecurityLevel,
KeyIdMode,
KeySource,
KeyIndex
DataRate
)

Table 25 specifies the parameters for the MCPS-DATA.request primitive.

Table 25—MCPS-DATA.request parameters

Name	Type	Valid range	Description
SrcAddrMode	Integer	0x00–0x03	The source addressing mode for this primitive and subsequent MPDU. This value can take one of the following values: 0x00 = no address (addressing fields omitted, see 7.2.1.1.8). 0x01 = reserved. 0x02 = 16-bit short address. 0x03 = 64-bit extended address.

Table 25—MCPS-DATA.request parameters *(continued)*

Name	Type	Valid range	Description
DstAddrMode	Integer	0x00–0x03	The destination addressing mode for this primitive and subsequent MPDU. This value can take one of the following values: 0x00 = no address (addressing fields omitted, see 7.2.1.1.6). 0x01 = reserved. 0x02 = 16-bit short address. 0x03 = 64-bit extended address.
DstVANId	Integer	0x0000–0xffff	The 16-bit VAN identifier of the entity to which the MSDU is being transferred.
DstAddr	Device address	As specified by the DstAddrMode parameter	The individual device address of the entity to which the MSDU is being transferred.
msduLength	Integer	≤ *aMaxMACPayloadSize*	The number of octets contained in the MSDU to be transmitted by the MAC sublayer entity.
msdu	Set of octets	—	The set of octets forming the MSDU to be transmitted by the MAC sublayer entity.
msduHandle	Integer	0x00–0xff	The handle associated with the MSDU to be transmitted by the MAC sublayer entity.
TxOptions	Bitmap	3-bit field	The 3 bits (b_0, b_1, b_2) indicate the transmission options for this MSDU. For b_0, 1 = acknowledged transmission, 0 = unacknowledged transmission. For b_1, 1 = GTS transmission, 0 = CAP transmission for a beacon-enabled VAN. For b_2, 1 = indirect transmission, 0 = direct transmission. For a nonbeacon-enabled VAN, bit b_1 should always be set to 0.
SecurityLevel	Integer	0x00–0x07	The security level to be used (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.
DataRate	Enumeration	TBD	The data rate of the PHY frame to be transmitted by the PHY entity.

7.1.1.1.2 Appropriate usage

The MCPS-DATA.request primitive is generated by a local SSCS entity when a data SPDU (i.e., MSDU) is to be transferred to a peer SSCS entity.

7.1.1.1.3 Effect on receipt

On receipt of the MCPS-DATA.request primitive, the MAC sublayer entity begins the transmission of the supplied MSDU.

The MAC sublayer builds an MPDU to transmit from the supplied arguments. The flags in the SrcAddrMode and DstAddrMode parameters correspond to the addressing subfields in the Frame Control field (see 7.2.1.1) and are used to construct both the Frame Control and addressing fields of the MHR. If both the SrcAddrMode and the DstAddrMode parameters are set to 0x00 (i.e., addressing fields omitted), the MAC sublayer will issue the MCPS-DATA.confirm primitive with a status of INVALID_ADDRESS.

If the msduLength parameter is greater than *aMaxMACSafePayloadSize*, the MAC sublayer will set the Frame Version subfield of the Frame Control field to one.

The TxOptions parameter indicates how the MAC sublayer data service transmits the supplied MSDU. If the TxOptions parameter specifies that an acknowledged transmission is required, the Acknowledgment Request subfield of the Frame Control field will be set to one (see 7.6.6.4).

If the TxOptions parameter specifies that a GTS transmission is required, the MAC sublayer will determine whether it has a valid GTS (for GTS usage rules, see 7.6.7.3). If a valid GTS could not be found, the MAC sublayer will issue the MCPS-DATA.confirm primitive with a status of INVALID_GTS. If a valid GTS was found, the MAC sublayer will defer, if necessary, until the GTS. If the TxOptions parameter specifies that a GTS transmission is not required, the MAC sublayer will transmit the MSDU using either slotted CSMA-CA in the CAP for a beacon-enabled VAN or unslotted random access for a nonbeacon-enabled VAN. Specifying a GTS transmission in the TxOptions parameter overrides an indirect transmission request.

If the TxOptions parameter specifies that an indirect transmission is required and this primitive is received by the MAC sublayer of a coordinator, the data frame is sent using indirect transmission, i.e., the data frame is added to the list of pending transactions stored on the coordinator and extracted at the discretion of the device concerned using the method described in 7.6.6.3. Transactions with a broadcast destination address will be transmitted using the mechanism described in 7.2.1.1.3. Transactions with a unicast destination address can then be extracted at the discretion of each device concerned using the method described in 7.6.6.3. If there is no capacity to store the transaction, the MAC sublayer will discard the MSDU and issue the MCPS-DATA.confirm primitive with a status of TRANSACTION_OVERFLOW. If there is capacity to store the transaction, the coordinator will add the information to the list. If the transaction is not handled within *macTransactionPersistenceTime*, the transaction information will be discarded and the MAC sublayer will issue the MCPS-DATA.confirm primitive with a status of TRANSACTION_EXPIRED. The transaction handling procedure is described in 7.6.5. If the TxOptions parameter specifies that an indirect transmission is required and if the device receiving this primitive is not a coordinator, the destination address is not present, or the TxOptions parameter also specifies a GTS transmission, the indirect transmission option will be ignored.

If the TxOptions parameter specifies that an indirect transmission is not required, the MAC sublayer will transmit the MSDU using random access either in the CAP for a beacon-enabled VAN or immediately for a nonbeacon-enabled VAN. If the TxOptions parameter specifies that a direct transmission is required and the MAC sublayer does not receive an acknowledgment from the recipient after *macMaxFrameRetries* retransmissions (see 7.6.6.4), it will discard the MSDU and issue the MCPS-DATA.confirm primitive with a status of NO_ACK.

If the SecurityLevel parameter is set to a valid value other than 0x00, indicating that security is required for this frame, the MAC sublayer will set the Security Enabled subfield of the Frame Control field to one. The MAC sublayer will perform outgoing processing on the frame based on the DstAddr, SecurityLevel, KeyIdMode, KeySource, and KeyIndex parameters, as described in 7.6.8.2.1. If any error occurs during

outgoing frame processing, the MAC sublayer will discard the frame and issue the MCPS-DATA.confirm primitive with the error status returned by outgoing frame processing.

If the requested transaction is too large to fit in the CAP or GTS, as appropriate, the MAC sublayer shall discard the frame and issue the MCPS-DATA.confirm primitive with a status of FRAME_TOO_LONG.

If the transmission uses random access and the random access algorithm failed due to adverse conditions on the channel, and the TxOptions parameter specifies that a direct transmission is required, the MAC sublayer will discard the MSDU and issue the MCPS-DATA.confirm primitive with a status of CHANNEL_ACCESS_FAILURE.

If the MAC sublayer receives the request while transmission is prohibited it shall delay transmission until transmission is permitted.

If the MPDU was successfully transmitted and, if requested, an acknowledgment was received, the MAC sublayer will issue the MCPS-DATA.confirm primitive with a status of SUCCESS.

If any parameter in the MCPS-DATA.request primitive is not supported or is out of range, the MAC sublayer will issue the MCPS-DATA.confirm primitive with a status of INVALID_PARAMETER.

7.1.1.2 MCPS-DATA.confirm

The MCPS-DATA.confirm primitive reports the results of a request to transfer a data SPDU (MSDU) from a local SSCS entity to a single peer SSCS entity.

7.1.1.2.1 Semantics of the service primitive

The semantics of the MCPS-DATA.confirm primitive are as follows:

```
MCPS-DATA.confirm       (
                        msduHandle,
                        status,
                        Timestamp
                        )
```

Table 26 specifies the parameters for the MCPS-DATA.confirm primitive.

Table 26—MCPS-DATA.confirm parameters

Name	Type	Valid range	Description
msduHandle	Integer	0x00–0xff	The handle associated with the MSDU being confirmed.

Table 26—MCPS-DATA.confirm parameters *(continued)*

Name	Type	Valid range	Description
status	Enumeration	SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, CHANNEL_ACCESS_FAILURE, INVALID_ADDRESS, INVALID_GTS, NO_ACK, COUNTER_ERROR, FRAME_TOO_LONG, UNAVAILABLE_KEY, UNSUPPORTED_SECURITY or INVALID_PARAMETER	The status of the last MSDU transmission.
Timestamp	Integer	0x000000–0xffffff	Optional. The time, in symbols, at which the data were transmitted (see 7.6.4.1). The value of this parameter will be considered valid only if the value of the status parameter is SUCCESS; if the status parameter is not equal to SUCCESS, the value of the Timestamp parameter shall not be used for any other purpose. The symbol boundary is described by *macSyncSymbolOffset* (see Table 31 in 7.5.1). This is a 24-bit value, and the precision of this value shall be a minimum of 20 bits, with the lowest 4 bits being the least significant.

7.1.1.2.2 When generated

The MCPS-DATA.confirm primitive is generated by the MAC sublayer entity in response to an MCPS-DATA.request primitive. The MCPS-DATA.confirm primitive returns a status of either SUCCESS, indicating that the request to transmit was successful, or the appropriate error code. The status values are fully described in 7.1.1.1.3 and subclauses referenced by 7.1.1.1.3.

7.1.1.2.3 Appropriate usage

On receipt of the MCPS-DATA.confirm primitive, the SSCS 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.

7.1.1.3 MCPS-DATA.indication

The MCPS-DATA.indication primitive indicates the transfer of a data SPDU (i.e., MSDU) from the MAC sublayer to the local SSCS entity.

7.1.1.3.1 Semantics of the service primitive

The semantics of the MCPS-DATA.indication primitive are as follows:

```
MCPS-DATA.indication    (
                        SrcAddrMode,
                        SrcVANId,
                        SrcAddr,
                        DstAddrMode,
                        DstVANId
                        DstAddr,
                        msduLength,
                        msdu,
                        mpduLinkQuality,
                        DSN,
                        Timestamp,
                        SecurityLevel,
                        KeyIdMode,
                        KeySource,
                        KeyIndex
                        DataRate,
                        )
```

Table 27 specifies the parameters for the MCPS-DATA.indication primitive.

Table 27—MCPS-DATA.indication parameters

Name	Type	Valid range	Description
SrcAddrMode	Integer	0x00–0x03	The source addressing mode for this primitive corresponding to the received MPDU. This value can take one of the following values: 0x00 = no address (addressing fields omitted). 0x01 = reserved. 0x02 = 16-bit short address. 0x03 = 64-bit extended address.
SrcVANId	Integer	0x0000–0xffff	The 16-bit VAN identifier of the entity from which the MSDU was received.
SrcAddr	Device address	As specified by the SrcAddrMode parameter	The individual device address of the entity from which the MSDU was received.

Table 27—MCPS-DATA.indication parameters *(continued)*

Name	Type	Valid range	Description
DstAddrMode	Integer	0x00–0x03	The destination addressing mode for this primitive corresponding to the received MPDU. This value can take one of the following values: 0x00 = no address (addressing fields omitted). 0x01 = reserved. 0x02 = 16-bit short device address. 0x03 = 64-bit extended device address.
DstVANId	Integer	0x0000–0xffff	The 16-bit VAN identifier of the entity to which the MSDU is being transferred.
DstAddr	Device address	As specified by the DstAddrMode parameter	The individual device address of the entity to which the MSDU is being transferred.
msduLength	Integer	≤ *aMaxMACFrame-Size*	The number of octets contained in the MSDU being indicated by the MAC sublayer entity.
msdu	Set of octets	—	The set of octets forming the MSDU being indicated by the MAC sublayer entity.
mpduLinkQuality	Integer	0x00–0xff	LQI value measured during reception of the MPDU. Lower values represent lower LQI (see 6.13.8).
DSN	Integer	0x00–0xff	The DSN of the received data frame.
Timestamp	Integer	0x000000–0xffffff	Optional. The time, in symbols, at which the data were received (see 7.6.4.1). The symbol boundary is described by *macSyncSymbolOffset* (see Table 31 in 7.5.1). This is a 24-bit value, and the precision of this value shall be a minimum of 20 bits, with the lowest 4 bits being the least significant.
SecurityLevel	Integer	0x00–0x07	The security level purportedly used by the received data frame (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key purportedly used by the originator of the received frame (see Table 77 in 7.7.2.2.2). This parameter is invalid if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key purportedly used by the originator of the received frame (see 7.7.2.4.1). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key purportedly used by the originator of the received frame (see 7.7.2.4.2). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
DataRate	Enumeration	0, 1, 2, 3, 4	The data rate of the PHY frame received by the PHY entity.

7.1.1.3.2 When generated

The MCPS-DATA.indication primitive is generated by the MAC sublayer and issued to the SSCS on receipt of a data frame at the local MAC sublayer entity that passes the appropriate message filtering operations as described in 7.6.6.2.

7.1.1.3.3 Appropriate usage

On receipt of the MCPS-DATA.indication primitive, the SSCS is notified of the arrival of data at the device. If the primitive is received while the device is in promiscuous mode, the parameters will be set as specified in 7.6.6.5.

7.1.1.4 MCPS-PURGE.request

The MCPS-PURGE.request primitive allows the next higher layer to purge an MSDU from the transaction queue.

This primitive is optional for an RFD.

7.1.1.4.1 Semantics of the service primitive

The semantics of the MCPS-PURGE.request primitive are as follows:

```
MCPS-PURGE.request          (
                            msduHandle
                            )
```

Table 28 specifies the parameters for the MCPS-PURGE.request primitive.

Table 28—MCPS-PURGE.request parameters

Name	Type	Valid range	Description
msduHandle	Integer	0x00–0xff	The handle of the MSDU to be purged from the transaction queue.

7.1.1.4.2 Appropriate usage

The MCPS-PURGE.request primitive is generated by the next higher layer whenever a MSDU is to be purged from the transaction queue.

7.1.1.4.3 Effect on receipt

On receipt of the MCPS-PURGE.request primitive, the MAC sublayer attempts to find in its transaction queue the MSDU indicated by the msduHandle parameter. If an MSDU has left the transaction queue, the handle will not be found, and the MSDU can no longer be purged. If an MSDU matching the given handle is found, the MSDU is discarded from the transaction queue, and the MAC sublayer issues the MCPS-PURGE.confirm primitive with a status of SUCCESS. If an MSDU matching the given handle is not found, the MAC sublayer issues the MCPS-PURGE.confirm primitive with a status of INVALID_HANDLE.

7.1.1.5 MCPS-PURGE.confirm

The MCPS-PURGE.confirm primitive allows the MAC sublayer to notify the next higher layer of the success of its request to purge an MSDU from the transaction queue.

This primitive is optional for an RFD.

7.1.1.5.1 Semantics of the service primitive

The semantics of the MCPS-PURGE.confirm primitive are as follows:

```
MCPS-PURGE.confirm      (
                        msduHandle,
                        status
                        )
```

Table 29 specifies the parameters for the MCPS-PURGE.confirm primitive.

Table 29—MCPS-PURGE.confirm parameters

Name	Type	Valid range	Description
msduHandle	Integer	0x00–0xff	The handle of the MSDU requested to be purge from the transaction queue.
status	Enumeration	SUCCESS or INVALID_HANDLE	The status of the request to be purged an MSDU from the transaction queue.

7.1.1.5.2 When generated

The MCPS-PURGE.confirm primitive is generated by the MAC sublayer entity in response to an MCPS-PURGE.request primitive. The MCPS-PURGE.confirm primitive returns a status of either SUCCESS, indicating that the purge request was successful, or INVALID_HANDLE, indicating an error. The status values are fully described in 7.1.1.4.3.

7.1.1.5.3 Appropriate usage

On receipt of the MCPS-PURGE.confirm primitive, the next higher layer is notified of the result of its request to purge an MSDU from the transaction queue. If the purge request was successful, the status parameter will be set to SUCCESS. Otherwise, the status parameter will indicate the error.

7.1.1.6 Data service message sequence chart

Figure 16 illustrates a sequence of messages necessary for a successful data transfer between two devices. Figure 72 and Figure 73 (see 7.8) also illustrate this, including the steps taken by the PHY.

7.1.2 MAC management service

The MLME-SAP allows the transport of management commands between the next higher layer and the MLME. Table 30 summarizes the primitives supported by the MLME through the MLME-SAP interface. Primitives marked with a diamond (♦) are optional for an RFD. Primitives marked with an asterisk (*) are

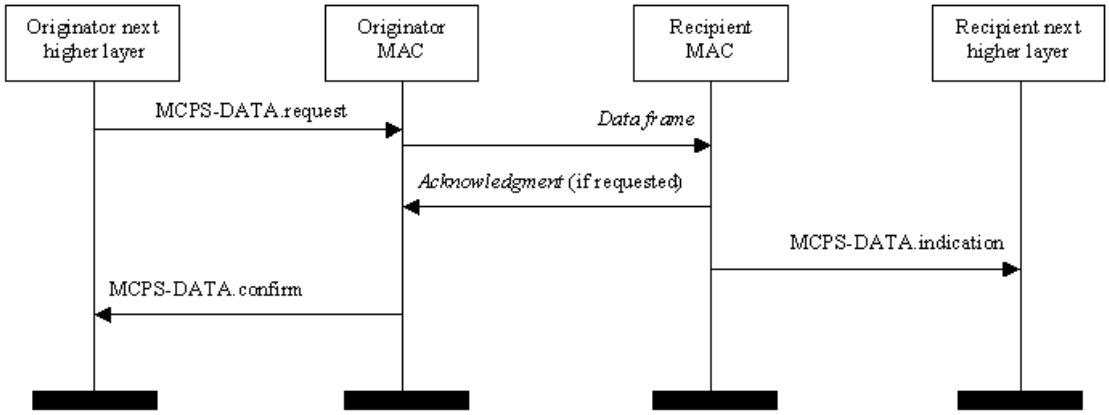


Figure 16—Message sequence chart describing the MAC data service

optional for both device types (i.e., RFD and FFD). The primitives are discussed in the subclauses referenced in the table.

Table 30—Summary of the primitives accessed through the MLME-SAP

Name	Request	Indication	Response	Confirm
MLME-ASSOCIATE	7.1.3.1	7.1.3.2♦	7.1.3.3♦	7.1.3.4
MLME-DISASSOCIATE	7.1.4.1	7.1.4.2		7.1.4.3
MLME-BEACON-NOTIFY		7.1.5.1		
MLME-GET	7.1.6.1			7.1.6.2
MLME-GTS	7.1.7.1*	7.1.7.3*		7.1.7.2*
MLME-ORPHAN		7.1.8.1♦	7.1.8.2♦	
MLME-RESET	7.1.9.1			7.1.9.2
MLME-RX-ENABLE	7.1.10.1*			7.1.10.2*
MLME-SCAN	7.1.11.1			7.1.11.2
MLME-COMM-STATUS		7.1.11.1		
MLME-SET	7.1.12.1			7.1.12.2
MLME-START	7.1.13.1♦			7.1.13.2♦
MLME-SYNC	7.1.14.1*			
MLME-SYNC-LOSS		7.1.14.2		
MLME-POLL	7.1.15.1			7.1.15.2

7.1.3 Association primitives

MLME-SAP association primitives define how a device becomes associated with a VAN.

All devices shall provide an interface for the request and confirm association primitives. The indication and response association primitives are optional for an RFD.

7.1.3.1 MLME-ASSOCIATE.request

The MLME-ASSOCIATE.request primitive allows a device to request an association with a coordinator.

7.1.3.1.1 Semantics of the service primitive

The semantics of the MLME-ASSOCIATE.request primitive are as follows:

```
MLME-ASSOCIATE.request      (
                            LogicalChannel,
                            CoordAddrMode,
                            CoordVANId,
                            CoordAddress,
                            CapabilityInformation,
                            SecurityLevel,
                            KeyIdMode,
                            KeySource,
                            KeyIndex
                            )
```

Table 31 specifies the parameters for the MLME-ASSOCIATE.request primitive.

Table 31—MLME-ASSOCIATE.request parameters

Name	Type	Valid range	Description
LogicalChannel	Integer	Selected from the available logical channels supported by the PHY (see 6.1.2).	The logical channel on which to attempt association.
CoordAddrMode	Integer	0x02–0x03	The coordinator addressing mode for this primitive and subsequent MPDU. This value can take one of the following values: 2=16-bit short address. 3=64-bit extended address.
CoordAddress	Device address	As specified by the CoordAddrMode parameter.	The address of the coordinator with which to associate.
CapabilityInformation	Bitmap	See 7.4.1.2	Specifies the operational capabilities of the associating device.
SecurityLevel	Integer	0x00–0x07	The security level to be used (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the SecurityLevel parameter is set to 0x00.

Table 31—MLME-ASSOCIATE.request parameters *(continued)*

Name	Type	Valid range	Description
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.

7.1.3.1.2 Appropriate usage

The MLME-ASSOCIATE.request primitive is generated by the next higher layer of an unassociated device and issued to its MLME to request an association with a VAN through a coordinator. If the device wishes to associate through a coordinator on a beacon-enabled VAN, the MLME may optionally track the beacon of that coordinator prior to issuing this primitive.

7.1.3.1.3 Effect on receipt

On receipt of the MLME-ASSOCIATE.request primitive, the MLME of an unassociated device first updates the appropriate PHY and MAC PIB attributes and then generates an association request command (see 7.4.1), as dictated by the association procedure described in 7.6.3.1.

The SecurityLevel parameter specifies the level of security to be applied to the association request command frame. Typically, the association request command should not be implemented using security. However, if the device requesting association shares a key with the coordinator, then security may be specified.

If the SecurityLevel parameter is set to a valid value other than 0x00, indicating that security is required for this frame, the MLME will set the Security Enabled subfield of the Frame Control field to one. The MAC sublayer will perform outgoing processing on the frame based on the CoordAddress, SecurityLevel, KeyIdMode, KeySource, and KeyIndex parameters, as described in 7.6.8.2.1. If any error occurs during outgoing frame processing, the MLME will discard the frame and issue the MLME-ASSOCIATE.confirm primitive with the error status returned by outgoing frame processing.

If the association request command cannot be sent to the coordinator due to the random access algorithm indicating a busy channel, the MLME will issue the MLME-ASSOCIATE.confirm primitive with a status of CHANNEL_ACCESS_FAILURE.

If the MLME successfully transmits an association request command, the MLME will expect an acknowledgment in return. If an acknowledgment is not received, the MLME will issue the MLME-ASSOCIATE.confirm primitive with a status of NO_ACK (see 7.6.6.4).

If the MLME of an unassociated device successfully receives an acknowledgment to its association request command, the MLME will wait for a response to the request (see 7.6.3.1). If the MLME of the device does not receive a response, it will issue the MLME-ASSOCIATE.confirm primitive with a status of NO_DATA.

If the MLME of the device extracts an association response command frame from the coordinator, it will then issue the MLME-ASSOCIATE.confirm primitive with a status equal to the contents of the Association Status field in the association response command (see 7.4.2.3).

On receipt of the association request command, the MLME of the coordinator issues the MLME-ASSOCIATE.indication primitive.

If any parameter in the MLME-ASSOCIATE.request primitive is either not supported or out of range, the MLME will issue the MLME-ASSOCIATE.confirm primitive with a status of INVALID_PARAMETER.

7.1.3.2 MLME-ASSOCIATE.indication

The MLME-ASSOCIATE.indication primitive is used to indicate the reception of an association request command.

7.1.3.2.1 Semantics of the service primitive

The semantics of the MLME-ASSOCIATE.indication primitive are as follows:

```
MLME-ASSOCIATE.indication    (
                             DeviceAddress,
                             CapabilityInformation,
                             SecurityLevel,
                             KeyIdMode,
                             KeySource,
                             KeyIndex
                             )
```

Table 32 specifies the parameters for the MLME-ASSOCIATE.indication primitive.

Table 32—MLME-ASSOCIATE.indication parameters

Name	Type	Valid range	Description
DeviceAddress	Device address	An extended 64-bit IEEE address	The address of the device requesting association.
CapabilityInformation	Bitmap	See 7.4.1.2	The operational capabilities of the device requesting association.
SecurityLevel	Integer	0x00–0x07	The security level purportedly used by the received MAC command frame (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key purportedly used by the originator of the received frame (see Table 77 in 7.7.2.2.2). This parameter is invalid if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key purportedly used by the originator of the received frame (see 7.7.2.4.1). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key purportedly used by the originator of the received frame (see 7.7.2.4.2). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.

7.1.3.2.2 When generated

The MLME-ASSOCIATE.indication primitive is generated by the MLME of the coordinator and issued to its next higher layer to indicate the reception of an association request command (see 7.4.1).

7.1.3.2.3 Appropriate usage

When the next higher layer of a coordinator receives the MLME-ASSOCIATE.indication primitive, the coordinator determines whether to accept or reject the unassociated device using an algorithm outside the scope of this standard. The next higher layer of the coordinator then issues the MLME-ASSOCIATE.response primitive to its MLME.

The association decision and the response should become available at the coordinator within a time of *macResponseWaitTime* (see 7.6.3.1). After this time, the device requesting association attempts to extract the association response command frame from the coordinator, using the method described in 7.6.6.3, in order to determine whether the association was successful.

7.1.3.3 MLME-ASSOCIATE.response

The MLME-ASSOCIATE.response primitive is used to initiate a response to an MLME-ASSOCIATE.indication primitive.

7.1.3.3.1 Semantics of the service primitive

The semantics of the MLME-ASSOCIATE.response primitive are as follows:

```
MLME-ASSOCIATE.response     (
                            DeviceAddress,
                            AssocShortAddress,
                            status,
                            SecurityLevel,
                            KeyIdMode,
                            KeySource,
                            KeyIndex
                            )
```

Table 33 specifies the parameters for the MLME-ASSOCIATE.response primitive.

7.1.3.3.2 Appropriate usage

The MLME-ASSOCIATE.response primitive is generated by the next higher layer of a coordinator and issued to its MLME in order to respond to the MLME-ASSOCIATE.indication primitive.

7.1.3.3.3 Effect on receipt

When the MLME of a coordinator receives the MLME-ASSOCIATE.response primitive, it generates an association response command (see 7.4.2). The command frame is sent to the device requesting association using indirect transmission, i.e., the command frame is added to the list of pending transactions stored on the coordinator and extracted at the discretion of the device concerned using the method described in 7.6.6.3.

If the SecurityLevel parameter is set to a valid value other than 0x00, indicating that security is required for this frame, the MLME will set the Security Enabled subfield of the Frame Control field to one. The MAC sublayer will perform outgoing processing on the frame based the DeviceAddress, SecurityLevel, KeyIdMode, KeySource, and KeyIndex parameters, as described in 7.6.8.2.1. If any error occurs during

Table 33—MLME-ASSOCIATE.response parameters

Name	Type	Valid range	Description
DeviceAddress	Device address	An extended 64 bit IEEE address	The address of the device requesting association.
AssocShortAddress	Integer	0x0000–0xffff	The 16-bit short device address allocated by the coordinator on successful association. This parameter is set to 0xffff if the association was unsuccessful.
status	Enumeration	See 7.4.2.3	The status of the association attempt.
SecurityLevel	Integer	0x00–0x07	The security level to be used (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.

outgoing frame processing, the MLME will discard the frame and issue the MLME-COMM-STATUS.indication primitive with the error status returned by outgoing frame processing.

Upon receipt of the MLME-ASSOCIATE.response primitive, the coordinator attempts to add the information contained in the primitive to its list of pending transactions. If there is no capacity to store the transaction, the MAC sublayer will discard the frame and issue the MLME-COMM-STATUS.indication primitive with a status of TRANSACTION_OVERFLOW. If there is capacity to store the transaction, the coordinator will add the information to the list. If the transaction is not handled within *macTransactionPersistenceTime*, the transaction information will be discarded and the MAC sublayer will issue the MLME-COMM-STATUS.indication primitive with a status of TRANSACTION_EXPIRED. The transaction handling procedure is described in 7.6.5.

If the frame was successfully transmitted and an acknowledgment was received, if requested, the MAC sublayer will issue the MLME-COMM-STATUS.indication primitive with a status of SUCCESS.

If any parameter in the MLME-ASSOCIATE.response primitive is not supported or is out of range, the MAC sublayer will issue the MLME-COMM-STATUS.indication primitive with a status of INVALID_PARAMETER.

7.1.3.4 MLME-ASSOCIATE.confirm

The MLME-ASSOCIATE.confirm primitive is used to inform the next higher layer of the initiating device whether its request to associate was successful or unsuccessful.

7.1.3.4.1 Semantics of the service primitive

The semantics of the MLME-ASSOCIATE.confirm primitive are as follows:

```
MLME-ASSOCIATE.confirm      (
                            AssocShortAddress,
                            status,
                            SecurityLevel,
                            KeyIdMode,
                            KeySource,
                            KeyIndex
                            )
```

Table 34 specifies the parameters for the MLME-ASSOCIATE.confirm primitive.

Table 34—MLME-ASSOCIATE.confirm parameters

Name	Type	Valid range	Description
AssocShortAddress	Integer	0x0000–0xffff	The short device address allocated by the coordinator on successful association. This parameter will be equal to 0xffff if the association attempt was unsuccessful.
status	Enumeration	The value of the Status field of the association response command (see 7.4.2.3), SUCCESS, CHANNEL_ACCESS_FAILURE, NO_ACK, NO_DATA, COUNTER_ERROR, FRAME_TOO_LONG, IMPROPER_KEY_TYPE, IMPROPER_SECURITY_LEVEL, SECURITY_ERROR, UNAVAILABLE_KEY, UNSUPPORTED_LEGACY, UNSUPPORTED_SECURITY INVALID_PARAMETER	The status of the association attempt.
SecurityLevel	Integer	0x00–0x07	If the primitive was generated following failed outgoing processing of an association request command: The security level to be used (see Table 76 in 7.7.2.2.1). If the primitive was generated following receipt of an association response command: The security level purportedly used by the received frame (see Table 76 in 7.7.2.2.1).

Table 34—MLME-ASSOCIATE.confirm parameters *(continued)*

Name	Type	Valid range	Description
KeyIdMode	Integer	0x00–0x03	If the primitive was generated following failed outgoing processing of an association request command: The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the SecurityLevel parameter is set to 0x00. If the primitive was generated following receipt of an association response command: The mode used to identify the key purportedly used by the originator of the received frame (see Table 77 in 7.7.2.2.2). This parameter is invalid if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	If the primitive was generated following failed outgoing processing of an association request command: The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00. If the primitive was generated following receipt of an association response command: The originator of the key purportedly used by the originator of the received frame (see 7.7.2.4.1). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
KeyIndex	Integer	0x01–0xff	If the primitive was generated following failed outgoing processing of an association request command: The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00. If the primitive was generated following receipt of an association response command: The index of the key purportedly used by the originator of the received frame (see 7.7.2.4.2). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.

7.1.3.4.2 When generated

The MLME-ASSOCIATE.confirm primitive is generated by the initiating MLME and issued to its next higher layer in response to an MLME-ASSOCIATE.request primitive. If the request was successful, the status parameter will indicate a successful association, as contained in the Status field of the association response command. Otherwise, the status parameter indicates either an error code from the received association response command or the appropriate error code from Table 34. The status values are fully described in 7.1.3.1.3 and subclauses referenced by 7.1.3.1.3.

7.1.3.4.3 Appropriate usage

On receipt of the MLME-ASSOCIATE.confirm primitive, the next higher layer of the initiating device is notified of the result of its request to associate with a coordinator. If the association attempt was successful, the status parameter will indicate a successful association, as contained in the Status field of the association response command, and the device will be provided with a 16-bit short address (see Table 68 in 7.6.3.1). If the association attempt was unsuccessful, the address will be equal to 0xffff, and the status parameter will indicate the error.

7.1.3.5 Association message sequence charts

Figure 17 illustrates a sequence of messages that may be used by a device that is not tracking the beacon of the coordinator (see 7.6.6.3) to successfully associate with a VAN. Figure 68 and Figure 69 (see 7.8) illustrate this same scenario, including steps taken by the PHY, for a device associating with a coordinator and for a coordinator allowing association by a device, respectively.

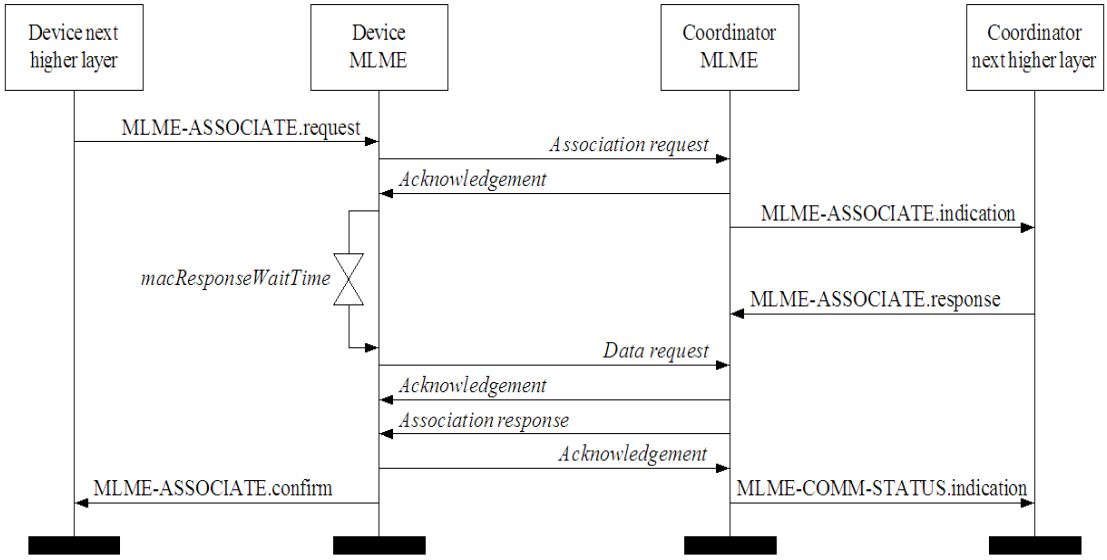


Figure 17—Message sequence chart for association

7.1.4 Disassociation primitives

The MLME-SAP disassociation primitives define how a device can disassociate from a VAN.

All devices shall provide an interface for these disassociation primitives.

7.1.4.1 MLME-DISASSOCIATE.request

The MLME-DISASSOCIATE.request primitive is used by an associated device to notify the coordinator of its intent to leave the VAN. It is also used by the coordinator to instruct an associated device to leave the VAN.

7.1.4.1.1 Semantics of the service primitive

The semantics of the MLME-DISASSOCIATE.request primitive are as follows:

```
MLME-DISASSOCIATE.request   (
                            DeviceAddrMode,
                            DeviceVANId,
                            DeviceAddress,
                            DisassociateReason,
                            TxIndirect,
                            SecurityLevel,
                            KeyIdMode,
                            KeySource,
                            KeyIndex
                            )
```

Table 35 specifies the parameters for the MLME-DISASSOCIATE.request primitive.

7.1.4.1.2 Appropriate usage

The MLME-DISASSOCIATE.request primitive is generated by the next higher layer of an associated device and issued to its MLME to request disassociation from the VAN. It is also generated by the next higher layer of the coordinator and issued to its MLME to instruct an associated device to leave the VAN.

7.1.4.1.3 Effect on receipt

On receipt of the MLME-DISASSOCIATE.request primitive, the MLME compares the DeviceVANId parameter with *macVANId*. If the DeviceVANId parameter is not equal to *macVANId*, the MLME issues the MLME-DISASSOCIATE.confirm primitive with a status of INVALID_PARAMETER. If the DeviceVANId parameter is equal to *macVANId*, the MLME evaluates the primitive address fields.

If the DeviceAddrMode parameter is equal to 0x02 and the DeviceAddress parameter is equal to *macCoordShortAddress* or if the DeviceAddrMode parameter is equal to 0x03 and the DeviceAddress parameter is equal to *macCoordExtendedAddress*, the TxIndirect parameter is ignored, and the MLME sends a disassociation notification command (see 7.4.3) to its coordinator in the CAP for a beacon-enabled VAN or immediately for a nonbeacon-enabled VAN.

If the DeviceAddrMode parameter is equal to 0x02 and the DeviceAddress parameter is not equal to *macCoordShortAddress* or if the DeviceAddrMode parameter is equal to 0x03 and the DeviceAddress parameter is not equal to *macCoordExtendedAddress*, and if this primitive was received by the MLME of a coordinator with the TxIndirect parameter set to TRUE, the disassociation notification command will be sent using indirect transmission, i.e., the command frame is added to the list of pending transactions stored on the coordinator and extracted at the discretion of the device concerned using the method described in 7.6.6.3.

If the DeviceAddrMode parameter is equal to 0x02 and the DeviceAddress parameter is not equal to *macCoordShortAddress* or if the DeviceAddrMode parameter is equal to 0x03 and the DeviceAddress parameter is not equal to *macCoordExtendedAddress*, and if this primitive was received by the MLME of a

Table 35—MLME-DISASSOCIATE.request parameters

Name	Type	Valid range	Description
DeviceAddrMode	Integer	0x02–0x03	The addressing mode of the device to which to send the disassociation notification command.
DeviceVANId	Integer	0x0000–0xffff	The VAN identifier of the device to which to send the disassociation notification command.
DeviceAddress	Device address	As specified by the DeviceAddrMode parameter.	The address of the device to which to send the disassociation notification command.
DisassociateReason	Integer	0x00–0xff	The reason for the disassociation (see 7.4.3.2).
TxIndirect	Boolean	TRUE or FALSE	TRUE if the disassociation notification command is to be sent indirectly.
SecurityLevel	Integer	0x00–0x07	The security level to be used (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.

coordinator with the TxIndirect parameter set to FALSE, the MLME sends a disassociation notification command to the device in the CAP for a beacon-enabled VAN or immediately for a nonbeacon-enabled VAN.

Otherwise, the MLME issues the MLME-DISASSOCIATE.confirm primitive with a status of INVALID_PARAMETER and does not generate a disassociation notification command.

If the disassociation notification command is to be sent using indirect transmission and there is no capacity to store the transaction, the MLME will discard the frame and issue the MLME-DISASSOCIATE.confirm primitive with a status of TRANSACTION_OVERFLOW. If there is capacity to store the transaction, the coordinator will add the information to the list. If the transaction is not handled within *macTransaction-PersistenceTime*, the transaction information will be discarded, and the MLME will issue the MLME-DISASSOCIATE.confirm with a status of TRANSACTION_EXPIRED. The transaction handling procedure is described in 7.6.5.

If the disassociation notification command cannot be sent due to a random access algorithm failure and this primitive was received either by the MLME of a coordinator with the TxIndirect parameter set to FALSE or by the MLME of a device, the MLME will issue the MLME-DISASSOCIATE.confirm primitive with a status of CHANNEL_ACCESS_FAILURE.

If the SecurityLevel parameter is set to a valid value other than 0x00, indicating that security is required for this frame, the MLME will set the Security Enabled subfield of the Frame Control field to one. The MAC sublayer will perform outgoing processing on the frame based on the DeviceAddress, SecurityLevel, KeyIdMode, KeySource, and KeyIndex parameters, as described in 7.6.8.2.1. If any error occurs during outgoing frame processing, the MLME will discard the frame and issue the MLME-DISASSOCIATE.confirm primitive with the error status returned by outgoing frame processing.

If the MLME successfully transmits a disassociation notification command, the MLME will expect an acknowledgment in return. If an acknowledgment is not received and this primitive was received either by the MLME of a coordinator with the TxIndirect parameter set to FALSE or by the MLME of a device, the MLME will issue the MLME-DISASSOCIATE.confirm primitive with a status of NO_ACK (see 7.6.6.4).

If the MLME successfully transmits a disassociation notification command and receives an acknowledgment in return, the MLME will issue the MLME-DISASSOCIATE.confirm primitive with a status of SUCCESS.

On receipt of the disassociation notification command, the MLME of the recipient issues the MLME-DISASSOCIATE.indication primitive.

If any parameter in the MLME-DISASSOCIATE.request primitive is not supported or is out of range, the MLME will issue the MLME-DISASSOCIATE.confirm primitive with a status of INVALID_PARAMETER.

7.1.4.2 MLME-DISASSOCIATE.indication

The MLME-DISASSOCIATE.indication primitive is used to indicate the reception of a disassociation notification command.

7.1.4.2.1 Semantics of the service primitive

The semantics of the MLME-DISASSOCIATE.indication primitive are as follows:

```
MLME-DISASSOCIATE.indication    (
                                DeviceAddress,
                                DisassociateReason,
                                SecurityLevel,
                                KeyIdMode,
                                KeySource,
                                KeyIndex
                                )
```

Table 36 specifies the parameters for the MLME-DISASSOCIATE.indication primitive.

7.1.4.2.2 When generated

The MLME-DISASSOCIATE.indication primitive is generated by the MLME and issued to its next higher layer on receipt of a disassociation notification command.

7.1.4.2.3 Appropriate usage

The next higher layer is notified of the reason for the disassociation.

Table 36—MLME-DISASSOCIATE.indication parameters

Name	Type	Valid range	Description
DeviceAddress	Device address	An extended 64-bit IEEE address	The address of the device requesting disassociation.
DisassociateReason	Integer	0x00–0xff	The reason for the disassociation (see 7.4.3.2).
SecurityLevel	Integer	0x00–0x07	The security level purportedly used by the received MAC command frame (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key purportedly used by the originator of the received frame (see Table 77 in 7.7.2.2.2). This parameter is invalid if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key purportedly used by the originator of the received frame (see 7.7.2.4.1). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key purportedly used by the originator of the received frame (see 7.7.2.4.2). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.

7.1.4.3 MLME-DISASSOCIATE.confirm

The MLME-DISASSOCIATE.confirm primitive reports the results of an MLME-DISASSOCIATE.request primitive.

7.1.4.3.1 Semantics of the service primitive

The semantics of the MLME-DISASSOCIATE.confirm primitive are as follows:

```
MLME-DISASSOCIATE.confirm       (
                                status,
                                DeviceAddrMode,
                                DevicePANId,
                                DeviceAddress
                                )
```

Table 37 specifies the parameters for the MLME-DISASSOCIATE.confirm primitive.

7.1.4.3.2 When generated

The MLME-DISASSOCIATE.confirm primitive is generated by the initiating MLME and issued to its next higher layer in response to an MLME-DISASSOCIATE.request primitive. This primitive returns a status of either SUCCESS, indicating that the disassociation request was successful, or the appropriate error code. The status values are fully described in 7.1.4.1.3 and subclauses referenced by 7.1.4.1.3.

Table 37—MLME-DISASSOCIATE.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, NO_ACK, CHANNEL_ACCESS_FAILURE, COUNTER_ERROR, FRAME_TOO_LONG, UNAVAILABLE_KEY, UNSUPPORTED_SECURITY or INVALID_PARAMETER	The status of the disassociation attempt.
DeviceAddrMode	Integer	0x02–0x03	The addressing mode of the device that has either requested disassociation or been instructed to disassociate by its coordinator.
DeviceVANId	Integer	0x0000–0xffff	The VAN identifier of the device that has either requested disassociation or been instructed to disassociate by its coordinator.
DeviceAddress	Device address	As specified by the DeviceAddrMode parameter.	The address of the device that has either requested disassociation or been instructed to disassociate by its coordinator.

7.1.4.3.3 Appropriate usage

On receipt of the MLME-DISASSOCIATE.confirm primitive, the next higher layer of the initiating device is notified of the result of the disassociation attempt. If the disassociation attempt was successful, the status parameter will be set to SUCCESS. Otherwise, the status parameter indicates the error.

7.1.4.4 Disassociation message sequence charts

The request to disassociate may originate either from a device or from the coordinator through which the device has associated. Figure 18 illustrates the sequence of messages necessary for a device to successfully disassociate itself from the VAN.

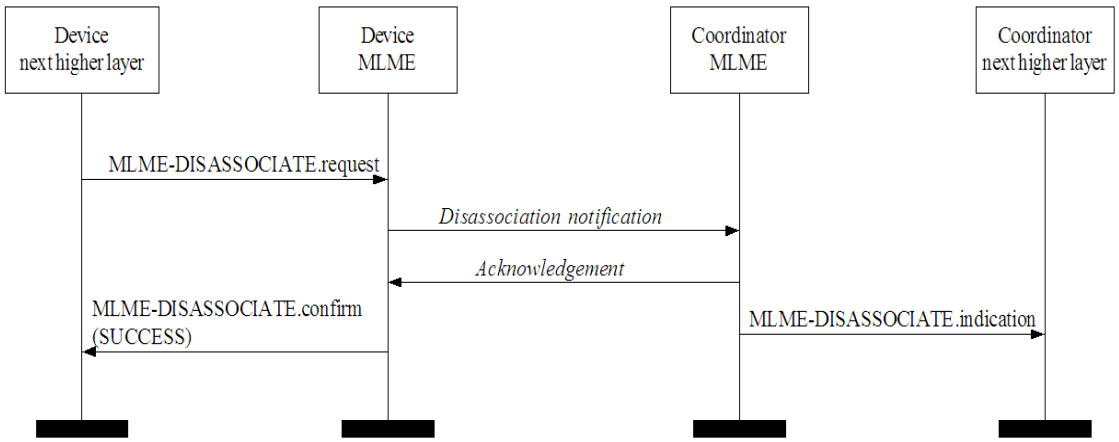


Figure 18—Message sequence chart for disassociation initiated by a device

Figure 19 illustrates the sequence necessary for a coordinator in a beacon-enabled VAN to successfully disassociate a device from its VAN using indirect transmission.

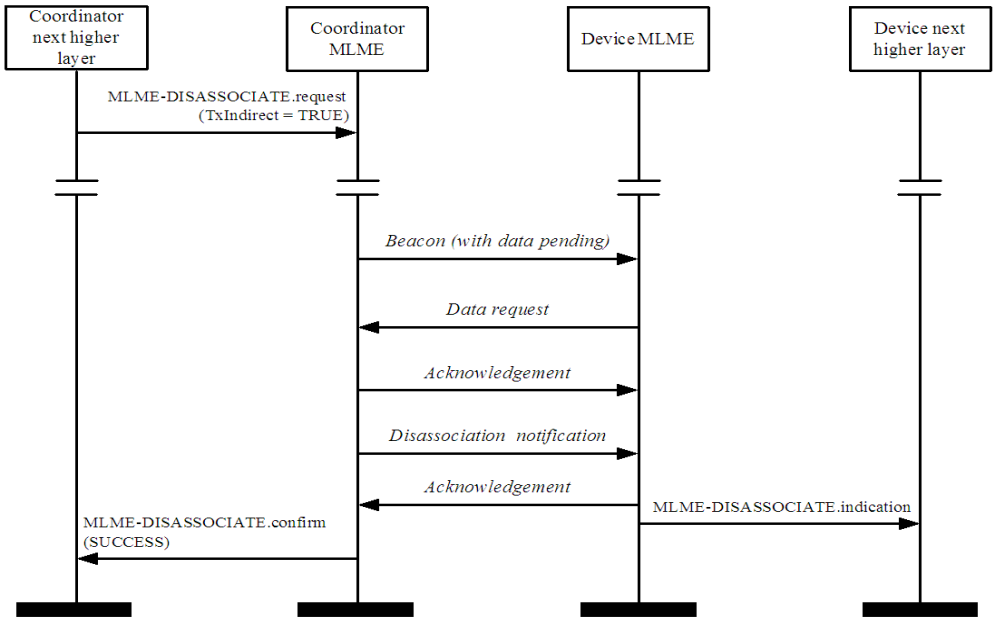


Figure 19—Message sequence chart for disassociation initiated by a coordinator, using indirect transmission, in a beacon-enabled VAN

7.1.5 Beacon notification primitive

The MLME-SAP beacon notification primitive defines how a device may be notified when a beacon is received during normal operating conditions.

All devices shall provide an interface for the beacon notification primitive.

7.1.5.1 MLME-BEACON-NOTIFY.indication

The MLME-BEACON-NOTIFY.indication primitive is used to send parameters contained within a beacon frame received by the MAC sublayer to the next higher layer. The primitive also sends a measure of the LQI and the time the beacon frame was received.

7.1.5.1.1 Semantics of the service primitive

The semantics of the MLME-BEACON-NOTIFY.indication primitive are as follows:

```
MLME-BEACON-NOTIFY.indication (
                              BSN,
                              VANDescriptor,
                              PendAddrSpec,
                              AddrList,
                              sduLength,
                              sdu
                              )
```

Table 38 specifies the parameters for the MLME-BEACON-NOTIFY.indication primitive.

Table 38—MLME-BEACON-NOTIFY.indication parameters

Name	Type	Valid range	Description
BSN	Integer	0x00–0xff	The beacon sequence number.
VANDescriptor	VANDescriptor value	See Table 39	The VANDescriptor for the received beacon.
PendAddrSpec	Bitmap	See 7.2.2.1.6	The beacon pending address specification.
AddrList	List of device addresses	—	The list of addresses of the devices for which the beacon source has data.
sduLength	Integer	0 – *aMaxBeaconPayloadLength*	The number of octets contained in the beacon payload of the beacon frame received by the MAC sublayer.
sdu	Set of octets	—	The set of octets comprising the beacon payload to be transferred from the MAC sublayer entity to the next higher layer.

Table 39 describes the elements of the VANDescriptor type.

Table 39—Elements of VANDescriptor

Name	Type	Valid range	Description
CoordAddrMode	Integer	0x02–0x03	The coordinator addressing mode corresponding to the received beacon frame. This value can take one of the following values: 2 = 16-bit short address. 3 = 64-bit extended address.
CoordVANId	Integer	0x0000–0xffff	The VAN identifier of the coordinator as specified in the received beacon frame.
CoordAddress	Device address	As specified by the CoordAddrMode parameter	The address of the coordinator as specified in the received beacon frame.
LogicalChannel	Integer	Selected from the available logical channels supported by the PHY (see 6.1.2)	The current logical channel occupied by the network.
SuperframeSpec	Bitmap	See 7.2.2.1.2	The superframe specification as specified in the received beacon frame.
GTSPermit	Boolean	TRUE or FALSE	TRUE if the beacon is from the VAN coordinator that is accepting GTS requests.
LinkQuality	Integer	0x00–0xff	The LQI at which the network beacon was received. Lower values represent lower LQI (see 6.13.8).
TimeStamp	Integer	0x000000–0xffffff	The time at which the beacon frame was received, in symbols. This value is equal to the timestamp taken when the beacon frame was received, as described in 7.6.4.1. This is a 24-bit value, and the precision of this value shall be a minimum of 20 bits, with the lowest 4 bits being the least significant.
SecurityFailure	Enumer-ation	SUCCESS, COUNTER_ERROR, IMPROPER_KEY_TYPE, IMPROPER_SECURITY_LEVEL, SECURITY_ERROR, UNAVAILABLE_KEY, UNSUPPORTED_LEGACY, UNSUPPORTED_SECURITY	SUCCESS if there was no error in the security processing of the frame. One of the other status codes indicating an error in the security processing otherwise (see 7.6.8.2.3).
SecurityLevel	Integer	0x00–0x07	The security level purportedly used by the received beacon frame (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key purportedly used by the originator of the received frame (see Table 77 in 7.7.2.2.2). This parameter is invalid if the SecurityLevel parameter is set to 0x00.

Table 39—Elements of VANDescriptor *(continued)*

Name	Type	Valid range	Description
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key purportedly used by the originator of the received frame (see 7.7.2.4.1). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key purportedly used by the originator of the received frame (see 7.7.2.4.2). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.

7.1.5.1.2 When generated

The MLME-BEACON-NOTIFY.indication primitive is generated by the MLME and issued to its next higher layer upon receipt of a beacon frame either when *macAutoRequest* is set to FALSE or when the beacon frame contains one or more octets of payload.

7.1.5.1.3 Appropriate usage

On receipt of the MLME-BEACON-NOTIFY.indication primitive, the next higher layer is notified of the arrival of a beacon frame at the MAC sublayer.

7.1.6 Primitives for reading PIB attributes

The MLME-SAP get primitives define how to read values from the PIB.

All devices shall provide an interface for these get primitives.

7.1.6.1 MLME-GET.request

The MLME-GET.request primitive requests information about a given PIB attribute.

7.1.6.1.1 Semantics of the service primitive

The semantics of the MLME-GET.request primitive are as follows:

```
MLME-GET.request        (
                        PIBAttribute,
                        PIBAttributeIndex
                        )
```

Table 40 specifies the parameters for the MLME-GET.request primitive.

7.1.6.1.2 Appropriate usage

The MLME-GET.request primitive is generated by the next higher layer and issued to its MLME to obtain information from the PIB.

Table 40—MLME-GET.request parameters

Name	Type	Valid range	Description
PIBAttribute	Integer	See Table 31 and Table 69	The identifier of the PIB attribute to read.
PIBAttributeIndex	Integer	Attribute specific; see Table 69	The index within the table of the specified PIB attribute to read. This parameter is valid only for MAC PIB attributes that are tables; it is ignored when accessing PHY PIB attributes.

7.1.6.1.3 Effect on receipt

On receipt of the MLME-GET.request primitive, the MLME checks to see if the PIB attribute is a MAC PIB attribute or PHY PIB attribute. If the requested attribute is a MAC attribute, the MLME attempts to retrieve the requested MAC PIB attribute from its database. If the identifier of the PIB attribute is not found in the database, the MLME will issue the MLME-GET.confirm primitive with a status of UNSUPPORTED_ATTRIBUTE. If the PIBAttributeIndex parameter specifies an index for a table that is out of range, the MLME will issue the MLME-GET.confirm primitive with a status of INVALID_INDEX. If the requested MAC PIB attribute is successfully retrieved, the MLME will issue the MLME-GET.confirm primitive with a status of SUCCESS.

If the requested attribute is a PHY PIB attribute, the request is passed to the PHY by issuing the PLME-GET.request primitive. Once the MLME receives the PLME-GET.confirm primitive, it will translate the received status value because the status values used by the PHY are not the same as those used by the MLME (e.g., the status values for SUCCESS are 0x00 and 0x07 in the MAC and PHY enumeration tables, respectively). Following the translation, the MLME will issue the MLME-GET.confirm primitive to the next higher layer with the status parameter resulting from the translation and the PIBAttribute and PIBAttributeValue parameters equal to those returned by the PLME primitive.

7.1.6.2 MLME-GET.confirm

The MLME-GET.confirm primitive reports the results of an information request from the PIB.

7.1.6.2.1 Semantics of the service primitive

The semantics of the MLME-GET.confirm primitive are as follows:

```
MLME-GET.confirm        (
                        status,
                        PIBAttribute,
                        PIBAttributeIndex,
                        PIBAttributeValue
                        )
```

Table 41 specifies the parameters for the MLME-GET.confirm primitive.

7.1.6.2.2 When generated

The MLME-GET.confirm primitive is generated by the MLME and issued to its next higher layer in response to an MLME-GET.request primitive. This primitive returns a status of either SUCCESS, indicating that the request to read a PIB attribute was successful, or an error code of UNSUPPORTED_ATTRIBUTE.

Table 41—MLME-GET.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	SUCCESS, UNSUPPORTED_ATTRIBUTE or INVALID_INDEX	The result of the request for PIB attribute information.
PIBAttribute	Integer	See Table 31 and Table 69	The identifier of the PIB attribute that was read.
PIBAttributeIndex	Integer	Attribute specific; see Table 69	The index within the table or array of the specified PIB attribute to read. This parameter is valid only for MAC PIB attributes that are tables or arrays; it is ignored when accessing PHY PIB attributes.
PIBAttributeValue	Various	Attribute specific; see Table 31 and Table 69	The value of the indicated PIB attribute that was read. This parameter has zero length when the status parameter is set to UNSUPPORTED_ATTRIBUTE.

When an error code of UNSUPPORTED_ATTRIBUTE is returned, the PIBAttribute value parameter will be set to length zero. The status values are fully described in 7.1.6.1.3.

7.1.6.2.3 Appropriate usage

On receipt of the MLME-GET.confirm primitive, the next higher layer is notified of the results of its request to read a PIB attribute. If the request to read a PIB attribute was successful, the status parameter will be set to SUCCESS. Otherwise, the status parameter indicates the error.

7.1.7 GTS management primitives

The MLME-SAP GTS management primitives define how GTSs are requested and maintained. A device wishing to use these primitives and GTSs in general will already be tracking the beacons of its VAN coordinator.

These GTS management primitives are optional.

7.1.7.1 MLME-GTS.request

The MLME-GTS.request primitive allows a device to send a request to the VAN coordinator to allocate a new GTS or to deallocate an existing GTS. This primitive is also used by the VAN coordinator to initiate a GTS deallocation.

7.1.7.1.1 Semantics of the service primitive

The semantics of the MLME-GTS.request primitive are as follows:

```
MLME-GTS.request        (
                        GTSCharacteristics,
                        SecurityLevel,
                        KeyIdMode,
                        KeySource,
                        KeyIndex
                        )
```

Table 42 specifies the parameters for the MLME-GTS.request primitive.

Table 42—MLME-GTS.request parameters

Name	Type	Valid range	Description
GTSCharacteristics	GTS characteristics	See 7.4.10.2	The characteristics of the GTS request, including whether the request is for the allocation of a new GTS or the deallocation of an existing GTS.
SecurityLevel	Integer	0x00–0x07	The security level to be used (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.

7.1.7.1.2 Appropriate usage

The MLME-GTS.request primitive is generated by the next higher layer of a device and issued to its MLME to request the allocation of a new GTS or to request the deallocation of an existing GTS. It is also generated by the next higher layer of the VAN coordinator and issued to its MLME to request the deallocation of an existing GTS.

7.1.7.1.3 Effect on receipt

On receipt of the MLME-GTS.request primitive by a device, the MLME of a device attempts to generate a GTS request command (see 7.4.10) with the information contained in this primitive and, if successful, sends it to the VAN coordinator.

If *macShortAddress* is equal to 0xfffe or 0xffff, the device is not permitted to request a GTS. In this case, the MLME issues the MLME-GTS.confirm primitive containing a status of NO_SHORT_ADDRESS.

If the SecurityLevel parameter is set to a valid value other than 0x00, indicating that security is required for this frame, the MLME will set the Security Enabled subfield of the Frame Control field to one. The MAC sublayer will perform outgoing processing on the frame based on *macCoordExtendedAddress* and the SecurityLevel, KeyIdMode, KeySource, and KeyIndex parameters, as described in 7.6.8.2.1. If any error

occurs during outgoing frame processing, the MLME will discard the frame and issue the MLME-GTS.confirm primitive with the error status returned by outgoing frame processing.

If the GTS request command cannot be sent due to a random access algorithm failure, the MLME will issue the MLME-GTS.confirm primitive with a status of CHANNEL_ACCESS_FAILURE.

If the MLME successfully transmits a GTS request command, the MLME will expect an acknowledgment in return. If an acknowledgment is not received, the MLME will issue the MLME-GTS.confirm primitive with a status of NO_ACK (see 7.6.6.4).

If a GTS is being allocated (see 7.6.7.2) and the request has been acknowledged, the device will wait for a confirmation via a GTS descriptor specified in a beacon frame from its VAN coordinator. If the MLME of the VAN coordinator can allocate the requested GTS, it will issue the MLME-GTS.indication primitive with the characteristics of the allocated GTS and generate a GTS descriptor with the characteristics of the allocated GTS and the 16-bit short address of the requesting device. If the MLME of the VAN coordinator cannot allocate the requested GTS, it will generate a GTS descriptor with a start slot of zero and the short address of the requesting device.

If the device receives a beacon frame from its VAN coordinator with a GTS descriptor containing a 16-bit short address that matches *macShortAddress*, the device will process the descriptor. If no descriptor for that device is received, the MLME will issue the MLME-GTS.confirm primitive with a status of NO_DATA.

If a descriptor is received that matches the characteristics requested (indicating that the VAN coordinator has approved the GTS allocation request), the MLME of the device will issue the MLME-GTS.confirm primitive with a status of SUCCESS and a GTSCharacteristics parameter with a characteristics type equal to one, indicating a GTS allocation.

If the descriptor is received with a start slot of zero (indicating that the VAN coordinator has denied the GTS allocation request), the device requesting the GTS issues the MLME-GTS.confirm primitive with a status of DENIED, indicating that the GTSCharacteristics parameter is to be ignored.

If a GTS is being deallocated (see 7.6.7.4) at the request of a device and the request has been acknowledged by the VAN coordinator, the device will issue the MLME-GTS.confirm primitive with a status of SUCCESS and a GTSCharacteristics parameter with a characteristics type equal to zero, indicating a GTS deallocation. On receipt of a GTS request command with a request type indicating a GTS deallocation, the VAN coordinator will acknowledge the frame and deallocates the GTS. The MLME of the VAN coordinator will then issue the MLME-GTS.indication primitive with the appropriate GTS characteristics. If the VAN coordinator does not receive the deallocation request, countermeasures can be applied by the VAN coordinator to ensure consistency is maintained (see 7.6.7.6).

If the MLME of the VAN coordinator receives an MLME-GTS.request primitive indicating deallocation, the VAN coordinator will deallocate the GTS and issue the MLME-GTS.confirm primitive with a status of SUCCESS and a GTSCharacteristics parameter with a characteristics type equal to zero.

If the device receives a beacon frame from its VAN coordinator with a GTS descriptor containing a short address that matches *macShortAddress* and a start slot equal to zero, the device immediately stops using the GTS. The MLME of the device then notifies the next higher layer of the deallocation by issuing the MLME-GTS.indication primitive with a GTSCharacteristics parameter containing the characteristics of the deallocated GTS.

If any parameter in the MLME-GTS.request primitive is not supported or is out of range, the MLME will issue the MLME-GTS.confirm primitive with a status of INVALID_PARAMETER.

7.1.7.2 MLME-GTS.confirm

The MLME-GTS.confirm primitive reports the results of a request to allocate a new GTS or deallocate an existing GTS.

7.1.7.2.1 Semantics of the service primitive

The semantics of the MLME-GTS.confirm primitive are as follows:

```
MLME-GTS.confirm        (
                        GTSCharacteristics,
                        status
                        )
```

Table 43 specifies the parameters for the MLME-GTS.confirm primitive.

Table 43—MLME-GTS.confirm parameters

Name	Type	Valid range	Description
GTSCharacteristics	GTS characteristics	See 7.4.10.2	The characteristics of the GTS.
status	Enumeration	SUCCESS, DENIED, NO_SHORT_ADDRESS, CHANNEL_ACCESS_FAILURE, NO_ACK, NO_DATA, COUNTER_ERROR, FRAME_TOO_LONG, UNAVAILABLE_KEY, UNSUPPORTED_SECURITY or INVALID_PARAMETER.	The status of the GTS request.

7.1.7.2.2 When generated

The MLME-GTS.confirm primitive is generated by the MLME and issued to its next higher layer in response to a previously issued MLME-GTS.request primitive.

If the request to allocate or deallocate a GTS was successful, this primitive will return a status of SUCCESS and the Characteristics Type field of the GTSCharacteristics parameter will have the value of one or zero, respectively. Otherwise, the status parameter will indicate the appropriate error code. The reasons for these status values are fully described in 7.1.7.1.3 and subclauses referenced by 7.1.7.1.3.

7.1.7.2.3 Appropriate usage

On receipt of the MLME-GTS.confirm primitive the next higher layer is notified of the result of its request to allocate or deallocate a GTS. If the request was successful, the status parameter will indicate a successful GTS operation. Otherwise, the status parameter will indicate the error.

7.1.7.3 MLME-GTS.indication

The MLME-GTS.indication primitive indicates that a GTS has been allocated or that a previously allocated GTS has been deallocated.

7.1.7.3.1 Semantics of the service primitive

The semantics of the MLME-GTS.indication primitive are as follows:

```
MLME-GTS.indication     (
                        DeviceAddress,
                        GTSCharacteristics,
                        SecurityLevel,
                        KeyIdMode,
                        KeySource,
                        KeyIndex
                        )
```

Table 44 specifies the parameters for the MLME-GTS.indication primitive.

Table 44—MLME-GTS.indication parameters

Name	Type	Valid range	Description
DeviceAddress	Device address	0x0000–0xfffd	The 16-bit short address of the device that has been allocated or deallocated a GTS.
GTSCharacteristics	GTS characteristics	See 7.4.10.2	The characteristics of the GTS.
SecurityLevel	Integer	0x00–0x07	If the primitive was generated when a GTS deallocation is initiated by the VAN coordinator itself, the security level to be used is set to 0x00. If the primitive was generated whenever a GTS is allocated or deallocated following the reception of a GTS request command: The security level purportedly used by the received MAC command frame (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	If the primitive was generated when a GTS deallocation is initiated by the VAN coordinator itself, this parameter is ignored. If the primitive was generated whenever a GTS is allocated or deallocated following the reception of a GTS request command: The mode used to identify the key purportedly used by the originator of the received frame (see Table 77 in 7.7.2.2.2). This parameter is invalid if the SecurityLevel parameter is set to 0x00.

Table 44—MLME-GTS.indication parameters *(continued)*

Name	Type	Valid range	Description
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	If the primitive was generated when a GTS deallocation is initiated by the VAN coordinator itself, this parameter is ignored. If the primitive was generated whenever a GTS is allocated or deallocated following the reception of a GTS request command: The originator of the key purportedly used by the originator of the received frame (see 7.7.2.4.1). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
KeyIndex	Integer	0x01–0xff	If the primitive was generated when a GTS deallocation is initiated by the VAN coordinator itself, this parameter is ignored. If the primitive was generated whenever a GTS is allocated or deallocated following the reception of a GTS request command: The index of the key purportedly used by the originator of the received frame (see 7.7.2.4.2). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.

7.1.7.3.2 When generated

The MLME-GTS.indication primitive is generated by the MLME of the VAN coordinator to its next higher layer whenever a GTS is allocated or deallocated following the reception of a GTS request command (see 7.4.10) by the MLME. The MLME of the VAN coordinator also generates this primitive when a GTS deallocation is initiated by the VAN coordinator itself. The Characteristics Type field in the GTSCharacteristics parameter will be equal to one if a GTS has been allocated or zero if a GTS has been deallocated.

This primitive is generated by the MLME of a device and issued to its next higher layer when the VAN coordinator has deallocated one of its GTSs. In this case, the Characteristics Type field of the GTSCharacteristics parameter is equal to zero.

7.1.7.3.3 Appropriate usage

On receipt of the MLME-GTS.indication primitive the next higher layer is notified of the allocation or de-allocation of a GTS.

7.1.7.4 GTS management message sequence charts

Figure 20 and Figure 21 illustrate the sequence of messages necessary for successful GTS management. The first depicts the message flow for the case in which the device initiates the GTS allocation. The second depicts the message flow for the two cases for which a GTS deallocation occurs, first, by a device (a) and, second, by the VAN coordinator (b).

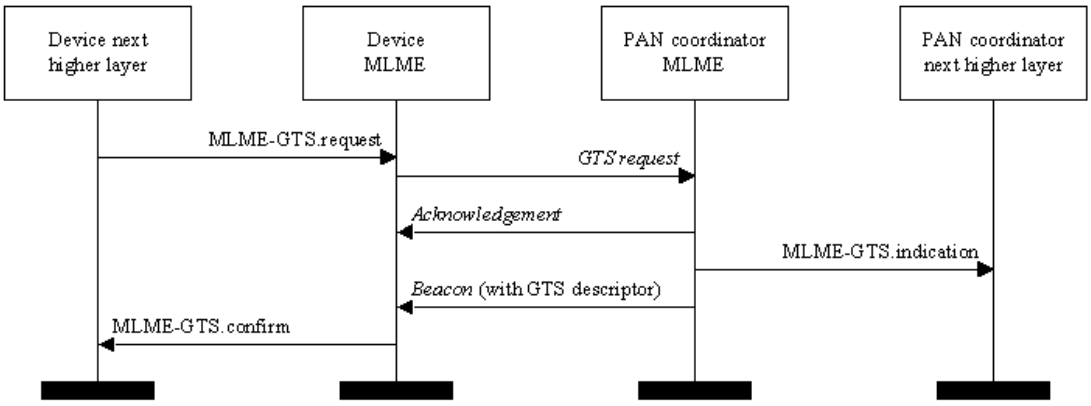


Figure 20—Message sequence chart for GTS allocation initiated by a device

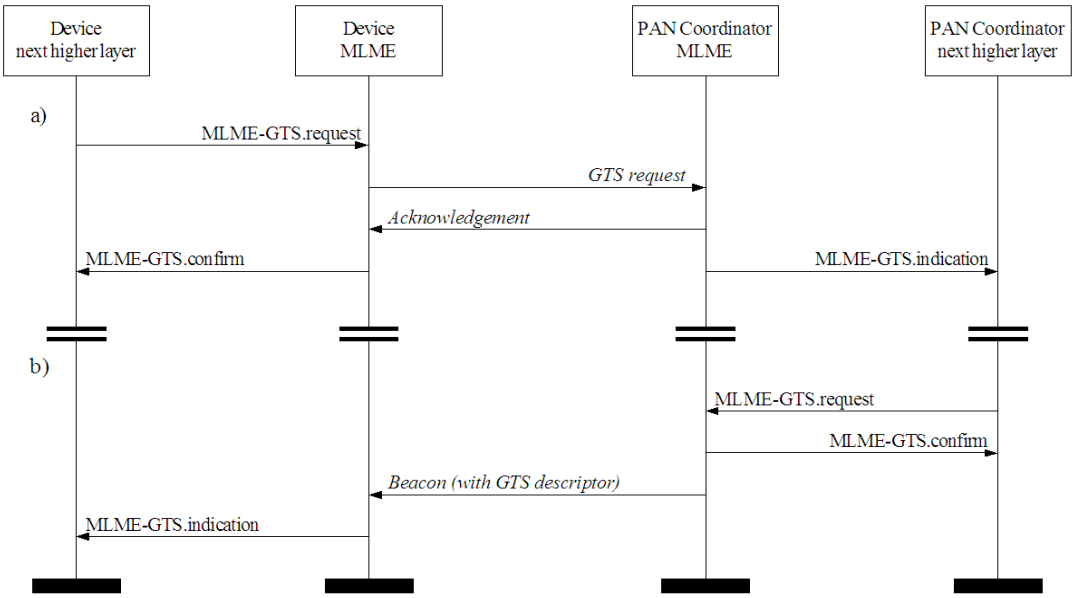


Figure 21—Message sequence chart for GTS deallocation initiated by a device (a) and the PAN coordinator (b)

7.1.8 Primitives for orphan notification

MLME-SAP orphan notification primitives define how a coordinator can issue a notification of an orphaned device.

These orphan notification primitives are optional for an RFD.

7.1.8.1 MLME-ORPHAN.indication

The MLME-ORPHAN.indication primitive allows the MLME of a coordinator to notify the next higher layer of the presence of an orphaned device.

7.1.8.1.1 Semantics of the service primitive

The semantics of the MLME-ORPHAN.indication primitive are as follows:

```
MLME-ORPHAN.indication      (
                            OrphanAddress,
                            SecurityLevel,
                            KeyIdMode,
                            KeySource,
                            KeyIndex
                            )
```

Table 45 specifies the parameters for the MLME-ORPHAN.indication primitive.

Table 45—MLME-ORPHAN.indication parameters

Name	Type	Valid range	Description
OrphanAddress	Device address	Extended 64-bit IEEE address	The address of the orphaned device.
SecurityLevel	Integer	0x00–0x07	The security level purportedly used by the received MAC command frame (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key purportedly used by the originator of the received frame (see Table 77 in 7.7.2.2.2). This parameter is invalid if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key purportedly used by the originator of the received frame (see 7.7.2.4.1). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key purportedly used by the originator of the received frame (see 7.7.2.4.2). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.

7.1.8.1.2 When generated

The MLME-ORPHAN.indication primitive is generated by the MLME of a coordinator and issued to its next higher layer on receipt of an orphan notification command (see 7.4.6).

7.1.8.1.3 Appropriate usage

The effect on receipt of the MLME-ORPHAN.indication primitive is that the next higher layer is notified of the orphaned device. The next higher layer then determines whether the device was previously associated and issues the MLME-ORPHAN.response primitive to the MLME with its decision (see 7.5.2.1.4).

If the device was previously associated with the coordinator, it will send the MLME-ORPHAN.response primitive with the AssociatedMember parameter set to TRUE and the ShortAddress parameter set to the corresponding 16-bit short address allocated to the orphaned device. If the device was not previously associated with the coordinator, it will send the MLME-ORPHAN.response primitive with the AssociatedMember parameter set to FALSE.

7.1.8.2 MLME-ORPHAN.response

The MLME-ORPHAN.response primitive allows the next higher layer of a coordinator to respond to the MLME-ORPHAN.indication primitive.

7.1.8.2.1 Semantics of the service primitive

The semantics of the MLME-ORPHAN.response primitive are as follows:

```
MLME-ORPHAN.response        (
                            OrphanAddress,
                            ShortAddress,
                            AssociatedMember,
                            SecurityLevel,
                            KeyIdMode,
                            KeySource,
                            KeyIndex
                            )
```

Table 46 specifies the parameters for the MLME-ORPHAN.response primitive.

Table 46—MLME-ORPHAN.response parameters

Name	Type	Valid range	Description
OrphanAddress	Device address	Extended 64-bit IEEE address	The address of the orphaned device.
ShortAddress	Integer	0x0000–0xffff	The 16-bit short address allocated to the orphaned device if it is associated with this coordinator. The special short address 0xfffe indicates that no short address was allocated, and the device will use its 64-bit extended address in all communications. If the device was not associated with this coordinator, this field will contain the value 0xffff and be ignored on receipt.
AssociatedMember	Boolean	TRUE or FALSE	TRUE if the orphaned device is associated with this coordinator or FALSE otherwise.
SecurityLevel	Integer	0x00–0x07	The security level to be used (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the SecurityLevel parameter is set to 0x00.

Table 46—MLME-ORPHAN.response parameters *(continued)*

Name	Type	Valid range	Description
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.

7.1.8.2.2 Appropriate usage

The MLME-ORPHAN.response primitive is generated by the next higher layer and issued to its MLME when it reaches a decision about whether the orphaned device indicated in the MLME-ORPHAN.indication primitive is associated.

7.1.8.2.3 Effect on receipt

If the AssociatedMember parameter is set to TRUE, the orphaned device is associated with the coordinator. In this case, the MLME generates and sends the coordinator realignment command (see 7.4.8) to the orphaned device containing the value of the ShortAddress field. This command is sent in the CAP if the coordinator is on a beacon-enabled VAN or immediately otherwise. If the AssociatedMember parameter is set to FALSE, the orphaned device is not associated with the coordinator and this primitive will be ignored. If the orphaned device does not receive the coordinator realignment command following its orphan notification within *macResponseWaitTime* symbols, it will assume it is not associated to any coordinator in range.

If the SecurityLevel parameter is set to a valid value other than 0x00, indicating that security is required for this frame, the MLME will set the Security Enabled subfield of the Frame Control field to one. The MAC sublayer will perform outgoing processing on the frame based on the OrphanAddress, SecurityLevel, KeyIdMode, KeySource, and KeyIndex parameters, as described in 7.6.8.2.1. If any error occurs during outgoing frame processing, the MLME will discard the frame and issue the MLME-COMM-STATUS.indication primitive with the error status returned by outgoing frame processing.

If the random access algorithm failed due to adverse conditions on the channel, the MAC sublayer will discard the frame and issue the MLME-COMM-STATUS.indication primitive with a status of CHANNEL_ACCESS_FAILURE.

The MAC sublayer enables its receiver immediately following the transmission of the frame and waits for an acknowledgment from the recipient (see 7.6.6.4). If the MAC sublayer does not receive an acknowledgment from the recipient, it will discard the frame and issue the MLME-COMM-STATUS.indication primitive with a status of NO_ACK.

If the frame was successfully transmitted and an acknowledgment was received, if requested, the MAC sublayer will issue the MLME-COMM-STATUS.indication primitive with a status of SUCCESS.

If any parameter in the MLME-ORPHAN.response primitive is not supported or is out of range, the MAC sublayer will issue the MLME-COMM-STATUS.indication primitive with a status of INVALID_PARAMETER.

7.1.8.3 Orphan notification message sequence chart

Figure 22 illustrates the sequence of messages necessary for a coordinator to issue a notification of an orphaned device.

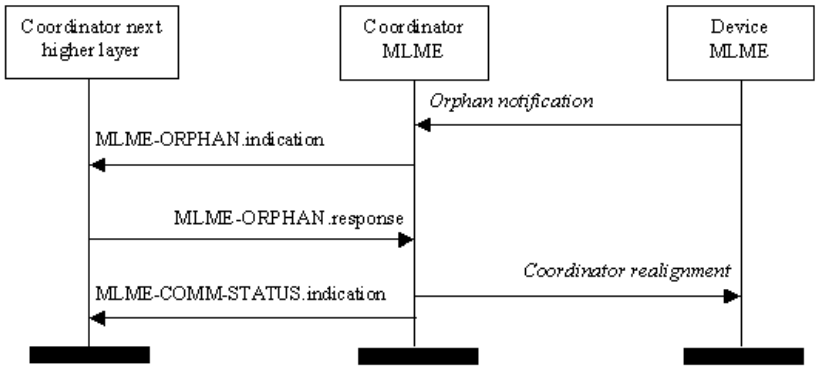


Figure 22—Message sequence chart for orphan notification

7.1.9 Primitives for resetting the MAC sublayer

MLME-SAP reset primitives specify how to reset the MAC sublayer to its default values.

All devices shall provide an interface for these reset primitives.

7.1.9.1 MLME-RESET.request

The MLME-RESET.request primitive allows the next higher layer to request that the MLME performs a reset operation.

7.1.9.1.1 Semantics of the service primitive

The semantics of the MLME-RESET.request primitive are as follows:

```
MLME-RESET.request          (
                            SetDefaultPIB
                            )
```

Table 47 specifies the parameter for the MLME-RESET.request primitive.

Table 47—MLME-RESET.request parameter

Name	Type	Valid range	Description
SetDefaultPIB	Boolean	TRUE or FALSE	If TRUE, the MAC sublayer is reset, and all MAC PIB attributes are set to their default values. If FALSE, the MAC sublayer is reset, but all MAC PIB attributes retain their values prior to the generation of the MLME-RESET.request primitive.

7.1.9.1.2 Appropriate usage

The MLME-RESET.request primitive is generated by the next higher layer and issued to the MLME to request a reset of the MAC sublayer to its initial conditions. The MLME-RESET.request primitive is issued prior to the use of the MLME-START.request or the MLME-ASSOCIATE.request primitives.

7.1.9.1.3 Effect on receipt

On receipt of the MLME-RESET.request primitive, the MLME issues the PLME-SET-TRX-STATE.request primitive with a state of FORCE_TRX_OFF. On receipt of the PLME-SET-TRX-STATE.confirm primitive, the MAC sublayer is then set to its initial conditions, clearing all internal variables to their default values. If the SetDefaultPIB parameter is set to TRUE, the MAC PIB attributes are set to their default values.

The MLME-RESET.confirm primitive with a status of SUCCESS is issued on completion.

7.1.9.2 MLME-RESET.confirm

The MLME-RESET.confirm primitive reports the results of the reset operation.

7.1.9.2.1 Semantics of the service primitive

The semantics of the MLME-RESET.confirm primitive are as follows:

```
MLME-RESET.confirm      (
                        status
                        )
```

Table 48 specifies the parameter for the MLME-RESET.confirm primitive.

Table 48—MLME-RESET.confirm parameter

Name	Type	Valid range	Description
status	Enumeration	SUCCESS	The result of the reset operation.

7.1.9.2.2 When generated

The MLME-RESET.confirm primitive is generated by the MLME and issued to its next higher layer in response to an MLME-RESET.request primitive and following the receipt of the PLME-SET-TRX-STATE.confirm primitive.

7.1.9.2.3 Appropriate usage

On receipt of the MLME-RESET.confirm primitive, the next higher layer is notified of its request to reset the MAC sublayer. This primitive returns a status of SUCCESS indicating that the request to reset the MAC sublayer was successful.

7.1.10 Primitives for channel scanning

TBD

7.1.11 Communication status primitive

The MLME-SAP communication status primitive defines how the MLME communicates to the next higher layer about transmission status, when the transmission was instigated by a response primitive, and about security errors on incoming packets.

All devices shall provide an interface for this communication status primitive.

7.1.11.1 MLME-COMM-STATUS.indication

The MLME-COMM-STATUS.indication primitive allows the MLME to indicate a communications status.

7.1.11.1.1 Semantics of the service primitive

The semantics of the MLME-COMM-STATUS.indication primitive are as follows:

```
MLME-COMM-STATUS.indication (
                            VANId,
                            SrcAddrMode,
                            SrcAddr,
                            DstAddrMode,
                            DstAddr,
                            status,
                            SecurityLevel,
                            KeyIdMode,
                            KeySource,
                            KeyIndex
                            )
```

Table 49 specifies the parameters for the MLME-COMM-STATUS.indication primitive.

Table 49—MLME-COMM-STATUS.indication parameters

Name	Type	Valid range	Description
VANId	Integer	0x0000–0xffff	The 16-bit VAN identifier of the device from which the frame was received or to which the frame was being sent.
SrcAddrMode	Integer	0x00–0x03	The source addressing mode for this primitive. This value can take one of the following values: 0 = no address (addressing fields omitted). 0x01 = reserved. 0x02 = 16-bit short address. 0x03 = 64-bit extended address.
SrcAddr	Device address	As specified by the SrcAddrMode parameter	The individual device address of the entity from which the frame causing the error originated.

Table 49—MLME-COMM-STATUS.indication parameters *(continued)*

Name	Type	Valid range	Description
DstAddrMode	Integer	0x00–0x03	The destination addressing mode for this primitive. This value can take one of the following values: 0x00 = no address (addressing fields omitted). 0x01 = reserved. 0x02 = 16-bit short address. 0x03 = 64-bit extended address.
DstAddr	Device address	As specified by the DstAddrMode parameter	The individual device address of the device for which the frame was intended.
status	Enumeration	SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, CHANNEL_ACCESS_FAILURE, NO_ACK, COUNTER_ERROR, FRAME_TOO_LONG, IMPROPER_KEY_TYPE, IMPROPER_SECURITY_LEVEL, SECURITY_ERROR, UNAVAILABLE_KEY, UNSUPPORTED_LEGACY, UNSUPPORTED_SECURITY or INVALID_PARAMETER	The communications status.
SecurityLevel	Integer	0x00–0x07	If the primitive was generated following a transmission instigated through a response primitive: The security level to be used (see Table 76 in 7.7.2.2.1). If the primitive was generated on receipt of a frame that generates an error in its security processing: The security level purportedly used by the received frame (see Table 76 in 7.7.2.2.1).

Table 49—MLME-COMM-STATUS.indication parameters *(continued)*

Name	Type	Valid range	Description
KeyIdMode	Integer	0x00–0x03	If the primitive was generated following a transmission instigated through a response primitive: The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the SecurityLevel parameter is set to 0x00. If the primitive was generated on receipt of a frame that generates an error in its security processing: The mode used to identify the key purportedly used by the originator of the received frame (see Table 77 in 7.7.2.2.2). This parameter is invalid if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	If the primitive was generated following a transmission instigated through a response primitive: The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00. If the primitive was generated on receipt of a frame that generates an error in its security processing: The originator of the key purportedly used by the originator of the received frame (see 7.7.2.4.1). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
KeyIndex	Integer	0x01–0xff	If the primitive was generated following a transmission instigated through a response primitive: The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00. If the primitive was generated on receipt of a frame that generates an error in its security processing: The index of the key purportedly used by the originator of the received frame (see 7.7.2.4.2). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.

7.1.11.1.2 When generated

The MLME-COMM-STATUS.indication primitive is generated by the MLME and issued to its next higher layer either following a transmission instigated through a response primitive or on receipt of a frame that generates an error in its security processing (see 7.6.8.2.3).

The MLME-COMM-STATUS.indication primitive is generated by the MAC sublayer entity following either the MLME-ASSOCIATE.response primitive or the MLME-ORPHAN.response primitive. This primitive returns a status of either SUCCESS, indicating that the request to transmit was successful, an error code of TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, CHANNEL_ACCESS_FAILURE, NO_ACK or INVALID_PARAMETER (these status values are fully described in 7.1.3.3.3 and 7.1.8.2.3), or an error code resulting from failed security processing (these status values are fully described in 7.6.8.2.1 and 7.6.8.2.3).

7.1.11.1.3 Appropriate usage

On receipt of the MLME-COMM-STATUS.indication primitive, the next higher layer is notified of the communication status of a transmission or notified of an error that has occurred during the secure processing of incoming frame.

7.1.12 Primitives for writing PIB attributes

MLME-SAP set primitives define how PIB attributes may be written.

All devices shall provide an interface for these set primitives.

7.1.12.1 MLME-SET.request

The MLME-SET.request primitive attempts to write the given value to the indicated PIB attribute.

7.1.12.1.1 Semantics of the primitive

The semantics of the MLME-SET.request primitive are as follows:

```
MLME-SET.request        (
                        PIBAttribute,
                        PIBAttributeIndex,
                        PIBAttributeValue
                        )
```

Table 50 specifies the parameters for the MLME-SET.request primitive.

7.1.12.1.2 Appropriate usage

The MLME-SET.request primitive is generated by the next higher layer and issued to its MLME to write the indicated PIB attribute.

7.1.12.1.3 Effect on receipt

On receipt of the MLME-SET.request primitive, the MLME checks to see if the PIB attribute is a MAC PIB attribute or PHY PIB attribute. If the requested attribute is a MAC attribute, the MLME attempts to write the given value to the indicated MAC PIB attribute in its database. If the PIBAttribute parameter specifies an attribute that is a read-only attribute (see Table 31 and Table 69), the MLME will issue the MLME-SET.confirm primitive with a status of READ_ONLY. If the PIBAttribute parameter specifies an attribute

Table 50—MLME-SET.request parameters

Name	Type	Valid range	Description
PIBAttribute	Integer	See Table 31 and Table 69	The identifier of the PIB attribute to write.
PIBAttributeIndex	Integer	Attribute specific; see Table 69	The index within the table of the specified PIB attribute to write. This parameter is valid only for MAC PIB attributes that are tables; it is ignored when accessing PHY PIB attributes.
PIBAttributeValue	Various	Attribute specific; see Table 31 and Table 69	The value to write to the indicated PIB attribute.

that is not found in the database, the MLME will issue the MLME-SET.confirm primitive with a status of UNSUPPORTED_ATTRIBUTE. If the PIBAttributeIndex parameter specifies an index for a table that is out of range, the MLME will issue the MLME-SET.confirm primitive with a status of INVALID_INDEX. If the PIBAttributeValue parameter specifies a value that is out of the valid range for the given attribute, the MLME will issue the MLME-SET.confirm primitive with a status of INVALID_PARAMETER. If the requested MAC PIB attribute is successfully written, the MLME will issue the MLME-SET.confirm primitive with a status of SUCCESS.

If the PIBAttribute parameter indicates that *macBeaconPayloadLength* is to be set and the length of the resulting beacon frame exceeds *aMaxPHYPacketSize* (e.g., due to the additional overhead required for security processing), the MAC sublayer shall not update *macBeaconPayloadLength* and will issue the MLME-GET.confirm primitive with a status of INVALID_PARAMETER.

If the requested attribute is a PHY PIB attribute, the request is passed to the PHY by issuing the PLME-SET.request primitive. Once the MLME receives the PLME-SET.confirm primitive, it will translate the received status value because the status values used by the PHY are not the same as those used by the MLME (e.g., the status values for SUCCESS are 0x00 and 0x07 in the MAC and PHY enumeration tables, respectively). Following the translation, the MLME will issue the MLME-SET.confirm primitive to the next higher layer with the status parameter resulting from the translation and the PIBAttribute parameter equal to that returned by the PLME primitive.

7.1.12.2 MLME-SET.confirm

The MLME-SET.confirm primitive reports the results of an attempt to write a value to a PIB attribute.

7.1.12.2.1 Semantics of the service primitive

The semantics of the MLME-SET.confirm primitive are as follows:

```
MLME-SET.confirm        (
                        status,
                        PIBAttribute,
                        PIBAttributeIndex
                        )
```

Table 51 specifies the parameters for the MLME-SET.confirm primitive.

Table 51—MLME-SET.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	SUCCESS, READ_ONLY, UNSUPPORTED_ATTRIBUTE, INVALID_INDEX or INVALID_PARAMETER	The result of the request to write the PIB attribute.
PIBAttribute	Integer	See Table 31 and Table 69	The identifier of the PIB attribute that was written.
PIBAttributeIndex	Integer	Attribute specific; see Table 69	The index within the table of the specified PIB attribute to write. This parameter is valid only for MAC PIB attributes that are tables; it is ignored when accessing PHY PIB attributes.

7.1.12.2.2 When generated

The MLME-SET.confirm primitive is generated by the MLME and issued to its next higher layer in response to an MLME-SET.request primitive. The MLME-SET.confirm primitive returns a status of either SUCCESS, indicating that the requested value was written to the indicated PIB attribute, or the appropriate error code. The status values are fully described in 7.1.12.1.3.

7.1.12.2.3 Appropriate usage

On receipt of the MLME-SET.confirm primitive, the next higher layer is notified of the result of its request to set the value of a PIB attribute. If the requested value was written to the indicated PIB attribute, the status parameter will be set to SUCCESS. Otherwise, the status parameter indicates the error.

7.1.13 Primitives for updating the superframe configuration

MLME-SAP start primitives define how an FFD can request to start using a new superframe configuration in order to initiate a VAN, begin transmitting beacons on an already existing VAN, thus facilitating device discovery, or to stop transmitting beacons.

These start primitives are optional for an RFD.

7.1.13.1 MLME-START.request

The MLME-START.request primitive allows the VAN coordinator to initiate a new VAN or to begin using a new superframe configuration. This primitive may also be used by a device already associated with an existing VAN to begin using a new superframe configuration.

7.1.13.1.1 Semantics of the service primitive

The semantics of the MLME-START.request primitive are as follows:

```
MLME-START.request      (
                        VANId,
                        LogicalChannel,
                        StartTime,
                        BeaconOrder,
                        SuperframeOrder,
                        VANCoordinator,
                        CoordRealignment,
                        CoordRealignSecurityLevel,
                        CoordRealignKeyIdMode,
                        CoordRealignKeySource,
                        CoordRealignKeyIndex,
                        BeaconSecurityLevel,
                        BeaconKeyIdMode,
                        BeaconKeySource,
                        BeaconKeyIndex
                        )
```

Table 52 specifies the parameters for the MLME-START.request primitive.

Table 52—MLME-START.request parameters

Name	Type	Valid range	Description
VANId	Integer	0x0000–0xffff	The VAN identifier to be used by the device.
LogicalChannel	Integer	Selected from the available logical channels specified by the ChannelPage parameter	The logical channel on which to start using the new superframe configuration.
ChannelPage	Integer	Selected from the available channel pages supported by the PHY (see 6.1.2)	The channel page on which to begin using the new superframe configuration.

Table 52—MLME-START.request parameters *(continued)*

Name	Type	Valid range	Description
StartTime	Integer	0x000000–0xffffff	The time at which to begin transmitting beacons. If this parameter is equal to 0x000000, beacon transmissions will begin immediately. Otherwise, the specified time is relative to the received beacon of the coordinator with which the device synchronizes. This parameter is ignored if either the BeaconOrder parameter has a value of 15 or the VANCoordinator parameter is TRUE. The time is specified in symbols and is rounded to a backoff slot boundary. This is a 24-bit value, and the precision of this value shall be a minimum of 20 bits, with the lowest 4 bits being the least significant.
BeaconOrder	Integer	0–15	How often the beacon is to be transmitted. A value of 15 indicates that the coordinator will not transmit periodic beacons. See 7.6.1.1 for an explanation of the relationship between the beacon order and the beacon interval.
SuperframeOrder	Integer	0–*BO* or 15	The length of the active portion of the superframe, including the beacon frame. If the BeaconOrder parameter (*BO*) has a value of 15, this parameter is ignored. See 7.6.1.1 for an explanation of the relationship between the superframe order and the superframe duration.
VANCoordinator	Boolean	TRUE or FALSE	If this value is TRUE, the device will become the VAN coordinator of a new VAN. If this value is FALSE, the device will begin using a new superframe configuration on the VAN with which it is associated.
CoordRealignment	Boolean	TRUE or FALSE	TRUE if a coordinator realignment command is to be transmitted prior to changing the superframe configuration or FALSE otherwise.
CoordRealignSecurity-Level	Integer	0x00–0x07	The security level to be used for coordinator realignment command frames (see Table 76 in 7.7.2.2.1).

Table 52—MLME-START.request parameters *(continued)*

Name	Type	Valid range	Description
CoordRealignKeyId-Mode	Integer	0x00–0x03	The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the CoordRealignSecurityLevel parameter is set to 0x00.
CoordRealignKey-Source	Set of 0, 4, or 8 octets	As specified by the CoordRealignKeyIdMode parameter	The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the CoordRealignKeyId-Mode parameter is ignored or set to 0x00.
CoordRealignKeyIndex	Integer	0x01–0xff	The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the CoordRealignKeyIdMode parameter is ignored or set to 0x00.
BeaconSecurityLevel	Integer	0x00–0x07	The security level to be used for beacon frames (see Table 76 in 7.7.2.2.1).
BeaconKeyIdMode	Integer	0x00–0x03	The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the BeaconSecurityLevel parameter is set to 0x00.
BeaconKeySource	Set of 0, 4, or 8 octets	As specified by the BeaconKeyIdMode parameter	The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the BeaconKeyIdMode parameter is ignored or set to 0x00.
BeaconKeyIndex	Integer	0x01–0xff	The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the BeaconKeyIdMode parameter is ignored or set to 0x00.

7.1.13.1.2 Appropriate usage

The MLME-START.request primitive is generated by the next higher layer and issued to its MLME to request that a device start using a new superframe configuration.

7.1.13.1.3 Effect on receipt

If the MLME-START.request primitive is received when *macShortAddress* is set to 0xffff, the MLME will issue the MLME-START.confirm primitive with a status of NO_SHORT_ADDRESS.

When the CoordRealignment parameter is set to TRUE, the coordinator attempts to transmit a coordinator realignment command frame as described in 7.6.2.3.2. If the transmission of the coordinator realignment command fails due to a channel access failure, the MLME will not make any changes to the superframe configuration (i.e., no PIB attributes will be changed) and will issue an MLME-START.confirm with a status of CHANNEL_ACCESS_FAILURE. If the coordinator realignment command is successfully transmitted, the MLME updates the appropriate PIB parameters with the values of the BeaconOrder, SuperframeOrder, VANId, ChannelPage, and LogicalChannel parameters, as described in 7.6.2.3.4, and will issue an MLME-START.confirm with a status of SUCCESS.

When the CoordRealignment parameter is set to FALSE, the MLME updates the appropriate PIB parameters with the values of the BeaconOrder, SuperframeOrder, VANId, ChannelPage, and LogicalChannel parameters, as described in 7.6.2.3.4.

The address used by the coordinator in its beacon frames is determined by the current value of *macShortAddress*, which is set by the next higher layer before issuing this primitive.

If the SecurityLevel parameter is set to a valid value other than 0x00, indicating that security is required for this frame, the MLME will set the Security Enabled subfield of the Frame Control field to one. The MAC sublayer will perform outgoing processing on the frame, as described in 7.6.8.2.1. If the CoordRealignment parameter is set to TRUE, the CoordRealignSecurityLevel, CoordRealignKeyIdMode, CoordRealignKeySource, and CoordRealignKeyIndex parameters will be used to process the MAC command frame. If the BeaconOrder parameter indicates a beacon-enabled network, the BeaconSecurityLevel, BeaconKeyIdMode, BeaconKeySource, and BeaconKeyIndex parameters will be used to process the beacon frame. If any error occurs during outgoing frame processing, the MLME will discard the frame and issue the MLME-START.confirm primitive with the error status returned by outgoing frame processing.

If the length of the beacon frame exceeds *aMaxPHYPacketSize* (e.g., due to the additional overhead required for security processing), the MAC sublayer shall discard the beacon frame and issue the MLME-START.confirm primitive with a status of FRAME_TOO_LONG.

The MLME shall ignore the StartTime parameter if the BeaconOrder parameter is equal to 15 because this indicates a nonbeacon-enabled VAN. If the BeaconOrder parameter is less than 15, the MLME examines the StartTime parameter to determine the time to begin transmitting beacons; the time is defined in symbols and is rounded to a backoff slot boundary. If the VAN coordinator parameter is set to TRUE, the MLME ignores the StartTime parameter and begins beacon transmissions immediately. Setting the StartTime parameter to 0x000000 also causes the MLME to begin beacon transmissions immediately. If the VANCoordinator parameter is set to FALSE and the StartTime parameter is nonzero, the MLME calculates the beacon transmission time by adding StartTime symbols to the time, obtained from the local clock, when the MLME receives the beacon of the coordinator through which it is associated. If the time calculated causes the outgoing superframe to overlap the incoming superframe, the MLME shall not begin beacon transmissions. In this case, the MLME issues the MLME-START.confirm primitive with a status of SUPERFRAME_OVERLAP. Otherwise, the MLME then begins beacon transmissions when the current time, obtained from the local clock, equals the number of calculated symbols.

If the StartTime parameter is nonzero and the MLME is not currently tracking the beacon of the coordinator through which it is associated, the MLME will issue the MLME-START.confirm primitive with a status of TRACKING_OFF.

On completion of this procedure, the MLME responds with the MLME-START.confirm primitive. If the attempt to start using a new superframe configuration was successful, the status parameter will be set to SUCCESS. If any parameter is not supported or is out of range, the status parameter will be set to INVALID_PARAMETER.

7.1.13.2 MLME-START.confirm

The MLME-START.confirm primitive reports the results of the attempt to start using a new superframe configuration.

7.1.13.2.1 Semantics of the service primitive

The semantics of the MLME-START.confirm primitive are as follows:

```
MLME-START.confirm      (
                        status
                        )
```

Table 53 specifies the parameters for the MLME-START.confirm primitive.

Table 53—MLME-START.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	SUCCESS, NO_SHORT_ADDRESS, SUPERFRAME_OVERLAP, TRACKING_OFF, INVALID_PARAMETER, COUNTER_ERROR, FRAME_TOO_LONG, UNAVAILABLE_KEY, UNSUPPORTED_SECURITY or CHANNEL_ACCESS_FAILURE	The result of the attempt to start using an updated superframe configuration.

7.1.13.2.2 When generated

The MLME-START.confirm primitive is generated by the MLME and issued to its next higher layer in response to an MLME-START.request primitive. The MLME-START.confirm primitive returns a status of either SUCCESS, indicating that the MAC sublayer has started using the new superframe configuration, or the appropriate error code. The status values are fully described in 7.1.13.1.3 and subclauses referenced by 7.1.13.1.3.

7.1.13.2.3 Appropriate usage

On receipt of the MLME-START.confirm primitive, the next higher layer is notified of the result of its request to start using a new superframe configuration. If the MAC sublayer has been successful, the status parameter will be set to SUCCESS. Otherwise, the status parameter indicates the error.

7.1.13.3 Message sequence chart for updating the superframe configuration

Figure 23 illustrates the sequence of messages necessary for initiating beacon transmissions in an FFD. Figure 66 (see 7.8) illustrates the sequence of messages necessary for the VAN coordinator to start beaconing on a new VAN; this figure includes steps taken by the PHY.

7.1.14 Primitives for synchronizing with a coordinator

MLME-SAP synchronization primitives define how synchronization with a coordinator may be achieved and how a loss of synchronization is communicated to the next higher layer.

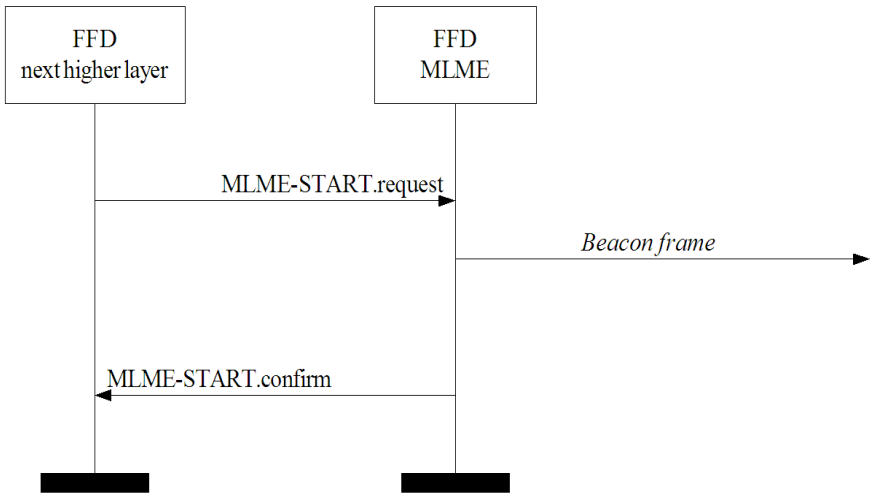


Figure 23—Message sequence chart for updating the superframe configuration

All devices shall provide an interface for the indication primitive. The request primitive is optional.

7.1.14.1 MLME-SYNC.request

The MLME-SYNC.request primitive requests to synchronize with the coordinator by acquiring and, if specified, tracking its beacons.

7.1.14.1.1 Semantics of the service primitive

The semantics of the MLME-SYNC.request primitive are as follows:

```
MLME-SYNC.request        (
                         LogicalChannel,
                         TrackBeacon
                         )
```

Table 54 specifies the parameters for the MLME-SYNC.request primitive.

Table 54—MLME-SYNC.request parameters

Name	Type	Valid range	Description
LogicalChannel	Integer	Selected from the available logical channels supported by the PHY	The logical channel on which to attempt coordinator synchronization.
TrackBeacon	Boolean	TRUE or FALSE	TRUE if the MLME is to synchronize with the next beacon and attempt to track all future beacons. FALSE if the MLME is to synchronize with only the next beacon.

7.1.14.1.2 Appropriate usage

The MLME-SYNC.request primitive is generated by the next higher layer of a device on a beacon-enabled VAN and issued to its MLME to synchronize with the coordinator.

7.1.14.1.3 Effect on receipt

If the MLME-SYNC.request primitive is received by the MLME on a beacon-enabled VAN, it will first set *phyCurrentPage* and *phyCurrentChannel* equal to the values of the ChannelPage and LogicalChannel parameters, respectively; both attributes are updated by issuing the PLME-SET.request primitive. If the TrackBeacon parameter is equal to TRUE, the MLME will track the beacon, i.e., enable its receiver just before the expected time of each beacon so that the beacon frame can be processed. If the TrackBeacon parameter is equal to FALSE, the MLME will locate the beacon, but not continue to track it.

If this primitive is received by the MLME while it is currently tracking the beacon, the MLME will not discard the primitive, but rather treat it as a new synchronization request.

If the beacon could not be located either on its initial search or during tracking, the MLME will issue the MLME-SYNC-LOSS.indication primitive with a loss reason of BEACON_LOST.

7.1.14.2 MLME-SYNC-LOSS.indication

The MLME-SYNC-LOSS.indication primitive indicates the loss of synchronization with a coordinator.

7.1.14.2.1 Semantics of the service primitive

The semantics of the MLME-SYNC-LOSS.indication primitive are as follows:

```
MLME-SYNC-LOSS.indication   (
                            LossReason,
                            VANId,
                            LogicalChannel,
                            SecurityLevel,
                            KeyIdMode,
                            KeySource,
                            KeyIndex
                            )
```

Table 55 specifies the parameters for the MLME-SYNC-LOSS.indication primitive.

Table 55—MLME-SYNC-LOSS.indication parameters

Name	Type	Valid range	Description
LossReason	Enumeration	VAN_ID_CONFLICT, REALIGNMENT, or BEACON_LOST	The reason that synchronization was lost.
VANId	Integer	0x0000–0xffff	The VAN identifier with which the device lost synchronization or to which it was realigned.

Table 55—MLME-SYNC-LOSS.indication parameters *(continued)*

Name	Type	Valid range	Description
LogicalChannel	Integer	Selected from the available logical channels supported by the PHY (see 6.1.2).	The logical channel on which the device lost synchronization or to which it was realigned.
SecurityLevel	Integer	0x00-0x07	If the primitive was either generated by the device itself following loss of synchronization or generated by the VAN coordinator upon detection of a VAN ID conflict, the security level is set to 0x00. If the primitive was generated following the reception of either a coordinator realignment command or a VAN ID conflict notification command: The security level purportedly used by the received MAC frame (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	If the primitive was either generated by the device itself following loss of synchronization or generated by the VAN coordinator upon detection of a VAN ID conflict, this parameter is ignored. If the primitive was generated following the reception of either a coordinator realignment command or a VAN ID conflict notification command: The mode used to identify the key purportedly used by the originator of the received frame (see Table 77 in 7.7.2.2.2). This parameter is invalid if the SecurityLevel parameter is set to 0x00.

Table 55—MLME-SYNC-LOSS.indication parameters *(continued)*

Name	Type	Valid range	Description
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	If the primitive was either generated by the device itself following loss of synchronization or generated by the VAN coordinator upon detection of a VAN ID conflict, this parameter is ignored. If the primitive was generated following the reception of either a coordinator realignment command or a VAN ID conflict notification command: The originator of the key purportedly used by the originator of the received frame (see 7.7.2.4.1). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.
KeyIndex	Integer	0x01–0xff	If the primitive was either generated by the device itself following loss of synchronization or generated by the VAN coordinator upon detection of a VAN ID conflict, this parameter is ignored. If the primitive was generated following the reception of either a coordinator realignment command or a VAN ID conflict notification command: The index of the key purportedly used by the originator of the received frame (see 7.7.2.4.2). This parameter is invalid if the KeyIdMode parameter is invalid or set to 0x00.

7.1.14.2.2 When generated

The MLME-SYNC-LOSS.indication primitive is generated by the MLME of a device and issued to its next higher layer in the event of a loss of synchronization with the coordinator. It is also generated by the MLME of the VAN coordinator and issued to its next higher layer in the event of a VAN ID conflict.

If a device that is associated through the VAN coordinator has detected a VAN identifier conflict and communicated it to the VAN coordinator, the MLME will issue this primitive with the LossReason parameter set to VAN_ID_CONFLICT. Similarly, if the VAN coordinator receives a VAN ID conflict notification command (see 7.4.5), the MLME will issue this primitive with the LossReason parameter set to VAN_ID_CONFLICT.

If a device has received the coordinator realignment command (see 7.4.8) from the coordinator through which it is associated and the MLME was not carrying out an orphan scan, the MLME will issue this primitive with the LossReason parameter set to REALIGNMENT and the VANId, LogicalChannel, ChannelPage, and security-related parameters set as described in 7.6.2.3.3.

If a device has not heard the beacon for *aMaxLostBeacons* consecutive superframes following an MLME-SYNC.request primitive, either initially or during tracking, the MLME will issue this primitive with the LossReason parameter set to BEACON_LOST. The VANId, LogicalChannel and ChannelPage parameters

shall be set according to the coordinator with which synchronization was lost. The SecurityLevel parameter shall be set to zero and the KeyIdMode, KeySource, and KeyIndex parameters shall be ignored. If the beacon was being tracked, the MLME will not attempt to track the beacon any further.

7.1.14.2.3 Appropriate usage

On receipt of the MLME-SYNC-LOSS.indication primitive, the next higher layer is notified of a loss of synchronization.

7.1.14.3 Message sequence chart for synchronizing with a coordinator

Figure 24 illustrates the sequence of messages necessary for a device to synchronize with a coordinator. In Figure 24a), a single synchronization request is issued. The MLME then searches for a beacon and, if found, determines whether the coordinator has any data pending for the device. If so, the data are requested as described in 7.6.6.3. In Figure 24b), a track synchronization request is issued. The MLME then searches for a beacon and, if found, attempts to keep track of it using a timer that expires just before the expected time of the next beacon.

For both examples Figure 24a) and Figure 24b), the received beacon frames do not contain payload, and *macAutoRequest* is set to TRUE. The MLME also checks for any data pending in the coordinator for the device when a beacon frame is received.

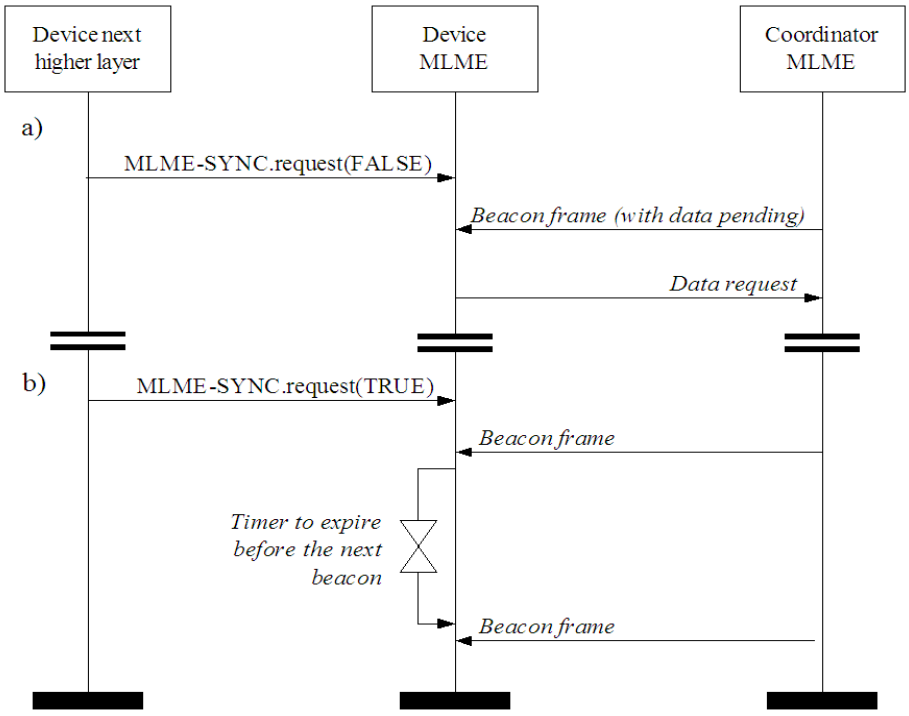


Figure 24—Message sequence chart for synchronizing to a coordinator in a beacon-enabled VAN

7.1.15 Primitives for requesting data from a coordinator

MLME-SAP polling primitives define how to request data from a coordinator.

All devices shall provide an interface for these polling primitives.

7.1.15.1 MLME-POLL.request

The MLME-POLL.request primitive prompts the device to request data from the coordinator.

7.1.15.1.1 Semantics of the service primitive

The semantics of the MLME-POLL.request primitive are as follows:

```
MLME-POLL.request       (
                        CoordAddrMode,
                        CoordVANId,
                        CoordAddress,
                        SecurityLevel,
                        KeyIdMode,
                        KeySource,
                        KeyIndex
                        )
```

Table 56 specifies the parameter for the MLME-POLL.request primitive.

Table 56—MLME-POLL.request parameters

Name	Type	Valid range	Description
CoordAddrMode	Integer	0x02–0x03	The addressing mode of the coordinator to which the poll is intended. This parameter can take one of the following values: 2 = 16-bit short address, 3 = 64-bit extended address.
CoordVANId	Integer	0x0000–0xfffe	The VAN identifier of the coordinator to which the poll is intended.
CoordAddress	Device-Address	As specified by the CoordAddrMode parameter	The address of the coordinator to which the poll is intended.
SecurityLevel	Integer	0x00–0x07	The security level to be used (see Table 76 in 7.7.2.2.1).
KeyIdMode	Integer	0x00–0x03	The mode used to identify the key to be used (see Table 77 in 7.7.2.2.2). This parameter is ignored if the SecurityLevel parameter is set to 0x00.
KeySource	Set of 0, 4, or 8 octets	As specified by the KeyIdMode parameter	The originator of the key to be used (see 7.7.2.4.1). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.
KeyIndex	Integer	0x01–0xff	The index of the key to be used (see 7.7.2.4.2). This parameter is ignored if the KeyIdMode parameter is ignored or set to 0x00.

7.1.15.1.2 Appropriate usage

The MLME-POLL.request primitive is generated by the next higher layer and issued to its MLME when data are to be requested from a coordinator.

7.1.15.1.3 Effect on receipt

On receipt of the MLME-POLL.request primitive, the MLME generates and sends a data request command (see 7.4.4). If the poll is directed to the VAN coordinator, the data request command may be generated without any destination address information present. Otherwise, the data request command is always generated with the destination address information in the CoordVANId and CoordAddress parameters.

If the SecurityLevel parameter is set to a valid value other than 0x00, indicating that security is required for this frame, the MLME will set the Security Enabled subfield of the Frame Control field to one. The MAC sublayer will perform outgoing processing on the frame based on the CoordAddress, SecurityLevel, KeyIdMode, KeySource, and KeyIndex parameters, as described in 7.6.8.2.1. If any error occurs during outgoing frame processing, the MLME will discard the frame and issue the MLME-POLL.confirm primitive with the error status returned by outgoing frame processing.

If the data request command cannot be sent due to a random access algorithm failure, the MLME will issue the MLME-POLL.confirm primitive with a status of CHANNEL_ACCESS_FAILURE.

If the MLME successfully transmits a data request command, the MLME will expect an acknowledgment in return. If an acknowledgment is not received, the MLME will issue the MLME-POLL.confirm primitive with a status of NO_ACK (see 7.6.6.4).

If an acknowledgment is received, the MLME will request that the PHY enable its receiver if the Frame Pending subfield of the acknowledgment frame is set to one. If the Frame Pending subfield of the acknowledgment frame is set to zero, the MLME will issue the MLME-POLL.confirm primitive with a status of NO_DATA.

If a frame is received from the coordinator with a zero length payload or if the frame is a MAC command frame, the MLME will issue the MLME-POLL.confirm primitive with a status of NO_DATA. If a frame is received from the coordinator with nonzero length payload, the MLME will issue the MLME-POLL.confirm primitive with a status of SUCCESS. In this case, the actual data are indicated to the next higher layer using the MCPS-DATA.indication primitive (see 7.1.1.3).

If a frame is not received within *macMaxFrameTotalWaitTime* CAP symbols in a beacon-enabled VAN, or symbols in a nonbeacon-enabled VAN, even though the acknowledgment to the data request command has its Frame Pending subfield set to one, the MLME will issue the MLME-POLL.confirm primitive with a status of NO_DATA.

If any parameter in the MLME-POLL.request primitive is not supported or is out of range, the MLME will issue the MLME-POLL.confirm primitive with a status of INVALID_PARAMETER.

7.1.15.2 MLME-POLL.confirm

The MLME-POLL.confirm primitive reports the results of a request to poll the coordinator for data.

7.1.15.2.1 Semantics of the service primitive

The semantics of the MLME-POLL.confirm primitive are as follows:

```
MLME-POLL.confirm       (
                        status
                        )
```

Table 57 specifies the parameters for the MLME-POLL.confirm primitive.

Table 57—MLME-POLL.confirm parameters

Name	Type	Valid range	Description
status	Integer	SUCCESS, CHANNEL_ACCESS_FAILURE, NO_ACK, NO_DATA, COUNTER_ERROR, FRAME_TOO_LONG, UNAVAILABLE_KEY, UNSUPPORTED_SECURITY or INVALID_PARAMETER	The status of the data request.

7.1.15.2.2 When generated

The MLME-POLL.confirm primitive is generated by the MLME and issued to its next higher layer in response to an MLME-POLL.request primitive. If the request was successful, the status parameter will be equal to SUCCESS, indicating a successful poll for data. Otherwise, the status parameter indicates the appropriate error code. The status values are fully described in 7.1.15.1.3 and the subclauses referenced by 7.1.15.1.3.

7.1.15.2.3 Appropriate usage

On receipt of the MLME-POLL.confirm primitive, the next higher layer is notified of the status of the procedure to request data from the coordinator.

7.1.15.3 Message sequence chart for requesting data from a coordinator

Figure 25 illustrates the sequence of messages necessary, including the layer behavior of the device and the over-the-air interface, for a device to request data from a coordinator.

In both scenarios Figure 25a) and Figure 25b), a poll request is issued to the MLME, which then sends a data request command to the coordinator. In Figure 25a), the corresponding acknowledgment has the Frame Pending (FP) subfield set to zero and the MLME issues the poll request confirmation immediately. In Figure 25b), the corresponding acknowledgment has the Frame Pending subfield set to one and the MLME enables the receiver in anticipation of the data frame from the coordinator. On receipt of this data frame, the MLME issues a poll request confirmation followed by a data indication containing the data of the received frame.

7.1.16 Primitives for requesting dimmer settings from the DME

MLME-SAP dimmer primitives defines how to request dimmer data from a DME.

All devices shall provide an interface for these dimmer primitives.

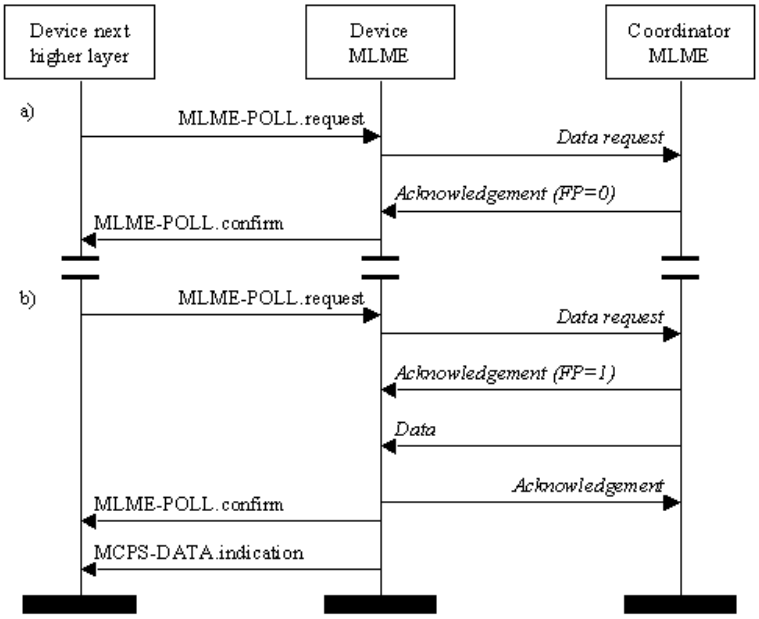


Figure 25—Message sequence chart for requesting data from the coordinator

(Ed. Note: Figure 25 needs to be modified for the DIMMER primitive)

7.1.16.1 MLME-DIMMER.request

The MLME-DIMMER.request primitive requests that the MLME perform the dimmer function as defined in x.x.x.

7.1.16.1.1 Semantics of the service primitive

The semantics of the PLME-DIMMER.request primitive is as follows:

MLME-DIMMER.request (

PIBDim

PIBDimValue

)

Table 14 specifies the parameters for the MLME-DIMMER.request primitive.

7.1.16.1.2 When generated

The MLME-DIMMER.request primitive is generated by the MLME and issued to its MLME whenever the dimmer algorithm requires the required dimming value.

Table 58—MLME-DIMMER.request parameters

Name	Type	Valid range	Description
PIBAttribute	Enumeration	See Table X	The identifier of the PIB attribute to set.
PIBAttributeValue	Various	Attribute specific	The value of the indicated PIB attribute to set.

7.1.16.1.3 Effect on receipt

If the transmitter is enabled on receipt of the MLME-DIMMER.request primitive, the MLME will cause the MAC to perform the required dimming function. When the MAC has completed the required dimming, the MLME will issue the MLME-DIMMER.confirm primitive with the status of COMPLETE.

7.1.16.2 MLME-DIMMER.confirm

The MLME-DIMMER.confirm primitive reports the results of a dimming request.

7.1.16.2.1 Semantics of the service primitive

The semantics of the MLME-DIMMER.confirm primitive is as follows:

```
MLME-DIMMER.confirm     (
                        status
                        )
```

Table 8 specifies the parameters for the MLME-DIMMER.confirm primitive.

Table 59—MLME-DIMMER.confirm parameters

Name	Type	Valid range	Description
status	Enumeration	TRX_OFF, TX_ON, BUSY, or IDLE	The result of the request to perform a CCA.

7.1.16.2.2 When generated

The MLME-DIMMER.confirm primitive is generated by the MLME and issued to its MLME in response to a MLME-DIMMER.request primitive. The MLME-DIMMER.confirm primitive will return a status of either BUSY or IDLE, indicating a successful dimming. The reasons for these status values are fully described in 6.x.x.x.x.

7.1.16.2.3 Effect on receipt

On receipt of the MLME-DIMMER.confirm primitive, the MLME is notified of the results of the dimming. If the DIMMER attempt was successful, the status parameter is set to either BUSY or IDLE. Otherwise, the status parameter will indicate the error.

7.2 MAC frame formats

This subclause specifies the format of the MAC frame (MPDU). Each MAC frame consists of the following basic components:

a) A MHR, which comprises frame control, sequence number, address information, and security-related information.

b) A MAC payload, of variable length, which contains information specific to the frame type. Acknowledgment frames do not contain a payload.

c) A MFR, which contains a FCS.

The frames in the MAC sublayer 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 PHY, 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.

For every MAC frame, all reserved bits shall be set to zero upon transmission and shall be ignored upon receipt.

7.2.1 General MAC frame format

The MAC frame format is composed of a MHR, a MAC payload, and a MFR. The fields of the MHR appear in a fixed order; however, the addressing fields may not be included in all frames. The general MAC frame shall be formatted as illustrated in Figure 26.

<table>
<tr><th>Octets: 2</th><th>1</th><th>0/2</th><th>0/2/8</th><th>0/2</th><th>0/2/8</th><th>0/5/6/10/ 14</th><th>variable</th><th>2</th></tr>
<tr><td rowspan="2">Frame Control</td><td rowspan="2">Sequence Number</td><td>Destination VAN Identifier</td><td>Destination Address</td><td>Source VAN Identifier</td><td>Source Address</td><td rowspan="2">Auxiliary Security Header</td><td rowspan="2">Frame Payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td></tr>
<tr><td colspan="7">MHR</td><td>MAC Payload</td><td>MFR</td></tr>
</table>

Figure 26—General MAC frame format

7.2.1.1 Frame Control field

The Frame Control field is 2 octets 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 27.

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame Type	Security Enabled	Frame Pending	Ack. Request	VAN ID Compression	Reserved	Dest. Addressing Mode	Frame Version	Source Addressing Mode

Figure 27—Format of the Frame Control field

7.2.1.1.1 Frame Type subfield

The Frame Type subfield is 3 bits in length and shall be set to one of the nonreserved values listed in Table 60.

Table 60—Values of the Frame Type subfield

Frame type value $b_2\ b_1\ b_0$	Description
000	Beacon
001	Data
010	Acknowledgment
011	MAC command
100–111	*Reserved*

7.2.1.1.2 Security Enabled subfield

The Security Enabled subfield is 1 bit in length, and it shall be set to one if the frame is protected by the MAC sublayer and shall be set to zero otherwise. The Auxiliary Security Header field of the MHR shall be present only if the Security Enabled subfield is set to one.

7.2.1.1.3 Frame Pending subfield

The Frame Pending subfield is 1 bit in length and shall be set to one if the device sending the frame has more data for the recipient. This subfield shall be set to zero otherwise (see 7.6.6.3).

The Frame Pending subfield shall be used only in beacon frames or frames transmitted either during the CAP by devices operating on a beacon-enabled VAN or at any time by devices operating on a nonbeacon-enabled VAN.

At all other times, it shall be set to zero on transmission and ignored on reception.

7.2.1.1.4 Acknowledgment Request subfield

The Acknowledgment Request subfield is 1 bit in length and specifies whether an acknowledgment is required from the recipient device on receipt of a data or MAC command frame. If this subfield is set to one, the recipient device shall send an acknowledgment frame only if, upon reception, the frame passes the third level of filtering (see 7.6.6.2). If this subfield is set to zero, the recipient device shall not send an acknowledgment frame.

7.2.1.1.5 VAN ID Compression subfield

The VAN ID Compression subfield is 1 bit in length and specifies whether the MAC frame is to be sent containing only one of the VAN identifier fields when both source and destination addresses are present. If this subfield is set to one and both the source and destination addresses are present, the frame shall contain only the Destination VAN Identifier field, and the Source VAN Identifier field shall be assumed equal to that of the destination. If this subfield is set to zero and both the source and destination addresses are present, the frame shall contain both the Source VAN Identifier and Destination VAN Identifier fields. If only one of the addresses is present, this subfield shall be set to zero, and the frame shall contain the VAN identifier field corresponding to the address. If neither address is present, this subfield shall be set to zero, and the frame shall not contain either VAN identifier field.

7.2.1.1.6 Destination Addressing Mode subfield

The Destination Addressing Mode subfield is 2 bits in length and shall be set to one of the nonreserved values listed in Table 61.

If this subfield is equal to zero and the Frame Type subfield does not specify that this frame is an acknowledgment or beacon frame, the Source Addressing Mode subfield shall be nonzero, implying that the frame is directed to the VAN coordinator with the VAN identifier as specified in the Source VAN Identifier field.

Table 61—Possible values of the Destination Addressing Mode and Source Addressing Mode subfields

Addressing mode value b_1 b_0	Description
00	VAN identifier and address fields are not present.
01	Reserved.
10	Address field contains a 16-bit short address.
11	Address field contains a 64-bit extended address.

7.2.1.1.7 Frame Version subfield

The Frame Version subfield is 2 bits in length and specifies the version number corresponding to the frame.

This subfield shall be set to 0x00 to indicate a frame compatible with IEEE Std 802.15.7 and 0x01 to indicate an IEEE802.15.7 frame. All other subfield values shall be reserved for future use. See 7.2.3 for details on frame compatibility.

7.2.1.1.8 Source Addressing Mode subfield

The Source Addressing Mode subfield is 2 bits in length and shall be set to one of the nonreserved values listed in Table 61.

If this subfield is equal to zero and the Frame Type subfield does not specify that this frame is an acknowledgment frame, the Destination Addressing Mode subfield shall be nonzero, implying that the frame has originated from the VAN coordinator with the VAN identifier as specified in the Destination VAN Identifier field.

7.2.1.2 Sequence Number field

The Sequence Number field is 1 octet in length and specifies the sequence identifier for the frame.

For a beacon frame, the Sequence Number field shall specify a BSN. For a data, acknowledgment, or MAC command frame, the Sequence Number field shall specify a DSN that is used to match an acknowledgment frame to the data or MAC command frame.

7.2.1.3 Destination VAN Identifier field

The Destination VAN Identifier field, when present, is 2 octets in length and specifies the unique VAN identifier of the intended recipient of the frame. A value of 0xffff in this field shall represent the broadcast VAN identifier, which shall be accepted as a valid VAN identifier by all devices currently listening to the channel.

This field shall be included in the MAC frame only if the Destination Addressing Mode subfield of the Frame Control field is nonzero.

7.2.1.4 Destination Address field

The Destination Address field, when present, is either 2 octets or 8 octets in length, according to the value specified in the Destination Addressing Mode subfield of the Frame Control field (see 7.2.1.1.6), and specifies the address of the intended recipient of the frame. A 16-bit value of 0xffff in this field shall represent the broadcast short address, which shall be accepted as a valid 16-bit short address by all devices currently listening to the channel.

This field shall be included in the MAC frame only if the Destination Addressing Mode subfield of the Frame Control field is nonzero.

7.2.1.5 Source VAN Identifier field

The Source VAN Identifier field, when present, is 2 octets in length and specifies the unique VAN identifier of the originator of the frame. This field shall be included in the MAC frame only if the Source Addressing Mode and VAN ID Compression subfields of the Frame Control field are nonzero and equal to zero, respectively.

The VAN identifier of a device is initially determined during association on a VAN, but may change following a VAN identifier conflict resolution (see 7.6.2.2).

7.2.1.6 Source Address field

The Source Address field, when present, is either 2 octets or 8 octets in length, according to the value specified in the Source Addressing Mode subfield of the Frame Control field (see 7.2.1.1.8), and specifies the address of the originator of the frame. This field shall be included in the MAC frame only if the Source Addressing Mode subfield of the Frame Control field is nonzero.

7.2.1.7 Auxiliary Security Header field

The Auxiliary Security Header field has a variable length and specifies information required for security processing, including how the frame is actually protected (security level) and which keying material from the MAC security PIB is used (see 7.7.1). This field shall be present only if the Security Enabled subfield is set to one. For details on formatting, see 7.7.2.

7.2.1.8 Frame Payload field

The Frame Payload field has a variable length and contains information specific to individual frame types. If the Security Enabled subfield is set to one in the Frame Control field, the frame payload is protected as defined by the security suite selected for that frame.

7.2.1.9 FCS field

The FCS field is 2 octets in length and contains a 16-bit ITU-T CRC. The FCS is calculated over the MHR and MAC payload parts of the frame.

The FCS shall be calculated using the following standard generator polynomial of degree 16:

$$G_{16}(x) = x^{16} + x^{12} + x^5 + 1 \quad (1)$$

The FCS shall be calculated for transmission using the following algorithm:

— Let $M(x) = b_0 x^{k-1} + b_1 x^{k-2} + \ldots + b_{k-2} x + b_{k-1}$ be the polynomial representing the sequence of bits for which the checksum is to be computed.

— Multiply $M(x)$ by x^{16}, giving the polynomial $x^{16} \times M(x)$.

— Divide $x^{16} \times M(x)$ modulo 2 by the generator polynomial, $G_{16}(x)$, to obtain the remainder polynomial, $R(x) = r_0 x^{15} + r_1 x^{14} + \ldots + r_{14} x + r_{15}$.

— The FCS field is given by the coefficients of the remainder polynomial, *R(x)*.

Here, binary polynomials are represented as bit strings, in highest polynomial degree first order.

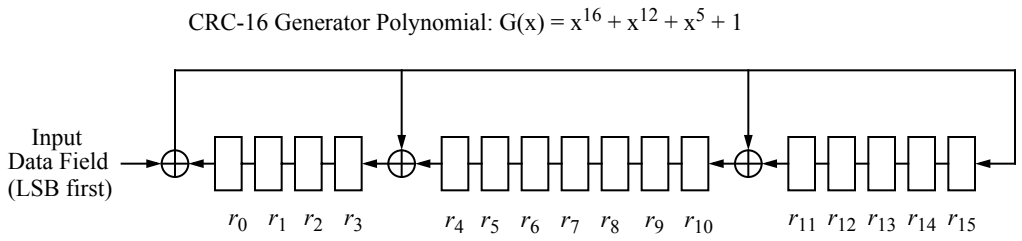
As an example, consider an acknowledgment frame with no payload and the following 3 byte MHR:

0100 0000 0000 0000 0101 0110 [leftmost bit (b_0) transmitted first in time]
b_0.....b_{23}

The FCS for this case would be the following:

0010 0111 1001 1110 [leftmost bit (r_0) transmitted first in time]
r_0.....r_{15}

A typical implementation is depicted in Figure 28.



1. Initialize the remainder register (r_0 through r_{15}) to zero.
2. Shift MHR and payload into the divider in the order of transmission (LSB first).
3. After the last bit of the data field is shifted into the divider, the remainder register contains the FCS.
4. The FCS is appended to the data field so that r_0 is transmitted first.

Figure 28—Typical FCS implementation

7.2.2 Format of individual frame types

Four frame types are defined: beacon, data, acknowledgment, and MAC command. These frame types are discussed in 7.2.2.1 through 7.2.2.4.

7.2.2.1 Beacon frame format

The beacon frame shall be formatted as illustrated in Figure 29.

Octets: 2	**1**	**4/10**	**0/5/6/10/14**	**2**	**variable**	**variable**	**variable**	**2**
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Superframe Specification	GTS fields (Figure 30)	Pending address fields (Figure 31)	Beacon Payload	FCS
MHR				MAC Payload				MFR

Figure 29—Beacon frame format

The GTS fields shall be formatted as illustrated in Figure 30, and the pending address fields shall be formatted as illustrated in Figure 31.

Octets: 1	**0/1**	**variable**
GTS Specification	GTS Directions	GTS List

Figure 30—Format of the GTS information fields

Octets: 1	variable
Pending Address Specification	Address List

Figure 31—Format of the pending address information fields

The order of the fields of the beacon frame shall conform to the order of the general MAC frame as illustrated in Figure 26.

7.2.2.1.1 Beacon frame MHR fields

The MHR for a beacon frame shall contain the Frame Control field, the Sequence Number field, the Source VAN Identifier field, and the Source Address field.

In the Frame Control field, the Frame Type subfield shall contain the value that indicates a beacon frame, as shown in Table 60, and the Source Addressing Mode subfield shall be set as appropriate for the address of the coordinator transmitting the beacon frame. If protection is used for the beacon, the Security Enabled subfield shall be set to one. The Frame Version subfield shall be set to one only if the Security Enabled subfield is set to one. If a broadcast data or command frame is pending, the Frame Pending subfield shall be set to one. All other subfields shall be set to zero and ignored on reception.

The Sequence Number field shall contain the current value of *macBSN*.

The addressing fields shall comprise only the source address fields. The Source VAN Identifier and Source Address fields shall contain the VAN identifier and address, respectively, of the device transmitting the beacon.

The Auxiliary Security Header field, if present, shall contain the information required for security processing of the beacon frame, as specified in 7.2.1.7.

7.2.2.1.2 Superframe Specification field

The Superframe Specification field is 16 bits in length and shall be formatted as illustrated in Figure 32.

Bits: 0–3	4-7	8-11	12	13	14	15
Beacon Order	Superframe Order	Final CAP Slot	Battery Life Extension (BLE)	Reserved	VAN Coordinator	Association Permit

Figure 32—Format of the Superframe Specification field

The Beacon Order subfield is 4 bits in length and shall specify the transmission interval of the beacon. See 7.6.1.1 for an explanation of the relationship between the beacon order and the beacon interval.

The Superframe Order subfield is 4 bits in length and shall specify the length of time during which the superframe is active (i.e., receiver enabled), including the beacon frame transmission time. See 7.6.1.1 for an explanation of the relationship between the superframe order and the superframe duration.

The Final CAP Slot subfield is 4 bits in length and specifies the final superframe slot utilized by the CAP. The duration of the CAP, as implied by this subfield, shall be greater than or equal to the value specified by

aMinCAPLength. However, an exception is allowed for the accommodation of the temporary increase in the beacon frame length needed to perform GTS maintenance (see 7.2.2.1.3).

The Battery Life Extension (BLE) subfield is 1 bit in length and shall be set to one if frames transmitted to the beaconing device during its CAP are required to start in or before *macBattLifeExtPeriods* full backoff periods after the IFS period following the beacon. Otherwise, the BLE subfield shall be set to zero.

The VAN Coordinator subfield is 1 bit in length and shall be set to one if the beacon frame is being transmitted by the VAN coordinator. Otherwise, the VAN Coordinator subfield shall be set to zero.

The Association Permit subfield is 1 bit in length and shall be set to one if *macAssociationPermit* is set to TRUE (i.e., the coordinator is accepting association to the VAN). The association permit bit shall be set to zero if the coordinator is currently not accepting association requests on its network.

7.2.2.1.3 GTS Specification field

The GTS Specification field is 8 bits in length and shall be formatted as illustrated in Figure 33.

Bits: 0-2	3-6	7
GTS Descriptor Count	Reserved	GTS Permit

Figure 33—Format of the GTS Specification field

The GTS Descriptor Count subfield is 3 bits in length and specifies the number of 3-octet GTS descriptors contained in the GTS List field of the beacon frame. If the value of this subfield is greater than zero, the size of the CAP shall be allowed to dip below *aMinCAPLength* to accommodate the temporary increase in the beacon frame length caused by the inclusion of the subfield. If the value of this subfield is zero, the GTS Directions field and GTS List field of the beacon frame are not present.

The GTS Permit subfield is 1 bit in length and shall be set to one if *macGTSPermit* is equal to TRUE (i.e., the VAN coordinator is accepting GTS requests). Otherwise, the GTS Permit field shall be set to zero.

7.2.2.1.4 GTS Directions field

The GTS Directions field is 8 bits in length and shall be formatted as illustrated in Figure 34.

Bits: 0-6	7
GTS Directions Mask	Reserved

Figure 34—Format of the GTS Directions field

The GTS Directions Mask subfield is 7 bits in length and contains a mask identifying the directions of the GTSs in the superframe. The lowest bit in the mask corresponds to the direction of the first GTS contained in the GTS List field of the beacon frame, with the remainder appearing in the order that they appear in the list. Each bit shall be set to one if the GTS is a receive-only GTS or to zero if the GTS is a transmit-only GTS. GTS direction is defined relative to the direction of the data frame transmission by the device.

7.2.2.1.5 GTS List field

The size of the GTS List field is defined by the values specified in the GTS Specification field of the beacon frame and contains the list of GTS descriptors that represents the GTSs that are being maintained. The maximum number of GTS descriptors shall be limited to seven.

Each GTS descriptor is 24 bits in length and shall be formatted as illustrated in Figure 35.

Bits: 0-15	16-19	20-23
Device Short Address	GTS Starting Slot	GTS Length

Figure 35—Format of the GTS descriptor

The Device Short Address subfield is 16 bits in length and shall contain the short address of the device for which the GTS descriptor is intended.

The GTS Starting Slot subfield is 4 bits in length and contains the superframe slot at which the GTS is to begin.

The GTS Length subfield is 4 bits in length and contains the number of contiguous superframe slots over which the GTS is active.

7.2.2.1.6 Pending Address Specification field

The Pending Address Specification field shall be formatted as illustrated in Figure 36.

Bits: 0-2	3	4-6	7
Number of Short Addresses Pending	Reserved	Number of Extended Addresses Pending	Reserved

Figure 36—Format of the Pending Address Specification field

The Number of Short Addresses Pending subfield is 3 bits in length and indicates the number of 16-bit short addresses contained in the Address List field of the beacon frame.

The Number of Extended Addresses Pending subfield is 3 bits in length and indicates the number of 64-bit extended addresses contained in the Address List field of the beacon frame.

7.2.2.1.7 Address List field

The size of the Address List field is determined by the values specified in the Pending Address Specification field of the beacon frame and contains the list of addresses of the devices that currently have messages pending with the coordinator. The address list shall not contain the broadcast short address 0xffff.

The maximum number of addresses pending shall be limited to seven and may comprise both short and extended addresses. All pending short addresses shall appear first in the list followed by any extended addresses. If the coordinator is able to store more than seven transactions, it shall indicate them in its beacon on a first-come-first-served basis, ensuring that the beacon frame contains at most seven addresses.

7.2.2.1.8 Beacon Payload field

The Beacon Payload field is an optional sequence of up to *aMaxBeaconPayloadLength* octets specified to be transmitted in the beacon frame by the next higher layer. The set of octets contained in *macBeaconPayload* shall be copied into this field.

7.2.2.2 Data frame format

The data frame shall be formatted as illustrated in Figure 37.

Octets: 2	1	(see 7.2.2.2.1)	0/5/6/10/14	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Data Payload	FCS
MHR				MAC Payload	MFR

Figure 37—Data frame format

The order of the fields of the data frame shall conform to the order of the general MAC frame as illustrated in Figure 26.

7.2.2.2.1 Data frame MHR fields

The MHR for a data frame shall contain the Frame Control field, the Sequence Number field, the destination VAN identifier/address fields, and/or the source VAN identifier/address fields.

In the Frame Control field, the Frame Type subfield shall contain the value that indicates a data frame, as shown in Table 60. If protection is used for the data, the Security Enabled subfield shall be set to one. The Frame Version subfield shall be set to one if either the Security Enabled subfield is set to one or the MAC Payload field is greater than *aMaxMACSafePayloadSize*. All other subfields shall be set appropriately according to the intended use of the data frame. All reserved subfields shall be set to zero and ignored on reception.

The Sequence Number field shall contain the current value of *macDSN*.

The addressing fields shall comprise the destination address fields and/or the source address fields, dependent on the settings in the Frame Control field.

The Auxiliary Security Header field, if present, shall contain the information required for security processing of the data frame, as specified in 7.2.1.7.

7.2.2.2.2 Data Payload field

The payload of a data frame shall contain the sequence of octets that the next higher layer has requested the MAC sublayer to transmit.

7.2.2.3 Acknowledgment frame format

The acknowledgment frame shall be formatted as illustrated in Figure 38.

Octets: 2	1	2
Frame Control	Sequence Number	FCS
MHR		MFR

Figure 38—Acknowledgment frame format

The order of the fields of the acknowledgment frame shall conform to the order of the general MAC frame as illustrated in Figure 26.

7.2.2.3.1 Acknowledgment frame MHR fields

The MHR for an acknowledgment frame shall contain only the Frame Control field and the Sequence Number field.

In the Frame Control field, the Frame Type subfield shall contain the value that indicates an acknowledgment frame, as shown in Table 60. If the acknowledgment frame is being sent in response to a received data request command, the device sending the acknowledgment frame shall determine whether it has data pending for the recipient. If the device can determine this before sending the acknowledgment frame (see 7.6.6.4.2), it shall set the Frame Pending subfield according to whether there is pending data. Otherwise, the Frame Pending subfield shall be set to one. If the acknowledgment frame is being sent in response to either a data frame or another type of MAC command frame, the device shall set the Frame Pending subfield to zero. All other subfields shall be set to zero and ignored on reception.

The Sequence Number field shall contain the value of the sequence number received in the frame for which the acknowledgment is to be sent.

7.2.2.4 MAC command frame format

The MAC command frame shall be formatted as illustrated in Figure 39.

Octets: 2	1	(see 7.2.2.4.1)	0/5/6/10/14	1	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Command Frame Identifier	Command Payload	FCS
MHR				MAC Payload		MFR

Figure 39—MAC command frame format

The order of the fields of the MAC command frame shall conform to the order of the general MAC frame as illustrated in Figure 26.

7.2.2.4.1 MAC command frame MHR fields

The MHR for a MAC command frame shall contain the Frame Control field, the Sequence Number field, the destination VAN identifier/address fields and/or the source VAN identifier/address fields.

In the Frame Control field, the Frame Type subfield shall contain the value that indicates a MAC command frame, as shown in Table 60. If the frame is to be secured, the Security Enabled subfield of the Frame

Control field shall be set to one and the frame secured according to the process described in 7.6.8.1.3. Otherwise the Security Enabled subfield of the Frame Control field shall be set to zero. All other subfields shall be set appropriately according to the intended use of the MAC command frame. All reserved subfields shall be set to zero and ignored on reception.

The Sequence Number field shall contain the current value of *macDSN*.

The addressing fields shall comprise the destination address fields and/or the source address fields, dependent on the settings in the Frame Control field.

The Auxiliary Security Header field, if present, shall contain the information required for security processing of the MAC command frame, as specified in 7.2.1.7.

7.2.2.4.2 Command Frame Identifier field

The Command Frame Identifier field identifies the MAC command being used. This field shall be set to one of the nonreserved values listed in Table 63.

7.2.2.4.3 Command Payload field

The Command Payload field contains the MAC command itself. The formats of the individual commands are described in 7.4.

7.3 Information elements

The information elements are listed in Table XX. Individual elements are described in the following subclauses.

Table 62—Information Elements

Element ID hex value	Element	Subclause	Present in beacon
0x00	Codeword	7.4.1	TBD

The format of an individual IE is shown in Figure XX. The first octet is the Element ID and the second octet is the Length (Ln) of the payload of the IE in octets. The following Ln octets are the payload for the IE. Unless otherwise specified, these elements may appear in any order in the frames that are allowed to include more than one of these elements.

Figure 40—Information element format

octets: Ln	1	1
IE payload	Length (=Ln)	Element ID

7.3.1 Codeword

The Codeword IE shall be formatted as illustrated in Figure XX.

For this Codeword IE, one of 64 orthogonal sixteen bit codewords is assigned. These sixteen bit codewords can be generated by the following tables and procedure.

64 16 bit codewords for the Codeword IE can be formed as shown in FigX. These codewords are formed by cascading two identical 8-bit orthogonal codes. Received data sequence is divided into two subcodewords. Each subcodeword is compared with 8 8-bit codewords. 64 combinations of codewords can be realized. Therefore up to 64 devices can be associated to a network.

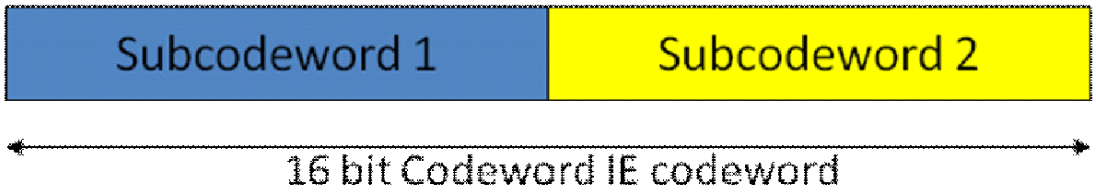


Figure 41—Cascading two identical codes to generate 16 bit codewords

For each subcodeword, the following table shown in Fig. is applied to form 16 bit codewords for Codeword IE codewords.

For subcodeword 1

RTS/ACK index	r0	r1	r2	r3	r4	r5	r6	r7
1	0	0	0	0	0	0	0	0
2	0	1	0	1	0	1	0	1
3	0	0	1	1	0	0	1	1
4	0	1	1	0	0	1	1	0
5	0	0	0	0	1	1	1	1
6	0	1	0	1	1	0	1	0
7	0	0	1	1	1	1	0	0
8	0	1	1	0	1	0	0	1

For subcodeword 2

RTS/ACK index	r8	r9	r10	r11	r12	r13	r14	r15
1	0	0	0	0	0	0	0	0
2	0	1	0	1	0	1	0	1
3	0	0	1	1	0	0	1	1
4	0	1	1	0	0	1	1	0
5	0	0	0	0	1	1	1	1
6	0	1	0	1	1	0	1	0
7	0	0	1	1	1	1	0	0
8	0	1	1	0	1	0	0	1

Any device who wants to be associated to a VAN selects one of 64 combinations of codewords randomly and sends it in the Codeword IE field of the Association Request command. If the coordinator receives this request and decides to allow this device to be associated with the VAN, it will send the same codeword that it received from that device to notify that this device is allowed to be associated with the network. If more than one device send request of association simultaneously, the coordinator selects one of them and sends exactly the same codeword among 64 combinations that was sent by the selected device in the Codeword IE of the Association Response command to notify that this device is allowed to be associated with the network. If more than one device selects the same codeword among 64 combinations with the same destination address and with different source address, collision occurs and random back-off algorithm is applied by resending request commands with random back-offs.

7.4 MAC command frames

The command frames defined by the MAC sublayer are listed in Table 63. An FFD shall be capable of transmitting and receiving all command frame types, with the exception of the GTS request command, while the requirements for an RFD are indicated by an "X" in the table. MAC commands shall only be transmitted in the CAP for beacon-enabled VANs or at any time for nonbeacon-enabled VANs.

How the MLME shall construct the individual commands for transmission is detailed in 7.4.1 through 7.4.10. MAC command reception shall abide by the procedure described in 7.6.6.2.

Table 63—MAC command frames

Command frame identifier	Command name	RFD		Subclause
		Tx	Rx	
0x01	Association request	X		7.4.1
0x02	Association response		X	7.4.2
0x03	Disassociation notification	X	X	7.4.3
0x04	Data request	X		7.4.4
0x05	VAN ID conflict notification	X		7.4.5
0x06	Orphan notification	X		7.4.6
0x07	Beacon request			7.4.7
0x08	Coordinator realignment		X	7.4.8
0x09	GTS request			7.4.10
0x0a–0xff	Reserved			—

7.4.1 Association request command

The association request command allows a device to request association with a VAN through the VAN coordinator or a coordinator.

This command shall only be sent by an unassociated device that wishes to associate with a VAN. A device shall only associate with a VAN through the VAN coordinator or a coordinator allowing association, as determined through the scan procedure.

All devices shall be capable of transmitting this command, although an RFD is not required to be capable of receiving it.

The association request command shall be formatted as illustrated in Figure 42.

7.4.1.1 MHR fields

The Source Addressing Mode subfield of the Frame Control field shall be set to three (64-bit extended addressing). The Destination Addressing Mode subfield shall be set to the same mode as indicated in the beacon frame to which the association request command refers.

octets: (see 7.2.2.4)	1	1	2
MHR fields	Command Frame Identifier (see Table 63)	Capability Information	Codeword IE

Figure 42—Association request command format

The Frame Pending subfield of the Frame Control field shall be set to zero and ignored upon reception, and the Acknowledgment Request subfield shall be set to one.

The Destination VAN Identifier field shall contain the identifier of the VAN to which to associate. The Destination Address field shall contain the address from the beacon frame that was transmitted by the coordinator to which the association request command is being sent. The Source VAN Identifier field shall contain the broadcast VAN identifier (i.e., 0xffff). The Source Address field shall contain the value of *aExtendedAddress*.

7.4.1.2 Capability Information field

The Capability Information field shall be formatted as illustrated in Figure 43.

bits: 0	1	2	3	4-5	6	7
Alternate VAN Coordinator	Device Type	Power Source	Receiver On When Idle	Reserved	Security Capability	Allocate Address

Figure 43—Capability Information field format

The Alternate VAN Coordinator subfield is 1 bit in length and shall be set to one if the device is capable of becoming the VAN coordinator. Otherwise, the Alternate VAN Coordinator subfield shall be set to zero.

The Device Type subfield is 1 bit in length and shall be set to one if the device is an FFD. Otherwise, the Device Type subfield shall be set to zero to indicate an RFD.

The Power Source subfield is 1 bit in length and shall be set to one if the device is receiving power from the alternating current mains. Otherwise, the Power Source subfield shall be set to zero.

The Receiver On When Idle subfield is 1 bit in length and shall be set to one if the device does not disable its receiver to conserve power during idle periods. Otherwise, the Receiver On When Idle subfield shall be set to zero.

The Security Capability subfield is 1 bit in length and shall be set to one if the device is capable of sending and receiving cryptographically protected MAC frames as specified in 7.6.8.2; it shall be set to zero otherwise.

The Allocate Address subfield is 1 bit in length and shall be set to one if the device wishes the coordinator to allocate a 16-bit short address as a result of the association procedure. Otherwise, it shall be set to zero.

7.4.1.3 Codeword IE

For this Codeword IE, one of 64 orthogonal sixteen bit codewords is assigned. These sixteen bit codewords can be generated by the same procedure as in 7.3.1. Operational procedure is also described in 7.3.1.3.

7.4.2 Association response command

The association response command allows the VAN coordinator or a coordinator to communicate the results of an association attempt back to the device requesting association.

This command shall only be sent by the VAN coordinator or coordinator to a device that is currently trying to associate.

All devices shall be capable of receiving this command, although an RFD is not required to be capable of transmitting it.

The association response command shall be formatted as illustrated in Figure 44.

octets: (see 7.2.2.4)	**1**	**2**	**1**	**2**
MHR fields	Command Frame Identifier (see Table 63)	Short Address	Association Status	Codeword IE

Figure 44—Association response command format

7.4.2.1 MHR fields

The Destination Addressing Mode and Source Addressing Mode subfields of the Frame Control field shall each be set to three (i.e., 64-bit extended addressing).

The Frame Pending subfield of the Frame Control field shall be set to zero and ignored upon reception, and the Acknowledgment Request subfield shall be set to one.

The VAN ID Compression subfield of the Frame Control field shall be set to one. In accordance with this value of the VAN ID Compression subfield, the Destination VAN Identifier field shall contain the value of *macVANId*, while the Source VAN Identifier field shall be omitted. The Destination Address field shall contain the extended address of the device requesting association. The Source Address field shall contain the value of *aExtendedAddress*.

7.4.2.2 Short Address field

If the coordinator was not able to associate this device to its VAN, the Short Address field shall be set to 0xffff, and the Association Status field shall contain the reason for the failure. If the coordinator was able to associate the device to its VAN, this field shall contain the short address that the device may use in its communications on the VAN until it is disassociated.

A Short Address field value equal to 0xfffe shall indicate that the device has been successfully associated with a VAN, but has not been allocated a short address. In this case, the device shall communicate on the VAN using only its 64-bit extended address.

7.4.2.3 Association Status field

The Association Status field shall contain one of the nonreserved values listed in Table 64.

Table 64—Valid values of the Association Status field

Association status	Description
0x00	Association successful.
0x01	VAN at capacity.
0x02	VAN access denied.
0x03–0x7f	Reserved.
0x80–0xff	Reserved for MAC primitive enumeration values.

7.4.2.4 Codeword IE

For this Codeword IE, one of 64 orthogonal sixteen bit codewords is assigned. These sixteen bit codewords can be generated by the same procedure as in 7.3.1. Operational procedure is also described in 7.3.1.3.

7.4.3 Disassociation notification command

The VAN coordinator, a coordinator, or an associated device may send the disassociate notification command.

All devices shall implement this command.

The disassociation notification command shall be formatted as illustrated in Figure 45.

octets: (see 7.2.2.4)	1	1
MHR fields	Command Frame Identifier (see Table 63)	Disassociation Reason

Figure 45—Disassociation notification command format

7.4.3.1 MHR fields

The Destination Addressing Mode subfield of the Frame Control field shall be set according to the addressing mode specified by the corresponding primitive. The Source Addressing Mode subfield shall be set to three (i.e., 64-bit extended addressing).

The Frame Pending subfield of the Frame Control field shall be set to zero and ignored upon reception, and the Acknowledgment Request subfield shall be set to one.

The VAN ID Compression subfield of the Frame Control field shall be set to one. In accordance with this value of the VAN ID Compression subfield, the Destination VAN Identifier field shall contain the value of *macVANId*, while the Source VAN Identifier field shall be omitted. If the coordinator wants an associated device to leave the VAN, then the Destination Address field shall contain the address of the device being

removed from the VAN. If an associated device wants to leave the VAN, then the Destination Address field shall contain the value of either *macCoordShortAddress*, if the Destination Addressing Mode subfield is equal to two, or *macCoordExtendedAddress*, if the Destination Addressing Mode subfield is equal to three. The Source Address field shall contain the value of *aExtendedAddress*.

7.4.3.2 Disassociation Reason field

The Disassociation Reason field shall contain one of the nonreserved values listed in Table 65.

Table 65—Valid disassociation reason codes

Disassociate reason	Description
0x00	Reserved.
0x01	The coordinator wishes the device to leave the VAN.
0x02	The device wishes to leave the VAN.
0x03–0x7f	Reserved.
0x80–0xff	Reserved for MAC primitive enumeration values.

7.4.4 Data request command

The data request command is sent by a device to request data from the VAN coordinator or a coordinator.

There are three cases for which this command is sent. On a beacon-enabled VAN, this command shall be sent by a device when *macAutoRequest* is equal to TRUE and a beacon frame indicating that data are pending for that device is received from its coordinator. The coordinator indicates pending data in its beacon frame by adding the address of the recipient of the data to the Address List field. This command shall also be sent when instructed to do so by the next higher layer on reception of the MLME-POLL.request primitive. In addition, a device may send this command to the coordinator *macResponseWaitTime* symbols after the acknowledgment to an association request command.

All devices shall be capable of transmitting this command, although an RFD is not required to be capable of receiving it.

The data request command shall be formatted as illustrated in Figure 46.

octets: (see 7.2.2.4)	1
MHR fields	Command Frame Identifier (see Table 63)

Figure 46—Data request command format

If the data request command is being sent in response to the receipt of a beacon frame indicating that data are pending for that device, the Destination Addressing Mode subfield of the Frame Control field may be set to zero (i.e., destination addressing information not present) if the beacon frame indicated in its Superframe Specification field (see 7.2.2.1.2) that it originated from the VAN coordinator (see 7.2.1.1.6) or set otherwise according to the coordinator to which the data request command is directed. If the destination

addressing information is to be included, the Destination Addressing Mode subfield shall be set according to the value of *macCoordShortAddress*. If *macCoordShortAddress* is equal to 0xfffe, extended addressing shall be used: the Destination Addressing Mode subfield shall be set to three, and the Destination Address field shall contain the value of *macCoordExtendedAddress*. Otherwise, short addressing shall be used: the Destination Addressing Mode subfield shall be set to two, and the Destination Address field shall contain the value of *macCoordShortAddress*.

If the data request command is being sent in response to the receipt of a beacon frame indicating that data are pending for that device, the Source Addressing Mode subfield shall be set according to the addressing mode used for the pending address. If the Source Addressing Mode subfield is set to two, short addressing shall be used: the Source Address field shall contain the value of *macShortAddress*. Otherwise, extended addressing shall be used: the Source Addressing Mode subfield shall be set to three, and the Source Address field shall contain the value of *aExtendedAddress*.

If the data request command is triggered by the reception of an MLME-POLL.request primitive from the next higher layer, then the destination addressing information shall be the same as that contained in the primitive. The Source Addressing Mode subfield shall be set according to the value of *macShortAddress*. If *macShortAddress* is less than 0xfffe, short addressing shall be used. Extended addressing shall be used otherwise.

If the data request command is being sent following the acknowledgment to an association request command frame, the Destination Addressing Mode subfield of the Frame Control field shall be set according to the coordinator to which the data request command is directed. If *macCoordShortAddress* is equal to 0xfffe, extended addressing shall be used. Short addressing shall be used otherwise. The Source Addressing Mode subfield shall be set to use extended addressing.

If the Destination Addressing Mode subfield is set to zero (i.e., destination addressing information not present), the VAN ID Compression subfield of the Frame Control field shall be set to zero and the source VAN identifier shall contain the value of *macVANId*. Otherwise, the VAN ID Compression subfield shall be set to one. In this case and in accordance with the VAN ID Compression subfield, the Destination VAN Identifier field shall contain the value of *macVANId*, while the Source VAN Identifier field shall be omitted.

The Frame Pending subfield of the Frame Control field shall be set to zero and ignored upon reception, and the Acknowledgment Request subfield shall be set to one.

7.4.5 VAN ID conflict notification command

The VAN ID conflict notification command is sent by a device to the VAN coordinator when a VAN identifier conflict is detected.

All devices shall be capable of transmitting this command, although an RFD is not required to be capable of receiving it.

The VAN ID conflict notification command shall be formatted as illustrated in Figure 47.

octets: (see 7.2.2.4)	**1**
MHR fields	Command Frame Identifier (see Table 63)

Figure 47—VAN ID conflict notification command format

The Destination Addressing Mode and Source Addressing Mode subfields of the Frame Control field shall both be set to three (i.e., 64-bit extended addressing).

The Frame Pending subfield of the Frame Control field shall be set to zero and ignored upon reception, and the Acknowledgment Request subfield shall be set to one.

The VAN ID Compression subfield of the Frame Control field shall be set to one. In accordance with this value of the VAN ID Compression subfield, the Destination VAN Identifier field shall contain the value of *macVANId*, while the Source VAN Identifier field shall be omitted. The Destination Address field shall contain the value of *macCoordExtendedAddress*. The Source Address field shall contain the value of *aExtendedAddress*.

7.4.6 Orphan notification command

The orphan notification command is used by an associated device that has lost synchronization with its coordinator.

All devices shall be capable of transmitting this command, although an RFD is not required to be capable of receiving it.

The orphan notification command shall be formatted as illustrated in Figure 48.

octets: 15	**1**
MHR fields	Command Frame Identifier (see Table 63)

Figure 48—Orphan notification command format

The Source Addressing Mode subfield of the Frame Control field shall be set to three (i.e., 64-bit extended addressing). The Destination Addressing Mode subfield shall be set to two (i.e., 16-bit short addressing).

The Frame Pending subfield and Acknowledgment Request subfield of the Frame Control field shall be set to zero and ignored upon reception.

The VAN ID Compression subfield of the Frame Control field shall be set to one. In accordance with this value of the VAN ID Compression subfield, the Destination VAN Identifier field shall contain the value of the broadcast VAN identifier (i.e., 0xffff), while the Source VAN Identifier field shall be omitted. The Destination Address field shall contain the broadcast short address (i.e., 0xffff). The Source Address field shall contain the value of *aExtendedAddress*.

7.4.7 Beacon request command

The beacon request command is used by a device to locate all coordinators within its POS during an active scan.

This command is optional for an RFD.

The beacon request command shall be formatted as illustrated in Figure 49.

octets: 7	1
MHR fields	Command Frame Identifier (see Table 63)

Figure 49—Beacon request command format

The Destination Addressing Mode subfield of the Frame Control field shall be set to two (i.e., 16-bit short addressing), and the Source Addressing Mode subfield shall be set to zero (i.e., source addressing information not present).

The Frame Pending subfield of the Frame Control field shall be set to zero and ignored upon reception. The Acknowledgment Request subfield and Security Enabled subfield shall also be set to zero.

The Destination VAN Identifier field shall contain the broadcast VAN identifier (i.e., 0xffff). The Destination Address field shall contain the broadcast short address (i.e., 0xffff).

7.4.8 Coordinator realignment command

The coordinator realignment command is sent by the VAN coordinator or a coordinator either following the reception of an orphan notification command from a device that is recognized to be on its VAN or when any of its VAN configuration attributes change due to the receipt of an MLME-START.request primitive.

If this command is sent following the reception of an orphan notification command, it is sent directly to the orphaned device. If this command is sent when any VAN configuration attributes (i.e., VAN identifier, short address, logical channel, or channel page) change, it is broadcast to the VAN.

All devices shall be capable of receiving this command, although an RFD is not required to be capable of transmitting it.

The coordinator realignment command shall be formatted as illustrated in Figure 50.

octets: 17/18/23/24	1	2	2	1	2	0/1
MHR fields	Command Frame Identifier (see Table 63)	VAN Identifier	Coordinator Short Address	Logical Channel	Short Address	Channel page

Figure 50—Coordinator realignment command format

7.4.8.1 MHR fields

The Destination Addressing Mode subfield of the Frame Control field shall be set to three (e.g., 64-bit extended addressing) if the command is directed to an orphaned device or set to two (e.g., 16-bit short addressing) if it is to be broadcast to the VAN. The Source Addressing Mode subfield of the Frame Control field shall be set to three (e.g., 64-bit extended addressing).

The Frame Pending subfield of the Frame Control field shall be set to zero and ignored upon reception.

The Acknowledgment Request subfield of the Frame Control field shall be set to one if the command is directed to an orphaned device or set to zero if the command is to be broadcast to the VAN.

The Frame Version subfield shall be set to 0x01 if the Channel Page field is present. Otherwise it shall be set as specified in 7.2.3.

The Destination VAN Identifier field shall contain the broadcast VAN identifier (e.g., 0xffff). The Destination Address field shall contain the extended address of the orphaned device if the command is directed to an orphaned device. Otherwise, the Destination Address field shall contain the broadcast short address (e.g., 0xffff). The Source VAN Identifier field shall contain the value of *macVANId*, and the Source Address field shall contain the value of *aExtendedAddress*.

7.4.8.2 VAN Identifier field

The VAN Identifier field shall contain the VAN identifier that the coordinator intends to use for all future communications.

7.4.8.3 Coordinator Short Address field

The Coordinator Short Address field shall contain the value of *macShortAddress*.

7.4.8.4 Logical Channel field

The Logical Channel field shall contain the logical channel that the coordinator intends to use for all future communications.

7.4.8.5 Short Address field

If the coordinator realignment command is broadcast to the VAN, the Short Address field shall be set to 0xffff and ignored on reception.

If the coordinator realignment command is sent directly to an orphaned device, this field shall contain the short address that the orphaned device shall use to operate on the VAN. If the orphaned device does not have a short address, because it always uses its 64-bit extended address, this field shall contain the value 0xfffe.

7.4.8.6 Channel Page field

The Channel Page field, if present, shall contain the channel page that the coordinator intends to use for all future communications. This field may be omitted if the new channel page is the same as the previous channel page.

7.4.9 GTS request command

The GTS request command is used by an associated device that is requesting the allocation of a new GTS or the deallocation of an existing GTS from the VAN coordinator. Only devices that have a 16-bit short address less than 0xfffe shall send this command.

This command is optional.

The GTS request command shall be formatted as illustrated in Figure 53.

octets: 7	1	1	2
MHR fields	Command Frame Identifier (see Table 63)	GTS Characteristics	Codeword IE

Figure 51—GTS request command format

7.4.9.1 MHR fields

The Destination Addressing Mode subfield of the Frame Control field shall be set to zero (e.g., destination addressing information not present), and the Source Addressing Mode subfield shall be set to two (e.g., 16-bit short addressing).

The Frame Pending subfield of the Frame Control field shall be set to zero and ignored upon reception, and the Acknowledgment Request subfield shall be set to one.

The Source VAN Identifier field shall contain the value of *macVANId*, and the Source Address field shall contain the value of *macShortAddress*.

7.4.9.2 GTS Characteristics field

The GTS Characteristics field shall be formatted as illustrated in Figure 54.

bits: 0–3	4	5	6–7
GTS Length	GTS Direction	Characteristics Type	Reserved

Figure 52—GTS Characteristics field format

The GTS Length subfield shall contain the number of superframe slots being requested for the GTS.

The GTS Direction subfield shall be set to one if the GTS is to be a receive-only GTS. Conversely, this subfield shall be set to zero if the GTS is to be a transmit-only GTS. GTS direction is defined relative to the direction of data frame transmissions by the device.

The Characteristics Type subfield shall be set to one if the characteristics refers to a GTS allocation or zero if the characteristics refers to a GTS deallocation.

7.4.9.3 Codeword IE

For this Codeword IE, one of 64 orthogonal sixteen bit codewords is assigned. These sixteen bit codewords can be generated by the same procedure as in 7.3.1.

Any device who wants to request a GTS slot selects one of 64 combinations of codewords randomly and sends it in the Codeword IE field of the GTS Request command. If the coordinator receives this request and decides to allocate a GTS slot to this device, it will send the same codeword that it received from that device to notify that a GTS slot is allocated to the device. If more than one device send request of association simultaneously, the VAN coordinator selects one of them and sends exactly the same codeword among 64 combinations that was sent by the selected device in the Codeword IE of the GTS Response command to notify that this device is allowed to send a frame in the allocated GTS slot. If more than one device selects

the same codeword among 64 combinations with different source address, collision occurs and random back-off algorithm is applied by resending request commands with random back-offs.

7.4.10 GTS response command

The GTS.response primitive is used generated in response to a GTS.requirest primitve.

This command is optional.

The GTS response command shall be formatted as illustrated in Figure 51.

octets: 7	1	1	1	2
MHR fields	Command Frame Identifier (see Table 63)	GTS Characteristics	GTS Starting Slot	Codeword IE

Figure 53—GTS response command format

7.4.10.1 MHR fields

The Destination Addressing Mode subfield of the Frame Control field shall be set to zero (e.g., destination addressing information not present), and the Source Addressing Mode subfield shall be set to two (e.g., 16-bit short addressing).

The Frame Pending subfield of the Frame Control field shall be set to zero and ignored upon reception, and the Acknowledgment Request subfield shall be set to one.

The Source VAN Identifier field shall contain the value of *macVANId*, and the Source Address field shall contain the value of *macShortAddress*.

7.4.10.2 GTS Characteristics field

The GTS Characteristics field shall be formatted as illustrated in Figure 54.

bits: 0–3	4	5	6–7
GTS Length	GTS Direction	Characteristics Type	Reserved

Figure 54—GTS Characteristics field format

The GTS Length subfield shall contain the number of superframe slots being requested for the GTS.

The GTS Direction subfield shall be set to one if the GTS is to be a receive-only GTS. Conversely, this subfield shall be set to zero if the GTS is to be a transmit-only GTS. GTS direction is defined relative to the direction of data frame transmissions by the device.

The Characteristics Type subfield shall be set to one if the characteristics refers to a GTS allocation or zero if the characteristics refers to a GTS deallocation.

7.4.10.3 GTS Starting Slot

TBD

7.4.10.4 Codeword IE

For this Codeword IE, one of 64 orthogonal sixteen bit codewords is assigned. These sixteen bit codewords can be generated by the same procedure as in 7.3.1. Operational procedure is also described in 7.4.9.3.

7.5 MAC constants and PIB attributes

This subclause specifies the constants and attributes required by the MAC sublayer.

7.5.1 MAC constants

The constants that define the characteristics of the MAC sublayer are presented in Table 66.

Table 66—MAC sublayer constants

Constant	Description	Value
aBaseSlotDuration	The number of symbols forming a superframe slot when the superframe order is equal to 0 (see 7.6.1.1).	60
aBaseSuperframeDuration	The number of symbols forming a superframe when the superframe order is equal to 0.	*aBaseSlotDuration* * *aNumSuperframeSlots*
aExtendedAddress	The 64-bit (IEEE) address assigned to the device.	Device specific
aGTSDescPersistenceTime	The number of superframes in which a GTS descriptor exists in the beacon frame of the VAN coordinator.	4
aMaxBeaconOverhead	The maximum number of octets added by the MAC sublayer to the MAC payload of a beacon frame.	75

Table 66—MAC sublayer constants *(continued)*

Constant	Description	Value
aMaxBeaconPayloadLength	The maximum size, in octets, of a beacon payload.	*aMaxPHYPacketSize* – *aMaxBeaconOverhead*
aMaxLostBeacons	The number of consecutive lost beacons that will cause the MAC sublayer of a receiving device to declare a loss of synchronization.	4
aMaxMACSafePayloadSize	The maximum number of octets that can be transmitted in the MAC Payload field of an unsecured MAC frame that will be guaranteed not to exceed *aMaxPHYPacketSize*.	*aMaxPHYPacketSize* – *aMaxMPDUUnsecuredOverhead*
aMaxMACPayloadSize	The maximum number of octets that can be transmitted in the MAC Payload field.	*aMaxPHYPacketSize* – *aMinMPDUOverhead*
aMaxMPDUUnsecuredOverhead	The maximum number of octets added by the MAC sublayer to the PSDU without security.	25
aMaxSIFSFrameSize	The maximum size of an MPDU, in octets, that can be followed by a SIFS period.	18
aMinCAPLength	The minimum number of symbols forming the CAP. This ensures that MAC commands can still be transferred to devices when GTSs are being used. An exception to this minimum shall be allowed for the accommodation of the temporary increase in the beacon frame length needed to perform GTS maintenance (see 7.2.2.1.3).	440
aMinMPDUOverhead	The minimum number of octets added by the MAC sublayer to the PSDU.	9
aNumSuperframeSlots	The number of slots contained in any superframe.	16
aUnitBackoffPeriod	The number of symbols forming the basic time period used by the CSMA-CA algorithm.	20

7.5.2 MAC PIB attributes

The MAC PIB comprises the attributes required to manage the MAC sublayer of a device. The attributes contained in the MAC PIB are presented in Table 31. Attributes marked with a dagger (†) are read-only attributes (i.e., attribute can only be set by the MAC sublayer), which can be read by the next higher layer using the MLME-GET.request primitive. All other attributes can be read or written by the next higher layer using the MLME-GET.request or MLME-SET.request primitives, respectively. Attributes marked with a diamond (◆) are optional for an RFD; attributes marked with an asterisk (*) are optional for both device types (i.e., RFD and FFD).

The read-only attribute *macAckWaitDuration* is dependent on a combination of constants and PHY PIB attributes. The formula for relating the constants and attributes is shown in Equation (2).

$$macAckWaitDuration = aUnitBackoffPeriod + aTurnaroundTime + phySHRDuration + \lceil 6 \cdot phySymbolsPerOctet \rceil \quad (2)$$

where

6 represents the number of PHY header octets plus the number of PSDU octets in an acknowledgment frame.

The attribute *macMaxFrameTotalWaitTime* may be set by the next higher layer and is dependent upon a combination of PHY and MAC PIB attributes and constants. The formula relating the attributes and constants is shown in Equation (3):

macMaxFrameTotalWaitTime = (3)

$$\left[\left(\sum_{k=0}^{m-1} 2^{macMinBE+k}\right) + (2^{macMaxBE} - 1) \cdot (macMaxCSMABackoffs - m)\right] \bullet aUnitBackoffPeriod + phyMaxFrameDuration$$

where

m is min(*macMaxBE-macMinBE, macMaxCSMABackoffs).*

For the mandatory data rate (1 Mb/s), *macAckWaitDuration* is calculated as shown in Equation (6).

$$macAckWaitDuration_{1M} = aUnitBackoffPeriod + aTurnaroundTime + phySHRDuration_{1M} + [1.5 + 3/4 \times ceiling(4/3 \times 5)] \times phySymbolsPerOctet_{1M} \quad (4)$$

For the optional data rate (250 kb/s), *macAckWaitDuration* is calculated as shown in Equation (7).

$$macAckWaitDuration_{250k} = aUnitBackoffPeriod + aTurnaroundTime + phySHRDuration_{250k} + 3 \times ceiling(1/3 \times [1.5 + 5]) \times phySymbolsPerOctet_{250k} \quad (5)$$

Table 67—MAC PIB attributes

Attribute	Identifier	Type	Range	Description	Default
macAckWaitDuration†	0x40	Integer	See Equation (2), Equation (4), Equation (6), and Equation (7) for range as a function of PHY type	The maximum number of symbols to wait for an acknowledgment frame to arrive following a transmitted data frame. This value is dependent on the supported PHY, which determines both the selected logical channel and channel page. The calculated value is the time to commence transmitting the ACK plus the length of the ACK frame. The commencement time is described in 7.6.6.4.2.	Dependent on currently selected PHY, indicated by *phyCurrentPage*.
macAssociatedVAN-Coord	0x56	Boolean	TRUE or FALSE	Indication of whether the device is associated to the VAN through the VAN coordinator. A value of TRUE indicates the device has associated through the VAN coordinator. Otherwise, the value is set to FALSE.	FALSE

Table 67—MAC PIB attributes *(continued)*

Attribute	Identifier	Type	Range	Description	Default
macAssociation-Permit◆	0x41	Boolean	TRUE or FALSE	Indication of whether a coordinator is currently allowing association. A value of TRUE indicates that association is permitted.	FALSE
macAutoRequest	0x42	Boolean	TRUE or FALSE	Indication of whether a device automatically sends a data request command if its address is listed in the beacon frame. A value of TRUE indicates that the data request command is automatically sent. This attribute also affects the generation of the MLME-BEACON-NOTIFY.indication primitive (see 7.1.5.1.2).	TRUE
macBattLifeExt	0x43	Boolean	TRUE or FALSE	Indication of whether BLE, through the reduction of coordinator receiver operation time during the CAP, is enabled. A value of TRUE indicates that it is enabled. Also, see 7.6.1.4 for an explanation of how this attribute affects the backoff exponent in the CSMA-CA algorithm.	FALSE

Table 67—MAC PIB attributes *(continued)*

Attribute	Identifier	Type	Range	Description	Default
macBattLifeExtPeriods	0x44	Integer	6-41	In BLE mode, the number of backoff periods during which the receiver is enabled after the IFS following a beacon. This value is dependent on the supported PHY and is the sum of three terms: Term 1: The value $2^x - 1$, where x is the maximum value of *macMinBE* in BLE mode (equal to two). This term is thus equal to 3 backoff periods. Term 2: The duration of the initial contention window length (see 7.6.1.4). This term is thus equal to 2 backoff periods. Term 3: The Preamble field length and the SFD field length of the supported PHY (see Table 26 and Table 27 in Clause 6), summed together and rounded up (if necessary) to an integer number of back-off periods.	Dependent on currently selected PHY, indicated by *phy-Current-Page*
macBeaconPayload◆	0x45	Set of octets	—	The contents of the beacon payload.	NULL
macBeaconPayload-Length◆	0x46	Integer	0 – *aMax-Beacon-PayloadLength*	The length, in octets, of the beacon payload.	0
macBeaconOrder◆	0x47	Integer	0–15	Specification of how often the coordinator transmits its beacon. If *BO* = 15, the coordinator will not transmit a periodic beacon. See 7.6.1.1 for an explanation of the relationship between the beacon order and the beacon interval.	15

Table 67—MAC PIB attributes *(continued)*

Attribute	Identifier	Type	Range	Description	Default
macBeaconTxTime†◆	0x48	Integer	0x000000–0xffffff	The time that the device transmitted its last beacon frame, in symbol periods. The measurement shall be taken at the same symbol boundary within every transmitted beacon frame, the location of which is implementation specific. This is a 24-bit value, and the precision of this value shall be a minimum of 20 bits, with the lowest four bits being the least significant.	0x000000
macBSN◆	0x49	Integer	0x00–0xff	The sequence number added to the transmitted beacon frame.	Random value from within the range
macCoordExtended-Address	0x4a	IEEE address	An extended 64-bit IEEE address	The 64-bit address of the coordinator through which the device is associated.	—
macCoordShort-Address	0x4b	Integer	0x0000–0xffff	The 16-bit short address assigned to the coordinator through which the device is associated. A value of 0xfffe indicates that the coordinator is only using its 64-bit extended address. A value of 0xffff indicates that this value is unknown.	0xffff
macDSN	0x4c	Integer	0x00–0xff	The sequence number added to the transmitted data or MAC command frame.	Random value from within the range
*macGTSPermit**	0x4d	Boolean	TRUE or FALSE	TRUE if the VAN coordinator is to accept GTS requests. FALSE otherwise.	TRUE
macMaxBE	0x57	Integer	3–15	The maximum value of the backoff exponent, BE, in the random access algorithm. See 7.6.1.4 for a detailed explanation of the backoff exponent.	5
macMaxCSMABackoffs	0x4e	Integer	0–5	The maximum number of backoffs the CSMA-CA algorithm will attempt before declaring a channel access failure.	4

Table 67—MAC PIB attributes *(continued)*

Attribute	Identifier	Type	Range	Description	Default
macMaxFrameTotal-WaitTime	0x58	Integer	See Equation (3) and Equation (5)	The maximum number of CAP symbols in a beacon-enabled VAN, or symbols in a nonbeacon-enabled VAN, to wait either for a frame intended as a response to a data request frame or for a broadcast frame following a beacon with the Frame Pending subfield set to one. This attribute, which shall only be set by the next higher layer, is dependent upon *macMinBE*, *macMaxBE*, *macMaxCSMABackoffs* and the number of symbols per octet. See 7.5.2 for the formula relating the attributes.	Dependent on currently selected PHY, indicated by *phy-Current-Page*
macMaxFrameRetries	0x59	Integer	0–7	The maximum number of retries allowed after a transmission failure.	3
macMinBE	0x4f	Integer	0–*macMaxBE*	The minimum value of the backoff exponent (BE) in the CSMA-CA algorithm. See 7.6.1.4 for a detailed explanation of the backoff exponent.	3
macMinLIFSPeriod†		Integer	See Table 5 in Clause 6	The minimum number of symbols forming a LIFS period.	Dependent on currently selected PHY, indicated by *phy-Current-Page*
macMinSIFSPeriod†		Integer	See Table 5 in Clause 6	The minimum number of symbols forming a SIFS period.	Dependent on currently selected PHY, indicated by *phy-Current-Page*
macVANId	0x50	Integer	0x0000–0xffff	The 16-bit identifier of the VAN on which the device is operating. If this value is 0xffff, the device is not associated.	0xffff

Table 67—MAC PIB attributes *(continued)*

Attribute	Identifier	Type	Range	Description	Default
macPromiscuous-Mode◆	0x51	Boolean	TRUE or FALSE	Indication of whether the MAC sublayer is in a promiscuous (receive all) mode. A value of TRUE indicates that the MAC sublayer accepts all frames received from the PHY.	FALSE
macRanging-Supported †	0x60	Boolean	TRUE or FALSE	This indicates whether the MAC sublayer supports the optional ranging features*.	Dependent on supported PHY and MAC capability.
macResponseWaitTime	0x5a	Integer	2–64	The maximum time, in multiples of *aBaseSuperframeDuration*, a device shall wait for a response command frame to be available following a request command frame.	32
macRxOnWhenIdle	0x52	Boolean	TRUE or FALSE	Indication of whether the MAC sublayer is to enable its receiver during idle periods. For a beacon-enabled VAN, this attribute is relevant only during the CAP of the incoming superframe. For a nonbeacon-enabled VAN, this attribute is relevant at all times.	FALSE
macSecurityEnabled	0x5d	Boolean	TRUE or FALSE	Indication of whether the MAC sublayer has security enabled. A value of TRUE indicates that security is enabled, while a value of FALSE indicates that security is disabled.	FALSE
macShortAddress	0x53	Integer	0x0000–0xffff	The 16-bit address that the device uses to communicate in the VAN. If the device is the VAN coordinator, this value shall be chosen before a VAN is started. Otherwise, the address is allocated by a coordinator during association. A value of 0xfffe indicates that the device has associated but has not been allocated an address. A value of 0xffff indicates that the device does not have a short address.	0xffff

Table 67—MAC PIB attributes *(continued)*

Attribute	Identifier	Type	Range	Description	Default
macSuperframe-Order†◆	0x54	Integer	0–15	The length of the active portion of the outgoing superframe, including the beacon frame. If superframe order, *SO*, = 15, the superframe will not be active following the beacon. See 7.6.1.1 for an explanation of the relationship between the superframe order and the superframe duration.	15
macSyncSymbolOffset†	0x5b	Integer	0x000–0x100 for the 2.4 GHz PHY 0x000–0x400 for the 868/915 MHz PHY	The offset, measured in symbols, between the symbol boundary at which the MLME captures the timestamp of each transmitted or received frame, and the onset of the first symbol past the SFD, namely, the first symbol of the Length field.	Imple-mentation specific
macTimestamp-Supported†	0x5c	Boolean	TRUE or FALSE	Indication of whether the MAC sublayer supports the optional timestamping feature for incoming and outgoing data frames.	Imple-mentation specific
macTransaction-PersistenceTime◆	0x55	Integer	0x0000–0xffff	The maximum time (in unit periods) that a transaction is stored by a coordinator and indicated in its beacon. The unit period is governed by *macBeaconOrder*, *BO*, as follows: For $0 \le BO \le 14$, the unit period will be *aBase-SuperframeDuration* * 2^{BO}. For *BO = 15*, the unit period will be *aBaseSuperframe-Duration*.	0x01f4
macTxControlActiveDuration	0x61	Integer	0–100000	The duration for which transmit is permitted without pause specified in symbols.	TDB
macTxControlPauseDuration	0x62	Integer	2000 or 10000	The duration after transmission before another transmission is permitted specified in symbols.	TBD
macTxTotalDuration	0x63	Integer	0x0–0xffffffff	The total transmit duration (including PHY header and FCS) specified in symbols. This can be read and cleared by NHL.	0

7.6 MAC functional description

This subclause provides a detailed description of the MAC functionality. Subclause 7.6.1 describes the following two mechanisms for channel access: contention based and contention free. Contention-based access allows devices to access the channel in a distributed fashion using a random access backoff algorithm. Contention-free access is controlled entirely by the VAN coordinator through the use of GTSs.

The mechanisms used for starting and maintaining a VAN are described in 7.6.2. Channel scanning is used by a device to assess the current state of a channel (or channels), locate all beacons within its POS, or locate a particular beacon with which it has lost synchronization. Before starting a new VAN, the results of a channel scan can be used to select an appropriate logical channel and channel page, as well as a VAN identifier that is not being used by any other VAN in the area. Because it is still possible for the POS of two VANs with the same VAN identifier to overlap, a procedure exists to detect and resolve this situation. Following a channel scan and suitable VAN identifier selection, an FFD can begin operating as the VAN coordinator. Also described in the subclause is a method to allow a beaconing FFD to discover other such devices during normal operations, i.e., when not scanning.

The mechanisms to allow devices to join or leave a VAN are defined in 7.6.3. The association procedure describes the conditions under which a device may join a VAN and the conditions necessary for a coordinator to permit devices to join. Also described is the disassociation procedure, which can be initiated by the associated device or its coordinator.

The mechanisms to allow devices to acquire and maintain synchronization with a coordinator are described in 7.6.4. Synchronization on a beacon-enabled VAN is described after first explaining how a coordinator generates beacon frames. Following this explanation, synchronization on a nonbeacon-enabled VAN is described. Also described is a procedure to reestablish communication between a device and its coordinator, as it is possible that a device may lose synchronization in the case of either a beacon-enabled or a nonbeacon-enabled VAN.

This standard has been designed so that application data transfers can be controlled by the devices on a VAN rather than by the coordinator. The procedures the coordinator uses to handle multiple transactions while preserving this requirement are described in 7.6.5.

The mechanisms for transmitting, receiving, and acknowledging frames, including frames sent using indirect transmission, are described in 7.6.6. In addition, methods for retransmitting frames are also described.

The mechanisms for allocating and deallocating a GTS are described in 7.6.7. The deallocation process may result in the fragmentation of the GTS space, i.e., an unused slot or slots. The subclause describes a mechanism to resolve fragmentation.

The MAC sublayer uses the mechanisms described in 7.6.8 for all incoming and outgoing frames.

Throughout this subclause, the receipt of a frame is defined as the successful receipt of the frame by the PHY and the successful verification of the FCS by the MAC sublayer, as described in 7.2.1.9.

7.6.1 Channel access

This subclause describes the mechanisms for accessing the physical radio channel.

7.6.1.1 Superframe structure

A coordinator on a VAN can optionally bound its channel time using a superframe structure. A superframe is bounded by the transmission of a beacon frame and can have an active portion and an inactive portion. The coordinator may enter a low-power (sleep) mode during the inactive portion.

The structure of this superframe is described by the values of *macBeaconOrder* and *macSuperframeOrder*. The MAC PIB attribute *macBeaconOrder*, describes the interval at which the coordinator shall transmit its beacon frames. The value of *macBeaconOrder*, *BO*, and the beacon interval, *BI*, are related as follows: for $0 \leq BO \leq 14$, $BI = aBaseSuperframeDuration * 2^{BO}$ symbols. If $BO = 15$, the coordinator shall not transmit beacon frames except when requested to do so, such as on receipt of a beacon request command. The value of *macSuperframeOrder* shall be ignored if $BO = 15$.

The MAC PIB attribute *macSuperframeOrder* describes the length of the active portion of the superframe, which includes the beacon frame. The value of *macSuperframeOrder*, *SO*, and the superframe duration, *SD*, are related as follows: for $0 \leq SO \leq BO \leq 14$, $SD = aBaseSuperframeDuration * 2^{SO}$ symbols. If $SO = 15$, the superframe shall not remain active after the beacon. If $BO = 15$, the superframe shall not exist (the value of *macSuperframeOrder* shall be ignored), and *macRxOnWhenIdle* shall define whether the receiver is enabled during periods of transceiver inactivity.

The active portion of each superframe shall be divided into *aNumSuperframeSlots* equally spaced slots of duration $2^{SO} * aBaseSlotDuration$ and is composed of three parts: a beacon, a CAP and a CFP. The beacon shall be transmitted, without the use of random access, at the start of slot 0, and the CAP shall commence immediately following the beacon. The start of slot 0 is defined as the point at which the first symbol of the beacon PPDU is transmitted. The CFP, if present, follows immediately after the CAP and extends to the end of the active portion of the superframe. Any allocated GTSs shall be located within the CFP.

The MAC sublayer shall ensure that the integrity of the superframe timing is maintained, e.g., compensating for clock drift error.

VANs that wish to use the superframe structure (referred to as a beacon-enabled VAN) shall set *macBeaconOrder* to a value between 0 and 14, both inclusive, and *macSuperframeOrder* to a value between 0 and the value of *macBeaconOrder*, both inclusive.

VANs that do not wish to use the superframe structure (referred to as a nonbeacon-enabled VAN) shall set both *macBeaconOrder* and *macSuperframeOrder* to 15. In this case, a coordinator shall not transmit beacons, except upon receipt of a beacon request command; all transmissions, with the exception of acknowledgment frames and any data frame that quickly follows the acknowledgment of a data request command (see 7.6.6.3), shall use an unslotted random access mechanism to access the channel. In addition, GTSs shall not be permitted.

An example of a superframe structure is shown in Figure 55. In this case, the beacon interval, *BI*, is twice as long as the active superframe duration, *SD*, and the CFP contains two GTSs.

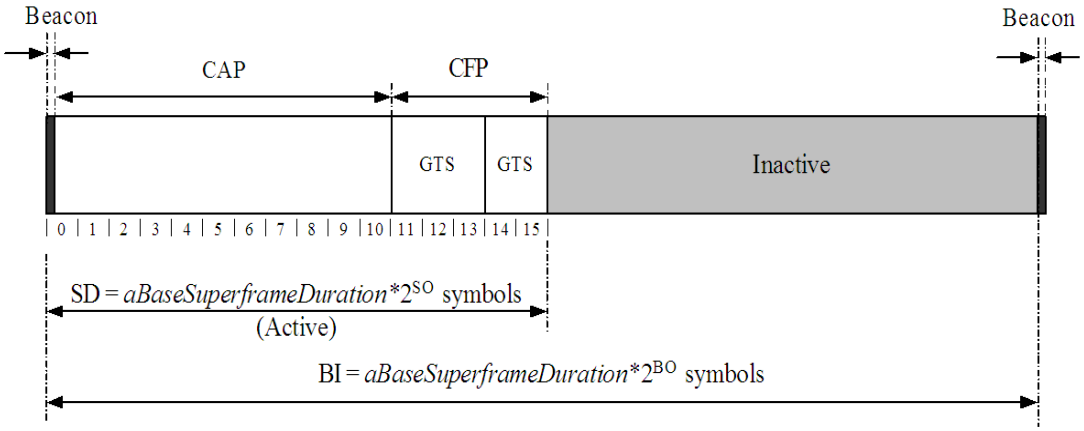


Figure 55—An example of the superframe structure

7.6.1.1.1 Contention access period (CAP)

The CAP shall start immediately following the beacon and complete before the beginning of the CFP on a superframe slot boundary. If the CFP is zero length, the CAP shall complete at the end of the active portion of the superframe. The CAP shall be at least *aMinCAPLength* symbols, unless additional space is needed to temporarily accommodate the increase in the beacon frame length needed to perform GTS maintenance (see 7.2.2.1.3), and shall shrink or grow dynamically to accommodate the size of the CFP.

All frames, except acknowledgment frames and any data frame that quickly follows the acknowledgment of a data request command (see 7.6.6.3), transmitted in the CAP shall use a slotted random access mechanism to access the channel. A device transmitting within the CAP shall ensure that its transaction is complete (i.e., including the reception of any acknowledgment) one IFS period (see 7.6.1.3) before the end of the CAP. If this is not possible, the device shall defer its transmission until the CAP of the following superframe.

MAC command frames shall always be transmitted in the CAP.

7.6.1.1.2 Contention-free period (CFP)

The CFP shall start on a slot boundary immediately following the CAP and it shall complete before the end of the active portion of the superframe. If any GTSs have been allocated by the VAN coordinator, they shall be located within the CFP and occupy contiguous slots. The CFP shall therefore grow or shrink depending on the total length of all of the combined GTSs.

No transmissions within the CFP shall use a random access mechanism to access the channel. A device transmitting in the CFP shall ensure that its transmissions are complete one IFS period (see 7.6.1.3) before the end of its GTS.

7.6.1.2 Incoming and outgoing superframe timing

On a beacon-enabled VAN, a coordinator that is not the VAN coordinator shall maintain the timing of both the superframe in which its coordinator transmits a beacon (the incoming superframe) and the superframe in which it transmits its own beacon (the outgoing superframe). The relative timing of these superframes is defined by the StartTime parameter of the MLME-START.request primitive (see 7.1.13.1 and 7.6.2.4). The relationship between incoming and outgoing superframes is illustrated in Figure 56.

The beacon order and superframe order shall be equal for all superframes on a VAN. All devices shall interact with the VAN only during the active portion of a superframe.

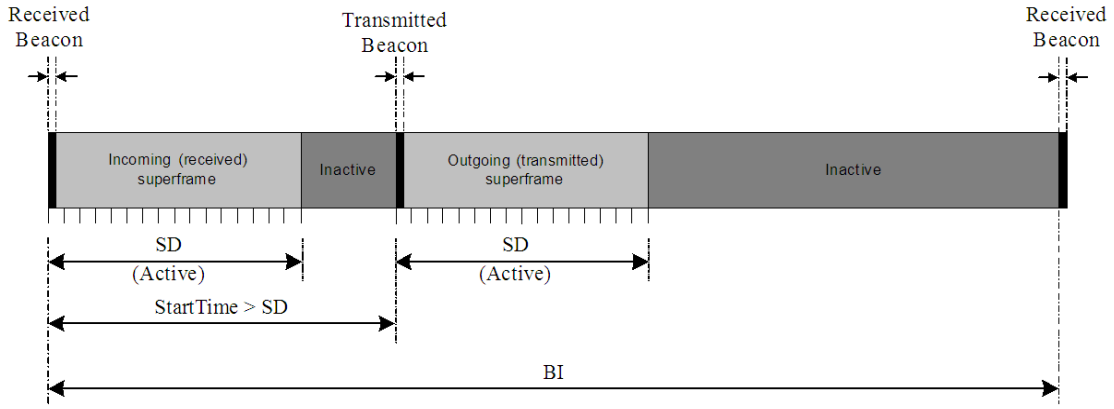


Figure 56—The relationship between incoming and outgoing beacons

7.6.1.3 Interframe spacing (IFS)

The MAC sublayer needs a finite amount of time to process data received by the PHY. To allow for this, two successive frames transmitted from a device shall be separated by at least an IFS period; if the first transmission requires an acknowledgment, the separation between the acknowledgment frame and the second transmission shall be at least an IFS period. The length of the IFS period is dependent on the size of the frame that has just been transmitted. Frames (i.e., MPDUs) of up to *aMaxSIFSFrameSize* octets in length shall be followed by a SIFS period of a duration of at least *macMinSIFSPeriod* symbols. Frames (i.e., MPDUs) with lengths greater than *aMaxSIFSFrameSize* octets shall be followed by a LIFS period of a duration of at least *macMinLIFSPeriod* symbols. These concepts are illustrated in Figure 57.

The random access algorithm shall take this requirement into account for transmissions in the CAP.

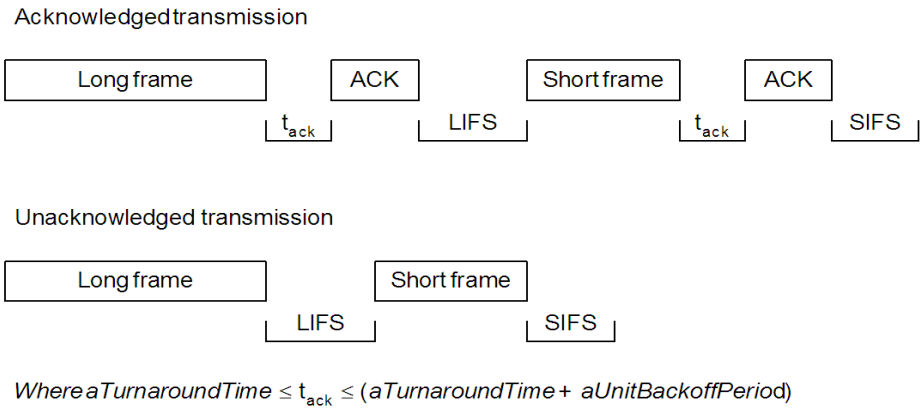


Figure 57—IFS

7.6.1.4 Random access algorithm

The random access algorithm shall be used before the transmission of data or MAC command frames transmitted within the CAP, unless the frame can be quickly transmitted following the acknowledgment of a

data request command (see 7.6.6.3 for timing requirements). The random access algorithm shall not be used for the transmission of beacon frames in a beacon-enabled VAN, acknowledgment frames, or data frames transmitted in the CFP.

If periodic beacons are being used in the VAN, the MAC sublayer shall employ the slotted version of the random access algorithm for transmissions in the CAP of the superframe. Conversely, if periodic beacons are not being used in the VAN or if a beacon could not be located in a beacon-enabled VAN, the MAC sublayer shall transmit using the unslotted version of the random access algorithm. In both cases, the algorithm is implemented using units of time called backoff periods, where one backoff period shall be equal to *aUnitBackoffPeriod* symbols.

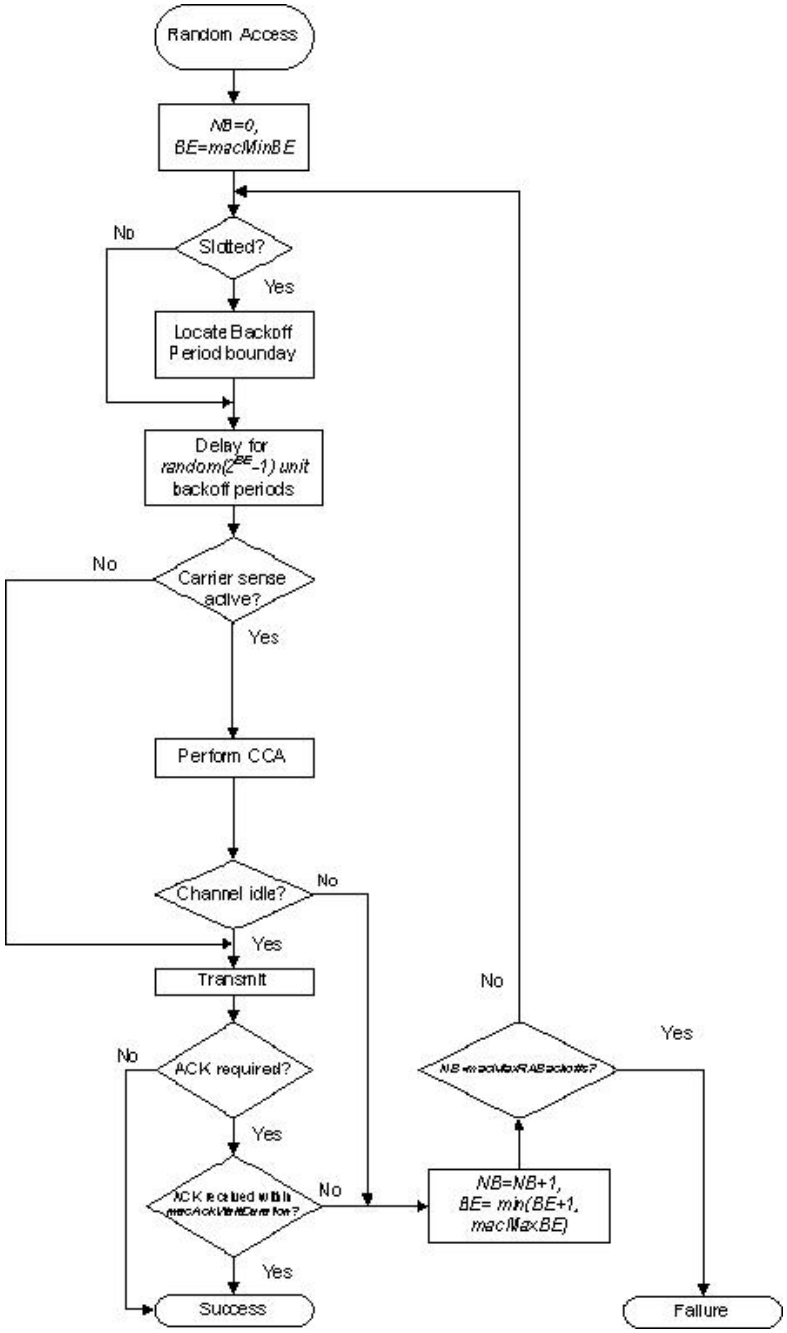
In slotted random access, the backoff period boundaries of every device in the VAN shall be aligned with the superframe slot boundaries of the VAN coordinator, i.e., the start of the first backoff period of each device is aligned with the start of the beacon transmission. In slotted random access, the MAC sublayer shall ensure that the PHY commences all of its transmissions on the boundary of a backoff period. In unslotted random access, the backoff periods of one device are not related in time to the backoff periods of any other device in the VAN.

Each device shall maintain two variables for each transmission attempt: *NB* and *BE*. *NB* is the number of times the random access algorithm was required to backoff while attempting the current transmission; this value shall be initialized to zero before each new transmission attempt. The variable *BE* is the backoff exponent, which is related to how many backoff periods a device shall wait before attempting to access/ assess a channel. *BE* shall be initialized to the value of *macMinBE*.

Figure 107 illustrates the steps of the random access algorithm. The MAC sublayer shall first initiaize NB and BE for slotted random acccess then locate the boundary of the next backoff period. The MAC sublayer shall delay for a random number of complete backoff periods in the range 0 to $2^{BE} - 1$ [step (2)] and then request that the PHY perform a transmission or optionally a CCA. In a slotted random access system, the transmission, or CCA if active, shall start on a backoff period boundary. In an unslotted system, the transmission, or CCA if active, shall start immediately.

In a slotted random access system, the MAC sublayer shall ensure that, after the random backoff, the remaining random access operations can be undertaken and the entire transaction can be transmitted before the end of the CAP. Note that any bit padding used by the supported PHY (see 6.9.2.2) must be considered in making this determination. If the number of backoff periods is greater than the remaining number of backoff periods in the CAP, the MAC sublayer shall pause the backoff countdown at the end of the CAP and resume it at the start of the CAP in the next superframe. If the number of backoff periods is less than or equal to the remaining number of backoff periods in the CAP, the MAC sublayer shall apply its backoff delay and then evaluate whether it can proceed. The MAC sublayer shall proceed if the remaining random access algorithm steps, the frame transmission, and any acknowledgment can be completed before the end of the CAP. If the MAC sublayer can proceed and CCA is active, it shall request that the PHY perform the CCA in the current superframe. If the MAC sublayer cannot proceed, it shall wait until the start of the CAP in the next superframe and apply a further random backoff delay before evaluating whether it can proceed again.

If CCA is active and the channel is assessed to be busy, the MAC sublayer shall increment both *NB* and *BE* by one, ensuring that *BE* shall be no more than *macMaxBE*. If the value of *NB* is less than or equal to *macMaxRABackoffs*, the random access algorithm shall return to ???. If the value of *NB* is greater than *macMaxRABackoffs*, the random access algorithm shall terminate with a channel access failure status.



7.6.2 Starting and maintaining VANs

This subclause specifies the procedures for scanning through channels, identifying VAN identifier conflicts, and starting VANs.

7.6.2.1 Scanning through channels

TBD

7.6.2.2 VAN identifier conflict resolution

In some instances a situation could occur in which two VANs exist in the same POS with the same VAN identifier. If this conflict happens, the VAN coordinator and its devices shall perform the VAN identifier conflict resolution procedure.

This procedure is optional for an RFD.

7.6.2.2.1 Detection

The PAN coordinator shall conclude that a PAN identifier conflict is present if either of the following apply:

— A beacon frame is received by the PAN coordinator with the PAN Coordinator subfield (see 7.2.2.1.2) set to one and the PAN identifier equal to *macPANId*.

— A PAN ID conflict notification command (see 7.4.5) is received by the PAN coordinator from a device associated with it on its PAN.

A device that is associated through the PAN coordinator (i.e., *macAssociatedPANCoord* is set to TRUE) shall conclude that a PAN identifier conflict is present if the following applies:

— A beacon frame is received by the device with the PAN Coordinator subfield set to one, the PAN identifier equal to *macPANId*, and an address that is equal to neither *macCoordShortAddress* nor *macCoordExtendedAddress*.

A device that is associated through a coordinator that is not the PAN coordinator shall not be capable of detecting a PAN identifier conflict.

7.6.2.2.2 Resolution

On the detection of a VAN identifier conflict by a device, it shall generate the VAN ID conflict notification command (see 7.4.5) and send it to its VAN coordinator. Since the VAN ID conflict notification command contains an acknowledgment request (see 7.4.3.1), the VAN coordinator shall confirm its receipt by sending an acknowledgment frame. Once the device has received the acknowledgment frame from the VAN coordinator, the MLME shall issue an MLME-SYNC-LOSS.indication primitive with the LossReason parameter set to VAN_ID_CONFLICT. If the device does not receive an acknowledgment frame, the MLME shall not inform the next higher layer of the VAN identifier conflict.

On the detection of a VAN identifier conflict by the VAN coordinator, the MLME shall issue an MLME-SYNC-LOSS.indication to the next higher layer with the LossReason parameter set to VAN_ID_CONFLICT. The next higher layer of the VAN coordinator may then perform an active scan and, using the information from the scan, select a new VAN identifier. The algorithm for selecting a suitable VAN identifier is out of the scope of this standard. If the next higher layer does select a new VAN identifier, it may then issue an MLME-START.request with the CoordRealignment parameter set to TRUE in order to realign the VAN, as described in 7.6.2.3.

7.6.2.3 Starting and realigning a VAN

This subclause specifies procedures for the VAN coordinator starting a VAN, coordinators realigning a VAN, and devices being realigned on a VAN.

7.6.2.3.1 Starting a VAN

A VAN should be started by an FFD only after having first performed a MAC sublayer reset, by issuing the MLME-RESET.request primitive with the SetDefaultPIB parameter set to TRUE, an active channel scan [Ed. Note - what is an active channel for VLC?], and a suitable VAN identifier selection. The algorithm for

selecting a suitable VAN identifier from the list of VAN descriptors returned from the active channel scan procedure is out of the scope of this standard. In addition, an FFD should set *macShortAddress* to a value less than 0xffff.

An FFD is instructed to begin operating a VAN through the use of the MLME-START.request primitive (see 7.1.13.1) with the VANCoordinator parameter set to TRUE and the CoordRealigment parameter set to FALSE. On receipt of this primitive, the MAC sublayer shall update the superframe configuration and channel parameters as specified in 7.6.2.3.4. After completing this, the MAC sublayer shall issue the MLME-START.confirm primitive with a status of SUCCESS and begin operating as the VAN coordinator.

7.6.2.3.2 Realigning a VAN

If a coordinator receives the MLME-START.request primitive (see 7.1.13.1) with the CoordRealignment parameter set to TRUE, the coordinator shall attempt to transmit a coordinator realignment command containing the new parameters for VANId, LogicalChannel.

When the coordinator is already transmitting beacons and the CoordRealignment parameter is set to TRUE, the next scheduled beacon shall be transmitted on the current channel using the current superframe configuration, with the Frame Pending subfield of the Frame Control field set to one. Immediately following the transmission of the beacon, the coordinator realignment command shall also be transmitted on the current channel using random access.

When the coordinator is not already transmitting beacons and the CoordRealignment parameter is set to TRUE, the coordinator realignment command shall be transmitted immediately on the current channel using random access.

If the transmission of the coordinator realignment command fails due to a channel access failure, the MLME shall notify the next higher layer by issuing the MLME-START.confirm primitive with a status of CHANNEL_ACCESS_FAILURE. The next higher layer may then choose to issue the MLME-START.request primitive again.

Upon successful transmission of the coordinator realignment command, the new superframe configuration and channel parameters shall be put into operation as described in 7.6.2.3.4 at the subsequent scheduled beacon, or immediately if the coordinator is not already transmitting beacons, and the MAC sublayer shall issue the MLME-START.confirm primitive with a status of SUCCESS.

7.6.2.3.3 Realignment in a VAN

If a device has received the coordinator realignment command (see 7.4.8) from the coordinator through which it is associated and the MLME was not carrying out an orphan scan, the MLME shall issue the MLME-SYNC-LOSS.indication primitive with the LossReason parameter set to REALIGNMENT and the VANId, LogicalChannel, ChannelPage, and the security-related parameters set to the respective fields in the coordinator realignment command. The next higher layer of a coordinator may then issue an MLME-START.request primitive with the CoordRealignment parameter set to TRUE. The next higher layer of a device that is not a coordinator may instead change the superframe configuration or channel parameters through use of the MLME-SET.request primitive.

7.6.2.3.4 Updating superframe configuration and channel PIB attributes

To update the superframe configuration and channel attributes, the MLME shall assign values from the MLME-START.request primitive parameters to the appropriate PIB attributes. The MLME shall set *macBeaconOrder* to the value of the BeaconOrder parameter. If *macBeaconOrder* is equal to 15, the MLME will also set *macSuperframeOrder* to 15. In this case, this primitive configures a nonbeacon-enabled VAN. If *macBeaconOrder* is less than 15, the MAC sublayer will set *macSuperframeOrder* to the value of the

SuperframeOrder parameter. The MAC sublayer shall also update *macVANID* with the value of the VANId parameter and update *phyCurrentChannel* with the values of the LogicalChannel parameters by issuing the PLME-SET.request primitive.

7.6.2.4 Beacon generation

A device shall be permitted to transmit beacon frames only if *macShortAddress* is not equal to 0xffff.

An FFD shall use the MLME-START.request primitive to begin transmitting beacons only if the BeaconOrder parameter is less than 15. The FFD may begin beacon transmission either as the VAN coordinator of a new VAN or as a device on a previously established VAN, depending upon the setting of the VANCoordinator parameter (see 7.1.13.1). The FFD shall begin beacon transmission on a previously established VAN only once it has successfully associated with that VAN.

If the FFD is the VAN coordinator (i.e., the VANCoordinator parameter is set to TRUE), the MAC sublayer shall ignore the StartTime parameter and begin beacon transmissions immediately. Setting the StartTime parameter to zero shall also cause the MAC sublayer to begin beacon transmissions immediately. If the FFD is not the VAN coordinator and the StartTime parameter is nonzero, the time to begin beacon transmissions shall be calculated using the following method. The StartTime parameter, which is rounded to a backoff slot boundary, shall be added to the time, obtained from the local clock, when the MAC sublayer receives the beacon of the coordinator through which it is associated. The MAC sublayer shall then begin beacon transmissions when the current time, obtained from the local clock, equals the number of calculated symbols. In order for the beacon transmission time to be calculated by the MAC sublayer, the MAC sublayer shall first track the beacon of the coordinator through which it is associated. If the MLME-START.request primitive is issued with a nonzero StartTime parameter and the MAC sublayer is not currently tracking the beacon of its coordinator, the MLME shall not begin beacon transmissions but shall instead issue the MLME-START.confirm primitive with a status of TRACKING_OFF.

If a device misses between one and (*aMaxLostBeacons*–1) consecutive beacon frames from its coordinator, the device shall continue to transmit its own beacons based on both *macBeaconOrder* (see 7.6.2.3.4) and its local clock. If the device then receives a beacon frame from its coordinator and, therefore, does not lose synchronization, the device shall resume transmitting its own beacons based on the StartTime parameter and the incoming beacon. If a device does lose synchronization with its coordinator, the MLME of the device shall issue the MLME-SYNC-LOSS.indication primitive to the next higher layer and immediately stop transmitting its own beacons. The next higher layer may, at any time following the reception of the MLME-SYNC-LOSS.indication primitive, resume beacon transmissions by issuing a new MLME-START.request primitive.

On receipt of the MLME-START.request primitive, the MAC sublayer shall set the VAN identifier in *macVANId* and use this value in the Source VAN Identifier field of the beacon frame. The address used in the Source Address field of the beacon frame shall contain the value of *aExtendedAddress* if *macShortAddress* is equal to 0xfffe or *macShortAddress* otherwise.

The time of transmission of the most recent beacon shall be recorded in *macBeaconTxTime* and shall be computed so that its value is taken at the same symbol boundary in each beacon frame, the location of which is implementation specific. The symbol boundary, which is specified by the *macSyncSymbolOffset* attribute, is the same as that used in the timestamp of the incoming beacon frame, as described in 7.6.4.1.

All beacon frames shall be transmitted at the beginning of each superframe at an interval equal to *aBase-SuperframeDuration* * 2^n symbols, where *n* is the value of *macBeaconOrder* (the construction of the beacon frame is specified in 7.2.2.1).

Beacon transmissions shall be given priority over all other transmit and receive operations.

7.6.2.5 Device discovery

The VAN coordinator or a coordinator indicates its presence on a VAN to other devices by transmitting beacon frames. This allows the other devices to perform device discovery.

A coordinator that is not the VAN coordinator shall begin transmitting beacon frames only when it has successfully associated with a VAN. The transmission of beacon frames by the device is initiated through the use of the MLME-START.request primitive with the VANCoordinator parameter set to FALSE. On receipt of this primitive, the MLME shall begin transmitting beacons based on the StartTime parameter (see 7.6.2.4) using the identifier of the VAN with which the device has associated, *macVANId*, and its extended address, *aExtendedAddress*, if *macShortAddress* is equal to 0xfffe, or its short address, *macShortAddress*, otherwise. A beacon frame shall be transmitted at a rate of one beacon frame every *aBaseSuperframeDuration* * 2^n symbols, where *n* is the value of *macBeaconOrder*.

7.6.3 Association and disassociation

This subclause specifies the procedures for association and disassociation.

7.6.3.1 Association

A device shall attempt to associate only after having first performed a MAC sublayer reset, by issuing the MLME-RESET.request primitive with the SetDefaultPIB parameter set to TRUE, and then having completed either an active channel scan (see 7.5.2.1.2) or a passive channel scan (see 7.5.2.1.3). The results of the channel scan would have then been used to choose a suitable VAN. The algorithm for selecting a suitable VAN with which to associate from the list of VAN descriptors returned from the channel scan procedure is out of the scope of this standard.

Following the selection of a VAN with which to associate, the next higher layers shall request through the MLME-ASSOCIATE.request primitive that the MLME configures the following PHY and MAC PIB attributes to the values necessary for association:

- *phyCurrentChannel* shall be set equal to the LogicalChannel parameter of the MLME-ASSOCIATE.request primitive.
- *macVANId* shall be set equal to the CoordPANId parameter of the MLME-ASSOCIATE.request primitive.
- *macCoordExtendedAddress* or *macCoordShortAddress*, depending on which is known from the beacon frame from the coordinator through which it wishes to associate, shall be set equal to the CoordAddress parameter of the MLME-ASSOCIATE.request primitive.

A coordinator shall allow association only if *macAssociationPermit* is set to TRUE. Similarly, a device should attempt to associate only with a VAN through a coordinator that is currently allowing association, as indicated in the results of the scanning procedure. If a coordinator with *macAssociationPermit* set to FALSE receives an association request command from a device, the command shall be ignored.

In order to optimize the association procedure on a beacon-enabled VAN, a device may begin tracking the beacon of the coordinator through which it wishes to associate a priori. This is achieved by the next higher layer issuing the MLME-SYNC.request primitive with the TrackBeacon parameter set to TRUE.

A device that is instructed to associate with a VAN, through the MLME-ASSOCIATE.request primitive, shall try to associate only with an existing VAN and shall not attempt to start its own PAN.

The MAC sublayer of an unassociated device shall initiate the association procedure by sending an association request command (see 7.4.1) to the coordinator of an existing VAN; if the association request command cannot be sent due to a channel access failure, the MAC sublayer shall notify the next higher

layer. Because the association request command contains an acknowledgment request (see 7.4.1.1), the coordinator shall confirm its receipt by sending an acknowledgment frame.

The acknowledgment to an association request command does not mean that the device has associated. The next higher layer of the coordinator needs time to determine whether the current resources available on the VAN are sufficient to allow another device to associate. The next higher layer should make this decision within *macResponseWaitTime* symbols. If the next higher layer of the coordinator finds that the device was previously associated on its VAN, all previously obtained device-specific information should be removed. If sufficient resources are available, the next higher layer should allocate a 16-bit short address to the device, and the MAC sublayer shall generate an association response command (see 7.4.2) containing the new address and a status indicating a successful association. If sufficient resources are not available, the next higher layer of the coordinator should inform the MAC sublayer, and the MLME shall generate an association response command containing a status indicating a failure (see Table 64). The association response command shall be sent to the device requesting association using indirect transmission, i.e., the association response command frame shall be added to the list of pending transactions stored on the coordinator and extracted at the discretion of the device concerned using the method described in 7.6.6.3.

If the Allocate Address subfield of the Capability Information field (see 7.4.1.2) of the association request command is set to one, the next higher layer of the coordinator shall allocate a 16-bit address with a range depending on the addressing mode supported by the coordinator, as described in Table 68. If the Allocate Address subfield of the association request command is set to zero, the 16-bit short address shall be equal to 0xfffe. A short address of 0xfffe is a special case that indicates that the device has associated, but has not been allocated a short address by the coordinator. In this case, the device shall use only its 64-bit extended address to operate on the network.

On receipt of the acknowledgment to the association request command, the device shall wait for at most *macResponseWaitTime* symbols for the coordinator to make its association decision; the PIB attribute *macResponseWaitTime* is a network-topology-dependent parameter and may be set to match the specific requirements of the network that a device is trying to join. If the device is tracking the beacon, it shall attempt to extract the association response command from the coordinator whenever it is indicated in the beacon frame. If the device is not tracking the beacon, it shall attempt to extract the association response command from the coordinator after *macResponseWaitTime* symbols. If the device does not extract an association response command frame from the coordinator within *macResponseWaitTime* symbols, the MLME shall issue the MLME-ASSOCIATE.confirm primitive with a status of NO_DATA, and the association attempt shall be deemed a failure. In this case, the next higher layer shall terminate any tracking of the beacon. This is achieved by issuing the MLME-SYNC.request primitive with the TrackBeacon parameter set to FALSE.

Because the association response command contains an acknowledgment request (see 7.4.2.1), the device requesting association shall confirm its receipt by sending an acknowledgment frame. If the Association Status field of the command indicates that the association was successful, the device shall store the address contained in the 16-bit Short Address field of the command in *macShortAddress*; communication on the VAN using this short address shall depend on its range, as described in Table 68. If the original beacon selected for association following a scan contained the short address of the coordinator, the extended address of the coordinator, contained in the MHR of the association response command frame, shall be stored in *macCoordExtendedAddress.*

If the Association Status field of the command indicates that the association was unsuccessful, the device shall set *macVANId* to the default value (0xffff).

7.6.3.2 Disassociation

The disassociation procedure is initiated by the next higher layer by issuing the MLME-DISASSOCIATE.request primitive to the MLME.

Table 68—Usage of the 16-bit short address

Value of *macShortAddress*	Description
0x0000–0xfffd	If a source address is included, the device shall use short source addressing mode for beacon and data frames and the appropriate source addressing mode specified in 7.4 for MAC command frames.
0xfffe	If a source address is included, the device shall use extended source addressing mode for beacon and data frames and the appropriate source addressing mode specified in 7.4 for MAC command frames.
0xffff	The device is not associated and, therefore, shall not perform any data frame communication. The device shall use the appropriate source addressing mode specified in 7.4 for MAC command frames.

When a coordinator wants one of its associated devices to leave the VAN, the MLME of the coordinator shall send the disassociation notification command in the manner specified by the TxIndirect parameter of the MLME-DISASSOCIATE.request primitive previously sent by the next higher layer. If TxIndirect is TRUE, the MLME of the coordinator shall send the disassociation notification command to the device using indirect transmission, i.e., the disassociation notification command frame shall be added to the list of pending transactions stored on the coordinator and extracted at the discretion of the device concerned using the method described in 7.6.6.3. If the command frame is not successfully extracted by the device, the coordinator should consider the device disassociated. Otherwise, the MLME shall send the disassociation notification command to the device directly. In this case, if the disassociation notification command cannot be sent due to a channel access failure, the MAC sublayer shall notify the next higher layer.

Because the disassociation command contains an acknowledgment request (see 7.4.3.1), the receiving device shall confirm its receipt by sending an acknowledgment frame. If the direct or indirect transmission fails, the coordinator should consider the device disassociated.

If an associated device wants to leave the VAN, the MLME of the device shall send a disassociation notification command to its coordinator. If the disassociation notification command cannot be sent due to a channel access failure, the MAC sublayer shall notify the next higher layer. Because the disassociation command contains an acknowledgment request (see 7.4.3.1), the coordinator shall confirm its receipt by sending an acknowledgment frame. However, even if the acknowledgment is not received, the device should consider itself disassociated.

If the source address contained in the disassociation notification command is equal to *macCoord-ExtendedAddress*, the device should consider itself disassociated. If the command is received by a coordinator and the source is not equal to *macCoordExtendedAddress*, it shall verify that the source address corresponds to one of its associated devices; if so, the coordinator should consider the device disassociated. If none of the above conditions is satisfied, the command shall be ignored.

An associated device shall disassociate itself by removing all references to the VAN; the MLME shall set *macVANId*, *macShortAddress*, *macAssociatedVANCoord*, *macCoordShortAddress* and *macCoordExtended-Address* to the default values. The next higher layer of a coordinator should disassociate a device by removing all references to that device.

The next higher layer of the requesting device shall be notified of the result of the disassociation procedure through the MLME-DISASSOCIATE.confirm primitive.

7.6.4 Synchronization

This subclause specifies the procedures for coordinators to generate beacon frames and for devices to synchronize with a coordinator. For VANs supporting beacons, synchronization is performed by receiving

and decoding the beacon frames. For VANs not supporting beacons, synchronization is performed by polling the coordinator for data.

7.6.4.1 Synchronization with beacons

All devices operating on a beacon-enabled VAN (i.e., *macBeaconOrder* < 15) shall be able to acquire beacon sychronization in order to detect any pending messages or to track the beacon. Devices shall be permitted to acquire beacon synchronization only with beacons containing the VAN identifier specified in *macVANId*. If *macVANId* specifies the broadcast VAN identifier (0xffff), a device shall not attempt to acquire beacon synchronization.

A device is instructed to attempt to acquire the beacon through the MLME-SYNC.request primitive. If tracking is specified in the MLME-SYNC.request primitive, the device shall attempt to acquire the beacon and keep track of it by regular and timely activation of its receiver. If tracking is not specified, the device shall either attempt to acquire the beacon only once or terminate the tracking after the next beacon if tracking was enabled through a previous request.

To acquire beacon synchronization, a device shall enable its receiver and search for at most [*aBaseSuperframeDuration* * (2^n + 1)] symbols, where *n* is the value of *macBeaconOrder*. If a beacon frame containing the current VAN identifier of the device is not received, the MLME shall repeat this search. Once the number of missed beacons reaches *aMaxLostBeacons*, the MLME shall notify the next higher layer by issuing the MLME-SYNC-LOSS.indication primitive with a loss reason of BEACON_LOSS.

The MLME shall timestamp each received beacon frame at the same symbol boundary within each frame, the location of which is described by the *macSyncSymbolOffset* attribute. The symbol boundary shall be the same as that used in the timestamp of the outgoing beacon frame, stored in *macBeaconTxTime*. The timestamp value shall be that of the local clock of the device at the time of the symbol boundary. The timestamp is intended to be a relative time measurement that may or may not be made absolute, at the discretion of the implementer.

If a protected beacon frame is received (i.e., the Security Enabled subfield in the Frame Control field is set to one), the device shall attempt to unsecure the beacon frame using the unsecuring process described in 7.6.8.2.3.

If the status from the unsecuring process is not SUCCESS, the MLME shall issue an MLME-COMM-STATUS.indication primitive with the status parameter set to the status from the unsecuring process, indicating the error.

The security-related elements of the VAN descriptor corresponding to the beacon (see Table 39) shall be set to the corresponding parameters returned by the unsecuring process. The SecurityFailure element of the VAN descriptor shall be set to SUCCESS if the status from the unsecuring process is SUCCESS and set to one of the other status codes indicating an error in the security processing otherwise.

If a beacon frame is received, the MLME shall discard the beacon frame if the Source Address and the Source VAN Identifier fields of the MHR of the beacon frame do not match the coordinator source address (*macCoordShortAddress* or *macCoordExtendedAddress*, depending on the addressing mode) and the VAN identifier of the device (*macVANId*).

If a valid beacon frame is received and *macAutoRequest* is set to FALSE, the MLME shall indicate the beacon parameters to the next higher layer by issuing the MLME-BEACON-NOTIFY.indication primitive. If a beacon frame is received and *macAutoRequest* is set to TRUE, the MLME shall first issue the MLME-BEACON-NOTIFY.indication primitive if the beacon contains any payload. The MLME shall then compare its address with those addresses in the Address List field of the beacon frame. If the Address List field contains the 16-bit short or 64-bit extended address of the device and the source VAN identifier matches

macVANId, the MLME shall follow the procedure for extracting pending data from the coordinator (see 7.6.6.3).

If beacon tracking is activated, the MLME shall enable its receiver at a time prior to the next expected beacon frame transmission, i.e., just before the known start of the next superframe. If the number of consecutive beacons missed by the MLME reaches *aMaxLostBeacons*, the MLME shall respond with the MLME-SYNC-LOSS.indication primitive with a loss reason of BEACON_LOST.

7.6.4.2 Synchronization without beacons

All devices operating on a nonbeacon-enabled VAN (*macBeaconOrder* = 15) shall be able to poll the coordinator for data at the discretion of the next higher layer.

A device is instructed to poll the coordinator when the MLME receives the MLME-POLL.request primitive. On receipt of this primitive, the MLME shall follow the procedure for extracting pending data from the coordinator (see 7.6.6.3).

7.6.4.3 Orphaned device realignment

If the next higher layer receives repeated communications failures following its requests to transmit data, it may conclude that it has been orphaned. A single communications failure occurs when a device transaction fails to reach the coordinator, i.e., an acknowledgment is not received after *macMaxFrameRetries* attempts at sending the data. If the next higher layer concludes that it has been orphaned, it may instruct the MLME to either perform the orphaned device realignment procedure, or to reset the MAC sublayer and then perform the association procedure.

If the decision has been made by the next higher layer to perform the orphaned device realignment procedure, it will have issued an MLME-SCAN.request with the ScanType parameter set to orphan scan and the ScanChannel parameter containing the list of channels to be scanned. Upon receiving this primitive, the MAC sublayer shall begin an orphan scan, as described in 7.5.2.1.4.

If the orphan scan is successful (i.e., its VAN has been located), the device shall update its MAC PIB with the VAN information contained in the coordinator realignment command (see 7.4.8).

7.6.5 Transaction handling

Transactions can be instigated from the devices themselves rather than from the coordinator. In other words, either the coordinator needs to indicate in its beacon when messages are pending for devices or the devices themselves need to poll the coordinator to determine whether they have any messages pending. Such transfers are called indirect transmissions.

The coordinator shall begin handling a transaction on receipt of an indirect transmission request either via the MCPS-DATA.request primitive or via a request from the MLME to send a MAC command instigated by a primitive from the next higher layer, such as the MLME-ASSOCIATE.response primitive (see 7.1.3.3). On completion of the transaction, the MAC sublayer shall indicate a status value to the next higher layer. If a request primitive instigated the indirect transmission, the corresponding confirm primitive shall be used to convey the appropriate status value. Conversely, if a response primitive instigated the indirect transmission, the MLME-COMM-STATUS.indication primitive shall be used to convey the appropriate status value. The MLME-COMM-STATUS.indication primitive can be related to its corresponding response primitive by examining the Destination Address field.

The information contained in the indirect transmission request forms a transaction, and the coordinator shall be capable of storing at least one transaction. On receipt of an indirect transmission request, if there is no

capacity to store another transaction, the MAC sublayer shall indicate to the next higher layer a status of TRANSACTION_OVERFLOW in the appropriate corresponding primitive.

If the coordinator is capable of storing more than one transaction, it shall ensure that all the transactions for the same device are sent in the order in which they arrived at the MAC sublayer. For each transaction sent, if another exists for the same device, the MAC sublayer shall set its Frame Pending subfield to one, indicating the additional pending data.

Each transaction shall persist in the coordinator for at most *macTransactionPersistenceTime*. If the transaction is not successfully extracted by the appropriate device within this time, the transaction information shall be discarded and the MAC sublayer shall indicate to the next higher layer a status of TRANSACTION_EXPIRED in the appropriate corresponding primitive. In order to be successfully extracted, an acknowledgment shall be received if one was requested.

If the transaction was successful, the transaction information shall be discarded, and the MAC sublayer shall indicate to the next higher layer a status of SUCCESS in the appropriate corresponding primitive.

If the coordinator transmits beacons, it shall list the addresses of the devices to which each transaction is associated in the Address List field and indicate the number of addresses in the Pending Address Specification field of the beacon frame. If the coordinator is able to store more than seven pending transactions, it shall indicate them in its beacon on a first-come-first-served basis, ensuring that the beacon frame contains at most seven addresses. For transactions requiring a GTS, the VAN coordinator shall not add the address of the recipient to its list of pending addresses in the beacon frame. Instead it shall transmit the transaction in the GTS allocated for the device (see 7.6.7.3).

On a beacon-enabled VAN, if there is a transaction pending for the broadcast address, the Frame Pending subfield of the Frame Control field in the beacon frame shall be set to one, and the pending message shall be transmitted immediately following the beacon using the random access algorithm. If there is a second message pending for the broadcast address, its transmission shall be delayed until the following superframe. Only one broadcast message shall be allowed to be sent indirectly per superframe.

On a beacon-enabled VAN, a device that receives a beacon containing its address in the list of pending addresses shall attempt to extract the data from the coordinator. On a nonbeacon-enabled VAN, a device shall attempt to extract the data from the coordinator on receipt of the MLME-POLL.request primitive. The procedure for extracting pending data from the coordinator is described in 7.6.6.3. If a device receives a beacon with the Frame Pending subfield set to one, it shall leave its receiver enabled for up to *macMaxFrameTotalWaitTime* symbols to receive the broadcast data frame from the coordinator.

7.6.6 Transmission, reception, and acknowledgment

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

7.6.6.1 Transmission

Each device shall store its current DSN value in the MAC PIB attribute *macDSN* and initialize it to a random value; the algorithm for choosing a random number is out of the scope of this standard. Each time a data or a MAC command frame is generated, the MAC sublayer shall copy the value of *macDSN* into the Sequence Number field of the MHR of the outgoing frame and then increment it by one. Each device shall generate exactly one DSN regardless of the number of unique devices with which it wishes to communicate. The value of *macDSN* shall be permitted to roll over.

Each coordinator shall store its current BSN value in the MAC PIB attribute *macBSN* and initialize it to a random value; the algorithm for choosing a random number is out of the scope of this standard. Each time a beacon frame is generated, the MAC sublayer shall copy the value of *macBSN* into the Sequence Number

field of the MHR of the outgoing frame and then increment it by one. The value of *macBSN* shall be permitted to roll over.

It should be noted that both the DSN and BSN are 8-bit values and, therefore, have limited use to the next higher layer (e.g., in the case of the DSN, in detecting retransmitted frames).

The Source Address field, if present, shall contain the address of the device sending the frame. When a device has associated and has been allocated a 16-bit short address (i.e., *macShortAddress* is not equal to 0xfffe or 0xffff), it shall use that address in preference to its 64-bit extended address (i.e., *aExtendedAddress*) wherever possible. When a device has not yet associated to a VAN or *macShortAddress* is equal to 0xffff, it shall use its 64-bit extended address in all communications requiring the Source Address field. If the Source Address field is not present, the originator of the frame shall be assumed to be the VAN coordinator, and the Destination Address field shall contain the address of the recipient.

The Destination Address field, if present, shall contain the address of the intended recipient of the frame, which may be either a 16-bit short address or a 64-bit extended address. If the Destination Address field is not present, the recipient of the frame shall be assumed to be the VAN coordinator, and the Source Address field shall contain the address of the originator.

If both destination and source addressing information is present, the MAC sublayer shall compare the destination and source VAN identifiers. If the VAN identifiers are identical, the VAN ID Compression subfield of the Frame Control field shall be set to one, and the source VAN identifier shall be omitted from the transmitted frame. If the VAN identifiers are different, the VAN ID Compression subfield of the Frame Control field shall be set to zero, and both Destination VAN Identifier and Source VAN Identifier fields shall be included in the transmitted frame. If only either the destination or the source addressing information is present, the VAN ID Compression subfield of the Frame Control field shall be set to zero, and the VAN identifier field of the single address shall be included in the transmitted frame.

If the frame is to be transmitted on a beacon-enabled PAN, the transmitting device shall attempt to find the beacon before transmitting. If the beacon is not being tracked (see 7.6.4.1) and hence the device does not know where the beacon will appear, it shall enable its receiver and search for at most [*aBaseSuperframeDuration* * (2^n + 1)] symbols, where *n* is the value of *macBeaconOrder,* in order to find the beacon. If the beacon is not found after this time, the device shall transmit the frame following the successful application of the unslotted version of the random access algorithm (see 7.6.1.4). Once the beacon has been found, either after a search or due to its being tracked, the frame shall be transmitted in the appropriate portion of the superframe. Transmissions in the CAP shall follow a successful application of the slotted version of the random access algorithm (see 7.6.1.4), and transmissions in a GTS shall not use random access.

If the frame is to be transmitted on a nonbeacon-enabled VAN, the frame shall be transmitted following the successful application of the unslotted version of the random access algorithm (see 7.6.1.4).

For either a beacon-enabled VAN or a nonbeacon-enabled VAN, if the transmission is direct and originates due to a primitive issued by the next higher layer and the random access algorithm fails, the next higher layer shall be notified. If the transmission is indirect and the random access algorithm fails, the frame shall remain in the transaction queue until it is requested again and successfully transmitted or until the transaction expires.

The device shall process the frame using the outgoing frame security procedure described in 7.6.8.2.1.

If the status from the outgoing frame security procedure is not SUCCESS, the MLME shall issue the corresponding confirm or MLME-COMM-STATUS.indication primitive with the status parameter set to the status from the outgoing frame security procedure, indicating the error.

To transmit the frame, the MAC sublayer shall first enable the transmitter by issuing the PLME-SET-TRX-STATE.request primitive with a state of TX_ON to the PHY. On receipt of the PLME-SET-TRX-STATE.confirm primitive with a status of either SUCCESS or TX_ON, the constructed frame shall then be transmitted by issuing the PD-DATA.request primitive. Finally, on receipt of the PD-DATA.confirm primitive, the MAC sublayer shall disable the transmitter by issuing the PLME-SET-TRX-STATE.request primitive with a state of RX_ON or TRX_OFF to the PHY, depending on whether the receiver is to be enabled following the transmission. In the case where the Acknowledgment Request subfield of the Frame Control field is set to one, the MAC sublayer shall enable the receiver immediately following the transmission of the frame by issuing the PLME-SET-TRX-STATE.request primitive with a state of RX_ON to the PHY.

7.6.6.2 Reception and rejection

Each device may choose whether the MAC sublayer is to enable its receiver during idle periods. During these idle periods, the MAC sublayer shall still service transceiver task requests from the next higher layer. A transceiver task shall be defined as a transmission request with acknowledgment reception, if required, or a reception request. On completion of each transceiver task, the MAC sublayer shall request that the PHY enables or disables its receiver, depending on the values of *macBeaconOrder* and *macRxOnWhenIdle.* If *macBeaconOrder* is less than 15, the value of *macRxOnWhenIdle* shall be considered relevant only during idle periods of the CAP of the incoming superframe. If *macBeaconOrder* is equal to 15, the value of *macRxOnWhenIdle* shall be considered relevant at all times.

Due to the nature of radio communications, a device with its receiver enabled will be able to receive and decode transmissions from all devices complying with this standard that are currently operating on the same channel and are in its POS, along with interference from other sources. The MAC sublayer shall, therefore, be able to filter incoming frames and present only the frames that are of interest to the upper layers.

For the first level of filtering, the MAC sublayer shall discard all received frames that do not contain a correct value in their FCS field in the MFR (see 7.2.1.9). The FCS field shall be verified on reception by recalculating the purported FCS over the MHR and MAC payload of the received frame and by subsequently comparing this value with the received FCS field. The FCS field of the received frame shall be considered to be correct if these values are the same and incorrect otherwise.

The second level of filtering shall be dependent on whether the MAC sublayer is currently operating in promiscuous mode. In promiscuous mode, the MAC sublayer shall pass all frames received after the first filter directly to the upper layers without applying any more filtering or processing. The MAC sublayer shall be in promiscuous mode if *macPromiscuousMode* is set to TRUE.

If the MAC sublayer is not in promiscuous mode (i.e., *macPromiscuousMode* is set to FALSE), it shall accept only frames that satisfy all of the following third-level filtering requirements:

— The Frame Type subfield shall not contain a reserved frame type.

— The Frame Version subfield shall not contain a reserved value.

— If a destination VAN identifier is included in the frame, it shall match *macVANId* or shall be the broadcast VAN identifier (0xffff).

— If a short destination address is included in the frame, it shall match either *macShortAddress* or the broadcast address (0xffff). Otherwise, if an extended destination address is included in the frame, it shall match *aExtendedAddress*.

— If the frame type indicates that the frame is a beacon frame, the source VAN identifier shall match *macVANId* unless *macVANId* is equal to 0xffff, in which case the beacon frame shall be accepted regardless of the source VAN identifier.

— If only source addressing fields are included in a data or MAC command frame, the frame shall be accepted only if the device is the VAN coordinator and the source VAN identifier matches *macVANId*.

If any of the third-level filtering requirements are not satisfied, the MAC sublayer shall discard the incoming frame without processing it further. If all of the third-level filtering requirements are satisfied, the frame shall be considered valid and processed further. For valid frames that are not broadcast, if the Frame Type subfield indicates a data or MAC command frame and the Acknowledgment Request subfield of the Frame Control field is set to one, the MAC sublayer shall send an acknowledgment frame. Prior to the transmission of the acknowledgment frame, the sequence number included in the received data or MAC command frame shall be copied into the Sequence Number field of the acknowledgment frame. This step will allow the transaction originator to know that it has received the appropriate acknowledgment frame.

If the VAN ID Compression subfield of the Frame Control field is set to one and both destination and source addressing information is included in the frame, the MAC sublayer shall assume that the omitted Source VAN Identifier field is identical to the Destination VAN Identifier field.

The device shall process the frame using the incoming frame security procedure described in 7.6.8.2.3.

If the status from the incoming frame security procedure is not SUCCESS, the MLME shall issue the corresponding confirm or MLME-COMM-STATUS.indication primitive with the status parameter set to the status from the incoming frame security procedure, indicating the error, and with the security-related parameters set to the corresponding parameters returned by the unsecuring process.

If the valid frame is a data frame, the MAC sublayer shall pass the frame to the next higher layer. This is achieved by issuing the MCPS-DATA.indication primitive containing the frame information. The security-related parameters of the MCPS-DATA.indication primitive shall be set to the corresponding parameters returned by the unsecuring process.

If the valid frame is a MAC command or beacon frame, it shall be processed by the MAC sublayer accordingly, and a corresponding confirm or indication primitive may be sent to the next higher layer. The security-related parameters of the corresponding confirm or indication primitive shall be set to the corresponding parameters returned by the unsecuring process.

7.6.6.3 Extracting pending data from a coordinator

A device on a beacon-enabled VAN can determine whether any frames are pending for it by examining the contents of the received beacon frame, as described in 7.6.4.1. If the address of the device is contained in the Address List field of the beacon frame and *macAutoRequest* is TRUE, the MLME of the device shall send a data request command (see 7.4.4) to the coordinator during the CAP with the Acknowledgment Request subfield of the Frame Control field set to one; the only exception to this is if the beacon frame is received while performing an active or passive scan (see 7.6.2.1). There are two other cases for which the MLME shall send a data request command to the coordinator. The first case is when the MLME receives the MLME-POLL.request primitive. In the second case, a device may send a data request command *macResponseWaitTime* symbols after the acknowledgment to a request command frame, such as during the association procedure. If the data request is intended for the VAN coordinator, the destination address information may be omitted.

If the data request command originated from an MLME-POLL.request primitive, the MLME shall perform the security process on the data request command based on the SecurityLevel, KeyIdMode, KeySource, and KeyIndex parameters of the MLME-POLL.request primitive, according to 7.6.8.2.1. Otherwise, the MLME shall perform the security process on the data request command based on the *macAutoRequestSecurityLevel*, *macAutoRequestKeyIdMode*, *macAutoRequestKeySource,* and *macAutoRequestKeyIndex* PIB attributes, according to 7.6.8.2.1.

On successfully receiving a data request command, the coordinator shall send an acknowledgment frame, thus confirming its receipt. If the coordinator has enough time to determine whether the device has a frame pending before sending the acknowledgment frame (see 7.6.6.4.2), it shall set the Frame Pending subfield of the Frame Control field of the acknowledgment frame accordingly to indicate whether a frame is actually pending for the device. If this is not possible, the coordinator shall set the Frame Pending subfield of the acknowledgment frame to one.

On receipt of the acknowledgment frame with the Frame Pending subfield set to zero, the device shall conclude that there are no data pending at the coordinator.

On receipt of the acknowledgment frame with the Frame Pending subfield set to one, a device shall enable its receiver for at most *macMaxFrameTotalWaitTime* CAP symbols in a beacon-enabled VAN, or symbols in a nonbeacon-enabled VAN, to receive the corresponding data frame from the coordinator. If there is an actual data frame pending within the coordinator for the requesting device, the coordinator shall send the frame to the device using one of the mechanisms described in this subclause. If there is no data frame pending for the requesting device, the coordinator shall send a data frame without requesting acknowledgment to the device containing a zero length payload, indicating that no data are present, using one of the mechanisms described in this subclause.

The data frame following the acknowledgment of the data request command shall be transmitted using one of the following mechanisms:

— Without using random access, if the MAC sublayer can commence transmission of the data frame between *aTurnaroundTime* and (*aTurnaroundTime* + *aUnitBackoffPeriod*) symbols, on a backoff slot boundary, and there is time remaining in the CAP for the message, appropriate IFS, and acknowledgment (see 6.4.1 for an explanation of *aTurnAroundTime*). If a requested acknowledgment frame is not received following this data frame, the process shall begin anew following the receipt of a new data request command.

— Using random access, otherwise.

If the requesting device does not receive a data frame from the coordinator within *macMaxFrameTotalWaitTime* CAP symbols in a beacon-enabled VAN, or symbols in a nonbeacon-enabled VAN, or if the requesting device receives a data frame from the coordinator with a zero length payload, it shall conclude that there are no data pending at the coordinator. If the requesting device does receive a data frame from the coordinator, it shall send an acknowledgment frame, if requested, thus confirming receipt.

If the Frame Pending subfield of the Frame Control field of the data frame received from the coordinator is set to one, the device still has more data pending with the coordinator. In this case it may extract the data by sending a new data request command to the coordinator.

7.6.6.4 Use of acknowledgments and retransmissions

A data or MAC command frame shall be sent with the Acknowledgment Request subfield of its Frame Control field set appropriately for the frame. A beacon or acknowledgment frame shall always be sent with the Acknowledgment Request subfield set to zero. Similarly, any frame that is broadcast shall be sent with its Acknowledgment Request subfield set to zero.

7.6.6.4.1 No acknowledgment

A frame transmitted with its Acknowledgment Request subfield set to zero shall not be acknowledged by its intended recipient. The originating device shall assume that the transmission of the frame was successful.

The message sequence chart in Figure 58 shows the scenario for transmitting a single frame of data from an originator to a recipient without requiring an acknowledgment. In this case, the originator transmits the data frame with the Acknowledgment Request (AR) subfield of the Frame Control field equal to zero.

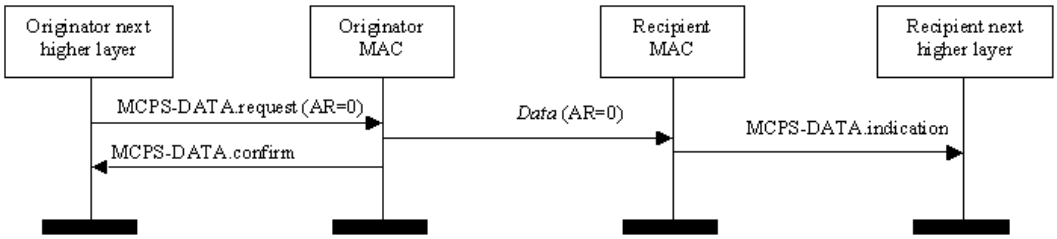


Figure 58—Successful data transmission without an acknowledgment

7.6.6.4.2 Acknowledgment

A frame transmitted with the Acknowledgment Request subfield of its Frame Control field set to one shall be acknowledged by the recipient. If the intended recipient correctly receives the frame, it shall generate and send an acknowledgment frame containing the same DSN from the data or MAC command frame that is being acknowledged.

The transmission of an acknowledgment frame in a nonbeacon-enabled VAN or in the CFP shall commence *aTurnaroundTime* symbols after the reception of the last symbol of the data or MAC command frame. The transmission of an acknowledgment frame in the CAP shall commence either *aTurnaroundTime* symbols after the reception of the last symbol of the data or MAC command frame or at a backoff slot boundary. In the latter case, the transmission of an acknowledgment frame shall commence between *aTurnaroundTime* and (*aTurnaroundTime* + *aUnitBackoffPeriod*) symbols after the reception of the last symbol of the data or MAC command frame. The constant *aTurnaroundTime* is defined in Table 30 (see 6.4.1).

The message sequence chart in Figure 59 shows the scenario for transmitting a single frame of data from an originator to a recipient with an acknowledgment. In this case, the originator indicates to the recipient that it requires an acknowledgment by transmitting the data frame with the Acknowledgment Request (AR) subfield of the Frame Control field set to one.

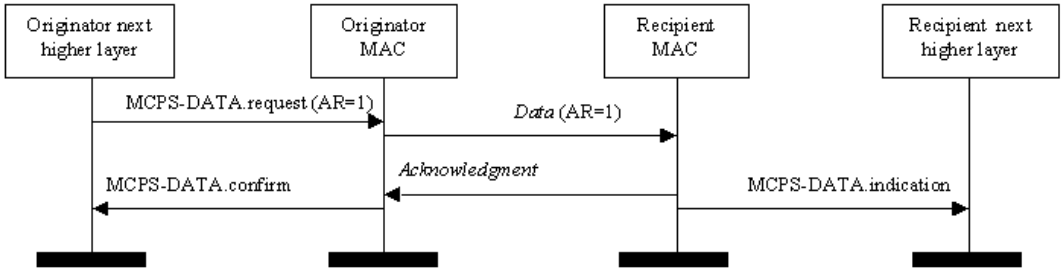


Figure 59—Successful data transmission with an acknowledgment

7.6.6.4.3 Retransmissions

A device that sends a frame with the Acknowledgment Request subfield of its Frame Control field set to zero shall assume that the transmission was successfully received and shall hence not perform the retransmission procedure.

A device that sends a data or MAC command frame with its Acknowledgment Request subfield set to one shall wait for at most *macAckWaitDuration* symbols for the corresponding acknowledgment frame to be received. If an acknowledgment frame is received within *macAckWaitDuration* symbols and contains the same DSN as the original transmission, the transmission is considered successful, and no further action regarding retransmission shall be taken by the device. If an acknowledgment is not received within *macAckWaitDuration* symbols or an acknowledgment is received containing a DSN that was not the same as the original transmission, the device shall conclude that the single transmission attempt has failed.

If a single transmission attempt has failed and the transmission was indirect, the coordinator shall not retransmit the data or MAC command frame. Instead, the frame shall remain in the transaction queue of the coordinator and can only be extracted following the reception of a new data request command. If a new data request command is received, the originating device shall transmit the frame using the same DSN as was used in the original transmission.

If a single transmission attempt has failed and the transmission was direct, the device shall repeat the process of transmitting the data or MAC command frame and waiting for the acknowledgment, up to a maximum of *macMaxFrameRetries* times. The retransmitted frame shall contain the same DSN as was used in the original transmission. Each retransmission shall only be attempted if it can be completed within the same portion of the superframe, i.e., the CAP or a GTS in which the original transmission was attempted. If this timing is not possible, the retransmission shall be deferred until the same portion in the next superframe. If an acknowledgment is still not received after *macMaxFrameRetries* retransmissions, the MAC sublayer shall assume the transmission has failed and notify the next higher layer of the failure.

7.6.6.5 Promiscuous mode

A device may activate promiscuous mode by setting *macPromiscuousMode*. If the MLME is requested to set *macPromiscuousMode* to TRUE, the MLME shall then request that the PHY enable its receiver. This request is achieved when the MLME issues the PLME-SET-TRX-STATE.request primitive with a state of RX_ON.

When in promiscuous mode, the MAC sublayer shall process received frames according to 7.6.6.2 and pass all frames correctly received to the next higher layer using the MCPS-DATA.indication primitive. The source and destination addressing mode parameters shall each be set to 0x00, the MSDU parameter shall contain the MHR concatenated with the MAC payload (see Figure 26), and the msduLength parameter shall contain the total number of octets in the MHR concatenated with the MAC payload. The mpduLinkQuality shall be valid.

If the MLME is requested to set *macPromiscuousMode* to FALSE, the MLME shall request that the PHY set its receiver to the state specified by *macRxOnWhenIdle*. This is achieved by the MLME issuing the PLME-SET-TRX-STATE.request primitive (see 6.2.2.7) with the state set accordingly.

7.6.6.6 Transmission scenarios

Due to the imperfect nature of the radio medium, a transmitted frame does not always reach its intended destination. Figure 60 illustrates three different data transmission scenarios:

— *Successful data transmission.* The originator MAC sublayer transmits the data frame to the recipient via the PHY data service. In waiting for an acknowledgment, the originator MAC sublayer starts a timer that will expire after *macAckWaitDuration* symbols. The recipient MAC sublayer receives the data frame, sends an acknowledgment back to the originator, and passes the data frame to the next higher layer. The originator MAC sublayer receives the acknowledgment from the recipient before its timer expires and then disables and resets the timer. The data transfer is now complete, and the originator MAC sublayer issues a success confirmation to the next higher layer.

— *Lost data frame.* The originator MAC sublayer transmits the data frame to the recipient via the PHY data service. In waiting for an acknowledgment, the originator MAC sublayer starts a timer that will expire after *macAckWaitDuration* symbols. The recipient MAC sublayer does not receive the data frame and so does not respond with an acknowledgment. The timer of the originator MAC sublayer expires before an acknowledgment is received; therefore, the data transfer has failed. If the transmission was direct, the originator retransmits the data, and this entire sequence may be repeated up to a maximum of *macMaxFrameRetries* times; if a data transfer attempt fails a total of (1 + *macMaxFrameRetries*) times, the originator MAC sublayer will issue a failure confirmation to the next higher layer. If the transmission was indirect, the data frame will remain in the transaction queue until either another request for the data is received and correctly acknowledged or until *macTransactionPersistenceTime* is reached. If *macTransactionPersistenceTime* is reached, the transaction information will be discarded, and the MAC sublayer will issue a failure confirmation to the next higher layer.

— *Lost acknowledgment frame.* The originator MAC sublayer transmits the data frame to the recipient via the PHY data service. In waiting for an acknowledgment, the originator MAC sublayer starts a timer that will expire after *macAckWaitDuration* symbols. The recipient MAC sublayer receives the data frame, sends an acknowledgment back to the originator, and passes the data frame to the next higher layer. The originator MAC sublayer does not receive the acknowledgment frame, and its timer expires. Therefore, the data transfer has failed. If the transmission was direct, the originator retransmits the data, and this entire sequence may be repeated up to a maximum of *macMaxFrameRetries* times. If a data transfer attempt fails a total of (1 + *macMaxFrameRetries*) times, the originator MAC sublayer will issue a failure confirmation to the next higher layer. If the transmission was indirect, the data frame will remain in the transaction queue either until another request for the data is received and correctly acknowledged or until *macTransactionPersistenceTime* is reached. If *macTransactionPersistenceTime* is reached, the transaction information will be discarded, and the MAC sublayer will issue a failure confirmation to the next higher layer.

7.6.7 GTS allocation and management

A GTS allows a device to operate on the channel within a portion of the superframe that is dedicated (on the VAN) exclusively to that device. A GTS shall be allocated only by the VAN coordinator, and it shall be used only for communications between the VAN coordinator and a device associated with the VAN through the VAN coordinator. A single GTS may extend over one or more superframe slots. The VAN coordinator may allocate up to seven GTSs at the same time, provided there is sufficient capacity in the superframe.

A GTS shall be allocated before use, with the VAN coordinator deciding whether to allocate a GTS based on the requirements of the GTS request and the current available capacity in the superframe. GTSs shall be allocated on a first-come-first-served basis, and all GTSs shall be placed contiguously at the end of the superframe and after the CAP. Each GTS shall be deallocated when the GTS is no longer required, and a GTS can be deallocated at any time at the discretion of the VAN coordinator or by the device that originally requested the GTS. A device that has been allocated a GTS may also operate in the CAP.

A data frame transmitted in an allocated GTS shall use only short addressing.

The management of GTSs shall be undertaken by the VAN coordinator only. To facilitate GTS management, the VAN coordinator shall be able to store all the information necessary to manage seven GTSs. For each GTS, the VAN coordinator shall be able to store its starting slot, length, direction, and associated device address.

The GTS direction, which is relative to the data flow from the device that owns the GTS, is specified as either transmit or receive. The device address and direction shall, therefore, uniquely identify each GTS. Each device may request one transmit GTS and/or one receive GTS. For each allocated GTS, the device shall be able to store its starting slot, length, and direction. If a device has been allocated a receive GTS, it shall enable its receiver for the entirety of the GTS. In the same way, the VAN coordinator shall enable its

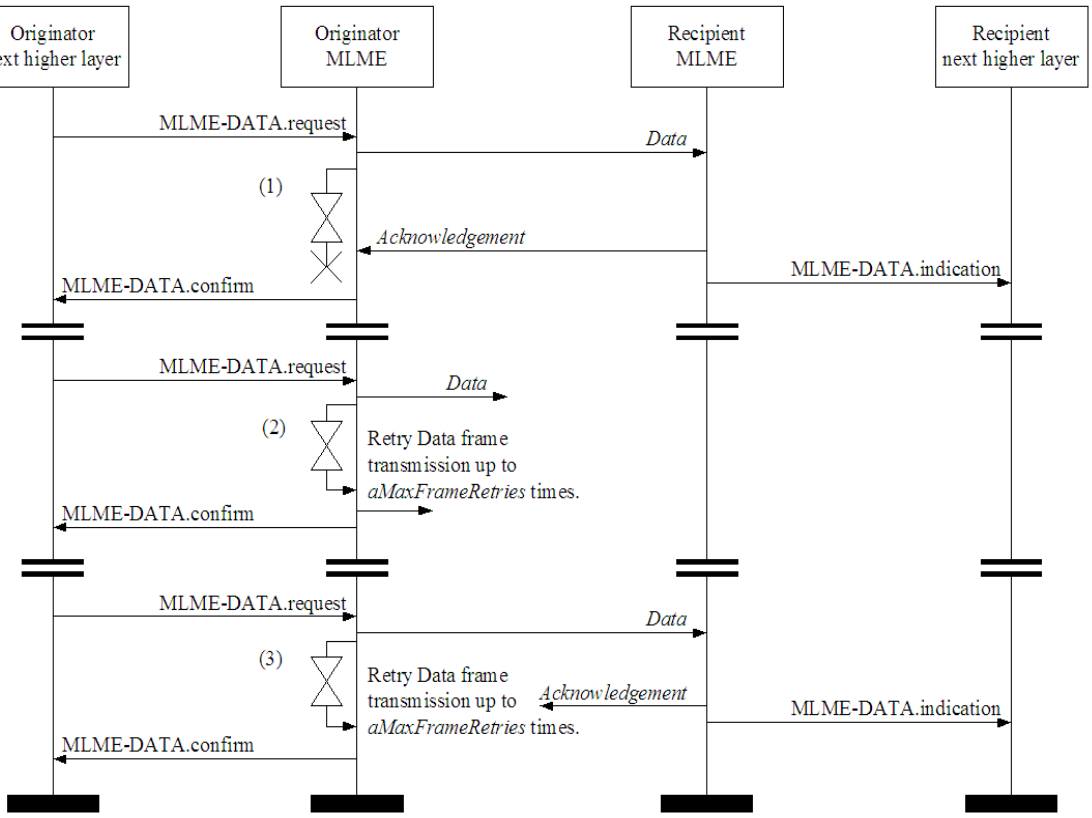


Figure 60—Transmission scenarios, using direct transmission, for frame reliability

receiver for the entirety of the GTS if a device has been allocated a transmit GTS. If a data frame is received during a receive GTS and an acknowledgment is requested, the device shall transmit the acknowledgment frame as usual. Similarly, a device shall be able to receive an acknowledgment frame during a transmit GTS.

A device shall attempt to allocate and use a GTS only if it is currently tracking the beacons. The MLME is instructed to track beacons by issuing the MLME-SYNC.request primitive with the TrackBeacon parameter set to TRUE. If a device loses synchronization with the VAN coordinator, all its GTS allocations shall be lost.

The use of GTSs is optional.

7.6.7.1 CAP maintenance

The VAN coordinator shall preserve the minimum CAP length of *aMinCAPLength* and take preventative action if the minimum CAP is not satisfied. However, an exception shall be allowed for the accommodation of the temporary increase in the beacon frame length needed to perform GTS maintenance. If preventative action becomes necessary, the action chosen is left up to the implementation, but may include one or more of the following:

— Limiting the number of pending addresses included in the beacon.

— Not including a payload field in the beacon frame.

— Deallocating one or more of the GTSs.

7.6.7.2 GTS allocation

A device is instructed to request the allocation of a new GTS through the MLME-GTS.request primitive, with GTS characteristics set according to the requirements of the intended application.

To request the allocation of a new GTS, the MLME shall send the GTS request command (see 7.4.10) to the VAN coordinator. The Characteristics Type subfield of the GTS Characteristics field of the request shall be set to one (GTS allocation), and the length and direction subfields shall be set according to the desired characteristics of the required GTS. Because the GTS request command contains an acknowledgment request (see 7.4.3.1), the VAN coordinator shall confirm its receipt by sending an acknowledgment frame.

On receipt of a GTS request command indicating a GTS allocation request, the VAN coordinator shall first check if there is available capacity in the current superframe, based on the remaining length of the CAP and the desired length of the requested GTS. The superframe shall have available capacity if the maximum number of GTSs has not been reached and allocating a GTS of the desired length would not reduce the length of the CAP to less than *aMinCAPLength*. GTSs shall be allocated on a first-come-first-served basis by the VAN coordinator provided there is sufficient bandwidth available. The VAN coordinator shall make this decision within *aGTSDescPersistenceTime* superframes.

On receipt of the acknowledgment to the GTS request command, the device shall continue to track beacons and wait for at most *aGTSDescPersistenceTime* superframes. If no GTS descriptor for the device appears in the beacon within this time, the MLME of the device shall notify the next higher layer of the failure. This notification is achieved when the MLME issues the MLME-GTS.confirm primitive (see 7.1.7.2) with a status of NO_DATA.

When the VAN coordinator determines whether capacity is available for the requested GTS, it shall generate a GTS descriptor with the requested specifications and the 16-bit short address of the requesting device. If the GTS was allocated successfully, the VAN coordinator shall set the start slot in the GTS descriptor to the superframe slot at which the GTS begins and the length in the GTS descriptor to the length of the GTS. In addition, the VAN coordinator shall notify the next higher layer of the new GTS. This notification is achieved when the MLME of the VAN coordinator issues the MLME-GTS.indication primitive (see 7.1.7.3) with the characteristics of the allocated GTS. If there was not sufficient capacity to allocate the requested GTS, the start slot shall be set to zero and the length to the largest GTS length that can currently be supported. The VAN coordinator shall then include this GTS descriptor in its beacon and update the GTS Specification field of the beacon frame accordingly. The VAN coordinator shall also update the Final CAP Slot subfield of the Superframe Specification field of the beacon frame, indicating the final superframe slot utilized by the decreased CAP. The GTS descriptor shall remain in the beacon frame for *aGTSDescPersistenceTime* superframes, after which it shall be removed automatically. The VAN coordinator shall be allowed to reduce its CAP below *aMinCAPLength* to accommodate the temporary increase in the beacon frame length due to the inclusion of the GTS descriptor.

On receipt of a beacon frame containing a GTS descriptor corresponding to *macShortAddress*, the device shall process the descriptor. The MLME of the device shall then notify the next higher layer of whether the GTS allocation request was successful. This notification is achieved when the MLME issues the MLME-GTS.confirm primitive with a status of SUCCESS (if the start slot in the GTS descriptor was greater than zero) or DENIED (if the start slot was equal to zero or if the length did not match the requested length).

7.6.7.3 GTS usage

When the MAC sublayer of a device that is not the VAN coordinator receives an MCPS-DATA.request primitive (see 7.1.1.1) with the TxOptions parameter indicating a GTS transmission, it shall determine whether it has a valid transmit GTS. If a valid GTS is found, the MAC sublayer shall transmit the data during the GTS, i.e., between its starting slot and its starting slot plus its length. At this time, the MAC sublayer shall transmit the MPDU immediately without using random access, provided the requested

transaction can be completed before the end of the GTS. If the requested transaction cannot be completed before the end of the current GTS, the MAC sublayer shall defer the transmission until the specified GTS in the next superframe. Note that the MAC must allow for the PHY overhead (see 6.9.2.2) in making this determination.

If the device has any receive GTSs, the MAC sublayer of the device shall ensure that the receiver is enabled at a time prior to the start of the GTS and for the duration of the GTS, as indicated by its starting slot and its length.

When the MAC sublayer of the VAN coordinator receives an MCPS-DATA.request primitive with the TxOptions parameter indicating a GTS transmission, it shall determine whether it has a valid receive GTS corresponding to the device with the requested destination address. If a valid GTS is found, the VAN coordinator shall defer the transmission until the start of the receive GTS. In this case, the address of the device with the message requiring a GTS transmission shall not be added to the list of pending addresses in the beacon frame (see 7.6.5). At the start of the receive GTS, the MAC sublayer shall transmit the data without using random access, provided the requested transaction can be completed before the end of the GTS. If the requested transaction cannot be completed before the end of the current GTS, the MAC sublayer shall defer the transmission until the specified GTS in the next superframe.

For all allocated transmit GTSs (relative to the device), the MAC sublayer of the VAN coordinator shall ensure that its receiver is enabled at a time prior to the start and for the duration of each GTS.

Before commencing transmission in a GTS, each device shall ensure that the data transmission, the acknowledgment, if requested, and the IFS, suitable to the size of the data frame, can be completed before the end of the GTS.

If a device misses the beacon at the beginning of a superframe, it shall not use its GTSs until it receives a subsequent beacon correctly. If a loss of synchronization occurs due to the loss of the beacon, the device shall consider all of its GTSs deallocated.

7.6.7.4 GTS deallocation

A device is instructed to request the deallocation of an existing GTS through the MLME-GTS.request primitive (see 7.1.7.1), using the characteristics of the GTS it wishes to deallocate. From this point onward, the GTS to be deallocated shall not be used by the device, and its stored characteristics shall be reset.

To request the deallocation of an existing GTS, the MLME shall send the GTS request command (see 7.4.10) to the VAN coordinator. The Characteristics Type subfield of the GTS Characteristics field of the request shall be set to zero (i.e., GTS deallocation), and the length and direction subfields shall be set according to the characteristics of the GTS to deallocate. Because the GTS request command contains an acknowledgment request (see 7.4.3.1), the VAN coordinator shall confirm its receipt by sending an acknowledgment frame. On receipt of the acknowledgment to the GTS request command, the MLME shall notify the next higher layer of the deallocation. This notification is achieved when the MLME issues the MLME-GTS.confirm primitive (see 7.1.7.2) with a status of SUCCESS and a GTSCharacteristics parameter with its Characteristics Type subfield set to zero. If the GTS request command is not received correctly by the VAN coordinator, it shall determine that the device has stopped using its GTS by the procedure described in 7.6.7.6.

On receipt of a GTS request command with the Characteristics Type subfield of the GTS Characteristics field set to zero (GTS deallocation), the VAN coordinator shall attempt to deallocate the GTS. If the GTS characteristics contained in the GTS request command do not match the characteristics of a known GTS, the VAN coordinator shall ignore the request. If the GTS characteristics contained in the GTS request command match the characteristics of a known GTS, the MLME of the VAN coordinator shall deallocate the specified GTS and notify the next higher layer of the change. This notification is achieved when the MLME issues the

MLME-GTS.indication primitive (see 7.1.7.3) with a GTSCharacteristics parameter containing the characteristics of the deallocated GTS and a Characteristics Type subfield set to zero. The VAN coordinator shall also update the Final CAP Slot subfield of the Superframe Specification field of the beacon frame, indicating the final superframe slot utilized by the increased CAP. It shall not add a descriptor to the beacon frame to describe the deallocation.

GTS deallocation may be initiated by the VAN coordinator due to a deallocation request from the next higher layer, the expiration of the GTS (see 7.6.7.6), or maintenance required to maintain the minimum CAP length, *aMinCAPLength* (see 7.6.7.1).

When a GTS deallocation is initiated by the next higher layer of the VAN coordinator, the MLME shall receive the MLME-GTS.request primitive with the GTS Characteristics field of the request set to zero (i.e., GTS deallocation) and the length and direction subfields set according to the characteristics of the GTS to deallocate.

When a GTS deallocation is initiated by the VAN coordinator either due to the GTS expiring or due to CAP maintenance, the MLME shall notify the next higher layer of the change. This notification is achieved when the MLME issues the MLME-GTS.indication primitive with a GTSCharacteristics parameter containing the characteristics of the deallocated GTS and a Characteristics Type subfield set to zero.

In the case of any deallocation initiated by VAN coordinator, the VAN coordinator shall deallocate the GTS and add a GTS descriptor into its beacon frame corresponding to the deallocated GTS, but with its starting slot set to zero. The descriptor shall remain in the beacon frame for *aGTSDescPersistenceTime* superframes. The VAN coordinator shall be allowed to reduce its CAP below *aMinCAPLength* to accommodate the temporary increase in the beacon frame length due to the inclusion of the GTS descriptor.

On receipt of a beacon frame containing a GTS descriptor corresponding to *macShortAddress* and a start slot equal to zero, the device shall immediately stop using the GTS. The MLME of the device shall then notify the next higher layer of the deallocation. This notification is achieved when the MLME issues the MLME-GTS.indication primitive with a GTSCharacteristics parameter containing the characteristics of the deallocated GTS and a Characteristics Type subfield set to zero.

7.6.7.5 GTS reallocation

The deallocation of a GTS may result in the superframe becoming fragmented. For example, Figure 61 shows three stages of a superframe with allocated GTSs. In stage 1, three GTSs are allocated starting at slots 14, 10, and 8, respectively. If GTS 2 is now deallocated (stage 2), there will be a gap in the superframe during which nothing can happen. To solve this, GTS 3 will have to be shifted to fill the gap, thus increasing the size of the CAP (stage 3).

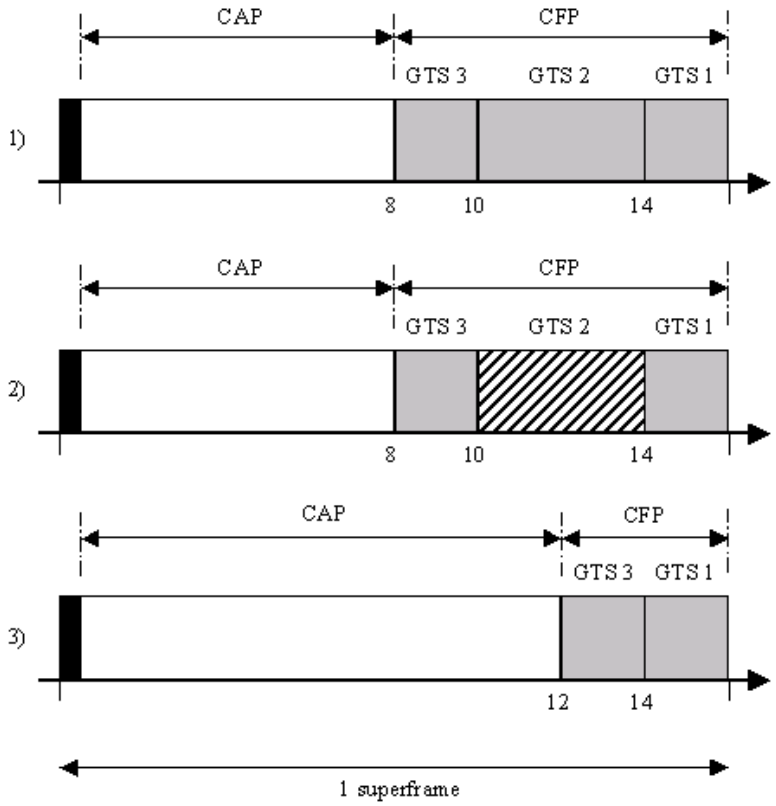


Figure 61—CFP defragmentation on GTS deallocations

The VAN coordinator shall ensure that any gaps occurring in the CFP, appearing due to the deallocation of a GTS, are removed to maximize the length of the CAP.

When a GTS is deallocated by the VAN coordinator, it shall add a GTS descriptor into its beacon frame indicating that the GTS has been deallocated. If the deallocation is initiated by a device, the VAN coordinator shall not add a GTS descriptor into its beacon frame to indicate the deallocation. For each device with an allocated GTS having a starting slot lower than the GTS being deallocated, the VAN coordinator shall update the GTS with the new starting slot and add a GTS descriptor to its beacon corresponding to this adjusted GTS. The new starting slot is computed so that no space is left between this GTS and either the end of the CFP, if the GTS appears at the end of the CFP, or the start of the next GTS in the CFP.

In situations where multiple reallocations occur at the same time, the VAN coordinator may choose to perform the reallocation in stages. The VAN coordinator shall keep each GTS descriptor in its beacon for *aGTSDescPersistenceTime* superframes.

On receipt of a beacon frame containing a GTS descriptor corresponding to *macShortAddress* and a direction and length corresponding to one of its GTSs, the device shall adjust the starting slot of the GTS corresponding to the GTS descriptor and start using it immediately.

In cases where it is necessary for the VAN coordinator to include a GTS descriptor in its beacon, it shall be allowed to reduce its CAP below *aMinCAPLength* to accommodate the temporary increase in the beacon frame length. After *aGTSDescPersistenceTime* superframes, the VAN coordinator shall remove the GTS descriptor from the beacon.

7.6.7.6 GTS expiration

The MLME of the VAN coordinator shall attempt to detect when a device has stopped using a GTS using the following rules:

— For a transmit GTS, the MLME of the VAN coordinator shall assume that a device is no longer using its GTS if a data frame is not received from the device in the GTS at least every 2**n* superframes, where *n* is defined below.

— For receive GTSs, the MLME of the VAN coordinator shall assume that a device is no longer using its GTS if an acknowledgment frame is not received from the device at least every 2**n* superframes, where *n* is defined below. If the data frames sent in the GTS do not require acknowledgment frames, the MLME of the VAN coordinator will not be able to detect whether a device is using its receive GTS. However, the VAN coordinator is capable of deallocating the GTS at any time.

The value of *n* is defined as follows:

$n = 2^{(8-macBeaconOrder)}$ $\quad 0 \le macBeaconOrder \le 8$

$n = 1$ $\quad 9 \le macBeaconOrder \le 14$

7.6.8 Frame security

[Editor's Note" this section and the security mechanism needs to be reviewed for its applicability to VLC]

The MAC sublayer is responsible for providing security services on specified incoming and outgoing frames when requested to do so by the higher layers. This standard supports the following security services (see 5.5.6 for definitions):

— Data confidentiality

— Data authenticity

— Replay protection

The information determining how to provide the security is found in the security-related PIB (see Table 69 in 7.7.1).

7.6.8.1 Security-related MAC PIB attributes

The security-related MAC PIB attributes contain:

— Key table (*macKeyTable*, *macKeyTableEntries*)

— Device table (*macDeviceTable*, *macDeviceTableEntries*)

— Minimum security level table (*macSecurityLevelTable*, *macSecurityLevelTableEntries*)

— Frame counter (*macFrameCounter*)

— Automatic request attributes (*macAutoRequestSecurityLevel*, *macAutoRequestKeyIdMode*, *macAutoRequestKeySource*, *macAutoRequestKeyIndex*)

— Default key source (*macDefaultKeySource*)

— VAN coordinator address (*macVANCoordExtendedAddress, macVANCoordShortAddress*)

7.6.8.1.1 Key table

The key table holds key descriptors (keys with related key-specific information) that are required for security processing of outgoing and incoming frames. Key-specific information in the key table is identified based on information explicitly contained in the requesting primitive or in the received frame, as described

in the outgoing frame key retrieval procedure (see 7.6.8.2.2) and the incoming frame security material retrieval procedure (see 7.6.8.2.4), as well as in the KeyDescriptor lookup procedure (see 7.6.8.2.5).

7.6.8.1.2 Device table

The device table holds device descriptors (device-specific addressing information and security-related information) that, when combined with key-specific information from the key table, provide all the keying material needed to secure outgoing (see 7.6.8.2.1) and unsecure incoming frames (see 7.6.8.2.3). Device-specific information in the device table is identified based on the originator of the frame, as described in the DeviceDescriptor lookup procedure (see 7.6.8.2.7), and on key-specific information, as described in the blacklist checking procedure (see 7.6.8.2.6).

7.6.8.1.3 Minimum security level table

The minimum security level table holds information regarding the minimum security level the device expects to have been applied by the originator of a frame, depending on frame type and, if it concerns a MAC command frame, the command frame identifier. Security processing of an incoming frame will fail if the frame is not adequately protected, as described in the incoming frame security procedure (see 7.6.8.2.3) and in the incoming security level checking procedure (see 7.6.8.2.8).

7.6.8.1.4 Frame counter

The 4-octet frame counter is used to provide replay protection and semantic security of the cryptographic building block used for securing outgoing frames. The frame counter is included in each secured frame and is one of the elements required for the unsecuring operation at the recipient(s). The frame counter is incremented each time an outgoing frame is secured, as described in the outgoing frame security procedure (see 7.6.8.2.1). When the frame counter reaches its maximum value of 0xffffffff, the associated keying material can no longer be used, thus requiring all keys associated with the device to be updated. This provides a mechanism for ensuring that the keying material for every frame is unique and, thereby, provides for sequential freshness.

7.6.8.1.5 Automatic request attributes

Automatic request attributes hold all the information needed to secure outgoing frames generated automatically and not as a result of a higher layer primitive, as is the case with automatic data requests.

7.6.8.1.6 Default key source

The default key source is information commonly shared between originator and receipient(s) of a secured frame, which, when combined with additional information explicitly contained in the requesting primitive or in the received frame, allows an originator or a recipient to determine the key required for securing or unsecuring this frame, respectively. This provides a mechanism for significantly reducing the overhead of security information contained in secured frames in particular use cases (see 7.6.8.2.2 and 7.6.8.2.4).

7.6.8.1.7 VAN coordinator address

The address of the VAN coordinator is information commonly shared between all devices in a VAN, which, when combined with additional information explicitly contained in the requesting primitive or in the received frame, allows an originator of a frame directed to the VAN coordinator or a recipient of a frame originating from the VAN coordinator to determine the key and security-related information required for securing or unsecuring, respectively, this frame (see 7.6.8.2.2 and 7.6.8.2.4).

7.6.8.2 Functional description

A device may optionally implement security. A device that does not implement security shall not provide a mechanism for the MAC sublayer to perform any cryptographic transformation on incoming and outgoing frames nor require any PIB attributes associated with security. A device that implements security shall provide a mechanism for the MAC sublayer to provide cryptographic transformations on incoming and outgoing frames using information in the PIB attributes associated with security when the *macSecurityEnabled* attribute is set to TRUE.

If the MAC sublayer is required to transmit a frame or receives an incoming frame, the MAC sublayer shall process the frame as specified in 7.6.8.2.1 and 7.6.8.2.3, respectively.

7.6.8.2.1 Outgoing frame security procedure

The inputs to this procedure are the frame to be secured and the SecurityLevel, KeyIdMode, KeySource, and KeyIndex parameters from the originating primitive or automatic request PIB attributes. The outputs from this procedure are the status of the procedure and, if this status is SUCCESS, the secured frame.

The outgoing frame security procedure involves the following steps as applicable:

a) If the Security Enabled subfield of the Frame Control field of the frame to be secured is set to zero, the procedure shall set the security level to zero.

b) If the Security Enabled subfield of the Frame Control field of the frame to be secured is set to one, the procedure shall set the security level to the SecurityLevel parameter. If the resulting security level is zero, the procedure shall return with a status of UNSUPPORTED_SECURITY.

c) If the *macSecurityEnabled* attribute is set to FALSE and the security level is not equal to zero, the procedure shall return with a status of UNSUPPORTED_SECURITY.

d) The procedure shall determine whether the frame to be secured satisfies the constraint on the maximum length of MAC frames, as follows:

   1) The procedure shall set the length *M*, in octets, of the Authentication field to zero if the security level is equal to zero and shall determine this value from the security level and Table 76 otherwise.

   2) The procedure shall determine the length AuxLen, in octets, of the auxiliary security header (see 7.7.2) using KeyIdMode and the security level.

   3) The procedure shall determine the data expansion as AuxLen + *M*.

   4) The procedure shall check whether the length of the frame to be secured, including data expansion and FCS, is less than or equal to *aMaxPHYPacketSize*. If this check fails, the procedure shall return with a status of FRAME_TOO_LONG.

e) If the security level is zero, the procedure shall set the secured frame to be the frame to be secured and return with the secured frame and a status of SUCCESS.

f) The procedure shall set the frame counter to the *macFrameCounter* attribute. If the frame counter has the value 0xffffffff, the procedure shall return with a status of COUNTER_ERROR.

g) The procedure shall obtain the key using the outgoing frame key retrieval procedure as described in 7.6.8.2.2. If that procedure fails, the procedure shall return with a status of UNAVAILABLE_KEY.

h) The procedure shall insert the auxiliary security header into the frame, with fields set as follows:

   1) The Security Level subfield of the Security Control field shall be set to the security level.

   2) The Key Identifier Mode subfield of the Security Control field shall be set to the KeyIdMode parameter.

   3) The Frame Counter field shall be set to the frame counter.

4) If the KeyIdMode parameter is set to a value not equal to zero, the Key Source and Key Index subfields of the Key Identifier field shall be set to the KeySource and KeyIndex parameters, respectively.

i) The procedure shall then use *aExtendedAddress*, the frame counter, the security level, and the key to produce the secured frame according to the transformation process known as CCM* [or the extension of CCM, which is the combined counter with CBC-MAC (i.e., cipher block chaining message authentication code) mode of operation] that is described in the security operations (see 7.7.3.4).

1) If the SecurityLevel parameter specifies the use of encryption (see Table 76 in 7.7.2.2.1), the encryption operation shall be applied only to the actual payload field within the MAC payload, i.e., the Beacon Payload field (see 7.2.2.1.8), Command Payload field (see 7.2.2.4.3), or Data Payload field (see 7.2.2.2.2), depending on the frame type. The corresponding payload field is passed to the CCM* transformation process described in 7.7.3.4 as the unsecured payload (see Table 78 in 7.7.3.4.2). The resulting encrypted payload shall substitute the original payload.

2) The remaining fields in the MAC payload part of the frame shall be passed to the CCM* transformation process described in 7.7.3.4 as the nonpayload fields (see Table 78).

3) The ordering and exact manner of performing the encryption and integrity operations and the placement of the resulting encrypted data or integrity code within the MAC Payload field shall be as defined in 7.7.3.4.

j) The procedure shall increment the frame counter by one and set the *macFrameCounter* attribute to the resulting value.

k) The procedure shall return with the secured frame and a status of SUCCESS.

7.6.8.2.2 Outgoing frame key retrieval procedure

The inputs to this procedure are the frame to be secured and the KeyIdMode, KeySource, and KeyIndex parameters from the originating primitive. The outputs from this procedure are a passed or failed status and, if passed, a key.

The outgoing frame key retrieval procedure involves the following steps as applicable:

a) If the KeyIdMode parameter is set to 0x00 (implicit key identification), the procedure shall determine the key lookup data and key lookup size as follows:

1) If the Destination Addressing Mode subfield of the Frame Control field of the frame is set to 0x00 and the *macVANCoordShortAddress* attribute is set to a value in the range 0x0000–0xfffd (i.e., the short address is used), the key lookup data shall be set to the 2-octet Source VAN Identifier field of the frame right-concatenated (see B.2.1) with the 2-octet *macVANCoordShortAddress* attribute right-concatenated with the single octet 0x00. The key lookup size shall be set to five.

2) If the Destination Addressing Mode subfield of the Frame Control field of the frame is set to 0x00 and the *macVANCoordShortAddress* attribute is set to 0xfffe (i.e., the extended address is used), the key lookup data shall be set to the 8-octet *macVANCoordExtendedAddress* attribute right-concatenated (see B.2.1) with the single octet 0x00. The key lookup size shall be set to nine.

3) If the Destination Addressing Mode subfield of the Frame Control field of the frame is set to 0x02, the key lookup data shall be set to the 2-octet Destination VAN Identifier field of the frame right-concatenated (see B.2.1) with the 2-octet Destination Address field of the frame right-concatenated with the single octet 0x00. The key lookup size shall be set to five.

4) If the Destination Addressing Mode subfield of the Frame Control field of the frame is set to 0x03, the key lookup data shall be set to the 8-octet Destination Address field of the frame right-concatenated (see B.2.1) with the single octet 0x00. The key lookup size shall be set to nine.

b) If the KeyIdMode parameter is set to a value not equal to 0x00 (explicit key identification), the procedure shall determine the key lookup data and key lookup size as follows:

   1) If the KeyIdMode parameter is set to 0x01, the key lookup data shall be set to the 8-octet *macDefaultKeySource* attribute right-concatenated (see B.2.1) with the single octet KeyIndex parameter. The key lookup size shall be set to nine.

   2) If the KeyIdMode parameter is set to 0x02, the key lookup data shall be set to the 4-octet KeySource parameter right-concatenated (see B.2.1) with the single octet KeyIndex parameter. The key lookup size shall be set to five.

   3) If the KeyIdMode parameter is set to 0x03, the key lookup data shall be set to the 8-octet KeySource parameter right-concatenated (see B.2.1) with the single octet KeyIndex parameter. The key lookup size shall be set to nine.

c) The procedure shall obtain the KeyDescriptor by passing the key lookup data and the key lookup size to the KeyDescriptor lookup procedure as described in 7.6.8.2.5. If that procedure returns with a failed status, this procedure shall also return with a failed status.

d) The MAC sublayer shall set the key to the Key element of the KeyDescriptor.

e) The procedure shall return with a passed status, having obtained the key identifier and the key.

NOTE—For broadcast frames, the outgoing frame key retrieval procedure will result in a failed status if implicit key identification is used. Hence, explicit key identification should be used for broadcast frames.[1]

7.6.8.2.3 Incoming frame security procedure

The input to this procedure is the frame to be unsecured. The outputs from this procedure are the unsecured frame, the security level, the key identifier mode, the key source, the key index, and the status of the procedure. All outputs of this procedure are assumed to be invalid unless and until explicitly set in this procedure. It is assumed that the PIB attributes associating KeyDescriptors in *macKeyTable* with a single, unique device or a number of devices will have been established by the next higher layer.

The incoming frame security procedure involves the following steps as applicable:

a) If the Security Enabled subfield of the Frame Control field of the frame to be unsecured is set to zero, the procedure shall set the security level to zero.

b) If the Security Enabled subfield of the Frame Control field of the frame to be unsecured is set to one and the Frame Version subfield of the Frame Control field of the frame to be unsecured is set to zero, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of UNSUPPORTED_LEGACY.

c) If the Security Enabled subfield of the Frame Control field of the frame to be unsecured is set to one, the procedure shall set the security level and the key identifier mode to the corresponding subfields of the Security Control field of the auxiliary security header of the frame to be unsecured and shall set the key source and key index to the corresponding subfields of the Key Identifier field of the auxiliary security header of the frame to be unsecured, if present. If the resulting security level is zero, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of UNSUPPORTED_SECURITY.

d) If the *macSecurityEnabled* attribute is set to FALSE, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of SUCCESS if the security level is equal to zero and with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of UNSUPPORTED_SECURITY otherwise.

[1]Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

e) The procedure shall determine whether the frame to be unsecured meets the minimum security level by passing the security level, the frame type, and, depending on whether the frame is a MAC command frame, the first octet of the MAC payload (i.e., command frame identifier for a MAC command frame) to the incoming security level checking procedure as described in 7.6.8.2.8. If that procedure fails, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of IMPROPER_SECURITY_LEVEL.

f) If the security level is set to zero, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of SUCCESS.

g) The procedure shall obtain the KeyDescriptor, DeviceDescriptor, and KeyDeviceDescriptor using the incoming frame security material retrieval procedure described in 7.6.8.2.4. If that procedure fails, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of UNAVAILABLE_KEY.

h) The procedure shall determine whether the frame to be unsecured conforms to the key usage policy by passing the KeyDescriptor, the frame type, and, depending on whether the frame is a MAC command frame, the first octet of the MAC payload (i.e., command frame identifier for a MAC command frame) to the incoming key usage policy checking procedure as described in 7.6.8.2.9. If that procedure fails, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of IMPROPER_KEY_TYPE.

i) If the Exempt element of the DeviceDescriptor is set to FALSE and if the incoming security level checking procedure of Step e) above had as output the “conditionally passed” status, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of IMPROPER_SECURITY_LEVEL.

j) The procedure shall set the frame counter to the Frame Counter field of the auxiliary security header of the frame to be unsecured. If the frame counter has the value 0xffffffff, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of COUNTER_ERROR.

k) The procedure shall determine whether the frame counter is greater than or equal to the FrameCounter element of the DeviceDescriptor. If this check fails, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of COUNTER_ERROR.

l) The procedure shall then use the ExtAddress element of the DeviceDescriptor, the frame counter, the security level, and the Key element of the KeyDescriptor to produce the unsecured frame according to the CCM* inverse transformation process described in the security operations (see 7.7.3.5).

   1) If the security level specifies the use of encryption (see Table 76 in 7.7.2.2.1), the decryption operation shall be applied only to the actual payload field within the MAC payload, i.e., the Beacon Payload field (see 7.2.2.1.8), Command Payload field (see 7.2.2.4.3), or Data Payload field (see 7.2.2.2.2), depending on the frame type. The corresponding payload field shall be passed to the CCM* inverse transformation process described in 7.7.3.5 as the secure payload.

   2) The remaining fields in the MAC payload part of the frame shall be passed to the CCM* inverse transformation process described in 7.7.3.5 as the nonpayload fields.

m) If the CCM* inverse transformation process fails, the procedure shall set the unsecured frame to be the frame to be unsecured and return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of SECURITY_ERROR.

n) The procedure shall increment the frame counter by one and set the FrameCounter element of the DeviceDescriptor to the resulting value.

o) If the FrameCounter element is equal to 0xffffffff, the procedure shall set the Blacklisted element of the KeyDeviceDescriptor.

p) The procedure shall return with the unsecured frame, the security level, the key identifier mode, the key source, the key index, and a status of SUCCESS.

7.6.8.2.4 Incoming frame security material retrieval procedure

The input to this procedure is the frame to be unsecured. The outputs from this procedure are a passed or failed status and, if passed, a KeyDescriptor, a DeviceDescriptor, and a KeyDeviceDescriptor.

The incoming frame security material retrieval procedure involves the following steps as applicable:

a) If the Key Identifier Mode subfield of the Security Control field of the auxiliary security header of the frame is set to 0x00 (implicit key identification), the procedure shall determine the key lookup data and the key lookup size as follows:

   1) If the source address mode of the Frame Control field of the frame is set to 0x00 and the *macVANCoordShortAddress* attribute is set to a value in the range 0x0000–0xfffd (i.e., the short address is used), the key lookup data shall be set to the 2-octet Destination VAN Identifier field of the frame right-concatenated (see B.2.1) with the 2-octet *macVANCoordShortAddress* attribute right-concatenated with the single octet 0x00. The key lookup size shall be set to five.

   2) If the source address mode of the Frame Control field of the frame is set to 0x00 and the *macVANCoordShortAddress* attribute is set to 0xfffe (i.e., the extended address is used), the key lookup data shall be set to the 8-octet *macVANCoordExtendedAddress* attribute right-concatenated (see B.2.1) with the single octet 0x00. The key lookup size shall be set to nine.

   3) If the source address mode of the Frame Control field of the frame is set to 0x02, the key lookup data shall be set to the 2-octet Source VAN Identifier field of the frame, or to the 2-octet Destination VAN Identifier field of the frame if the VAN ID Compression subfield of the Frame Control field of the frame is set to one, right-concatenated (see B.2.1) with the 2-octet Source Address field of the frame right-concatenated with the single octet 0x00. The key lookup size shall be set to five.

   4) If the source address mode of the Frame Control field of the frame is set to 0x03, the key lookup data shall be set to the 8-octet Source Address field of the frame right-concatenated (see B.2.1) with the single octet 0x00. The key lookup size shall be set to nine.

b) If the Key Identifier Mode subfield of the Security Control field of the auxiliary security header of the frame is set to a value not equal to 0x00 (explicit key identification), the procedure shall determine the key lookup data and key lookup size as follows:

   1) If the key identifier mode is set to 0x01, the key lookup data shall be set to the 8-octet *macDefaultKeySource* attribute right-concatenated (see B.2.1) with the 1-octet Key Index subfield of the Key Identifier field of the auxiliary security header. The key lookup size shall be set to nine.

   2) If the key identifier mode is set to 0x02, the key lookup data shall be set to the right-concatenation (see B.2.1) of the 4-octet Key Source subfield and the 1-octet Key Index subfield of the Key Identifier field of the auxiliary security header. The key lookup size shall be set to five.

   3) If the key identifier mode is set to 0x03, the key lookup data shall be set to the right-concatenation (see B.2.1) of the 8-octet Key Source subfield and the 1-octet Key Index subfield of the Key Identifier field of the auxiliary security header. The key lookup size shall be set to nine.

c) The procedure shall obtain the KeyDescriptor by passing the key lookup data and the key lookup size to the KeyDescriptor lookup procedure as described in 7.6.8.2.5. If that procedure returns with a failed status, the procedure shall also return with a failed status.

d) The procedure shall determine the device lookup data and the device lookup size as follows:

1) If the source address mode of the Frame Control field of the frame is set to 0x00 and the *macVANCoordShortAddress* attribute is set to a value in the range 0x0000–0xfffd (i.e., the short address is used), the device lookup data shall be set to the 2-octet Destination VAN Identifier field of the frame right-concatenated (see B.2.1) with the 2-octet *macVANCoordShortAddress* attribute. The device lookup size shall be set to four.
2) If the source address mode of the Frame Control field of the frame is set to 0x00 and the *macVANCoordShortAddress* attribute is set to 0xfffe (i.e., the extended address is used), the device lookup data shall be set to the 8-octet *macVANCoordExtendedAddress* attribute. The device lookup size shall be set to eight.
3) If the source address mode of the Frame Control field of the frame is set to 0x02, the device lookup data shall be set to the 2-octet Source VAN Identifier field of the frame, or to the 2-octet Destination VAN Identifier field of the frame if the VAN ID Compression subfield of the Frame Control field of the frame is set to one, right-concatenated (see B.2.1) with the 2-octet Source Address field of the frame. The device lookup size shall be set to four.
4) If the source address mode of the Frame Control field of the frame is set to 0x03, the device lookup data shall be set to the 8-octet Source Address field of the frame. The device lookup size shall be set to eight.

e) The procedure shall obtain the DeviceDescriptor and the KeyDeviceDescriptor by passing the KeyDescriptor, the device lookup data, and the device lookup size to the blacklist checking procedure as described in 7.6.8.2.6. If that procedure returns with a failed status, the procedure shall also return with a failed status.
f) The procedure shall return with a passed status having obtained the KeyDescriptor, the DeviceDescriptor, and the KeyDeviceDescriptor.

7.6.8.2.5 KeyDescriptor lookup procedure

The inputs to this procedure are the key lookup data and the key lookup size. The outputs from this procedure are a passed or failed status and, if passed, a KeyDescriptor.

The KeyDescriptor lookup procedure involves the following steps as applicable:

a) For each KeyDescriptor in the *macKeyTable* attribute and for each KeyIdLookupDescriptor in the KeyIdLookupList of the KeyDescriptor, the procedure shall check whether the LookupDataSize element of the KeyIdLookupDescriptor indicates the same integer value (see Figure 75) as the key lookup size and whether the LookupData element of the KeyIdLookupDescriptor is equal to the key lookup data. If both checks pass (i.e., there is a match), the procedure shall return with this (matching) KeyDescriptor and a passed status.
b) The procedure shall return with a failed status.

7.6.8.2.6 Blacklist checking procedure

The inputs to this procedure are the KeyDescriptor, the device lookup data, and the device lookup size. The outputs from this procedure are a passed or failed status and, if passed, a DeviceDescriptor and a KeyDeviceDescriptor.

The blacklist checking procedure involves the following steps as applicable:

a) For each KeyDeviceDescriptor in the KeyDeviceList of the KeyDescriptor:
    1) The procedure shall obtain the DeviceDescriptor using the DeviceDescriptorHandle element of the KeyDeviceDescriptor.
    2) If the UniqueDevice element of the KeyDeviceDescriptor is set to TRUE, the procedure shall return with the DeviceDescriptor, the KeyDeviceDescriptor, and a passed status if the

BlackListed element of the KeyDeviceDescriptor is set to FALSE, or the procedure shall return with a failed status if this Blacklisted element is set to TRUE.

3) If the UniqueDevice element of the KeyDeviceDescriptor is set to FALSE, the procedure shall execute the DeviceDescriptor lookup procedure as described in 7.6.8.2.7, with the device lookup data and the device lookup size as inputs. If the corresponding output of that procedure is a passed status, the procedure shall return with the DeviceDescriptor, the KeyDevice-Descriptor, and a passed status if the Blacklisted element of the KeyDeviceDescriptor is set to FALSE, or the procedure shall return with a failed status if this Blacklisted element is set to TRUE.

b) The procedure shall return with a failed status.

7.6.8.2.7 DeviceDescriptor lookup procedure

The inputs to this procedure are the DeviceDescriptor, the device lookup data, and the device lookup size. The output from this procedure is a passed or failed status.

The DeviceDescriptor lookup procedure involves the following steps as applicable:

a) If the device lookup size is four and the device lookup data is equal to the VAN ID element of the DeviceDescriptor right-concatenated (see B.2.1) with the ShortAddress element of the Device-Descriptor, this procedure shall return with a passed status.

b) If the device lookup size is eight and the device lookup data is equal to the ExtAddress element of the DeviceDescriptor, this procedure shall return with a passed status.

c) The procedure shall return with a failed status.

7.6.8.2.8 Incoming security level checking procedure

The inputs to this procedure are the incoming security level, the frame type and the command frame identifier. The output from this procedure is a passed, failed, or “conditionally passed” status.

The incoming security level checking procedure involves the following steps as applicable:

a) For each SecurityLevelDescriptor in the *macSecurityLevelTable* attribute:

1) If the frame type is not equal to 0x03 and the frame type is equal to the FrameType element of the SecurityLevelDescriptor, the procedure shall compare the incoming security level (as SEC1) with the SecurityMinimum element of the SecurityLevelDescriptor (as SEC2) according to the algorithm described in 7.7.2.2.1. If this comparison fails (i.e., evaluates to FALSE), the procedure shall return with a “conditionally passed” status if the DeviceOverrideSecurityMinimum element of the SecurityLevelDescriptor is set to TRUE and the security level is set to zero and with a failed status otherwise.

2) If the frame type is equal to 0x03, the frame type is equal to the FrameType element of the SecurityLevelDescriptor, and the command frame identifier is equal to the CommandFrame-Identifier element of the SecurityLevelDescriptor, the procedure shall compare the incoming security level (as SEC1) with the SecurityMinimum element of the SecurityLevelDescriptor (as SEC2) according to the algorithm described in 7.7.2.2.1. If this comparison fails (i.e., evaluates to FALSE), the procedure shall return with a “conditionally passed” status if the DeviceOverrideSecurityMinimum element of the SecurityLevelDescriptor is set to TRUE and the security level is set to zero and with a failed status otherwise.

b) The procedure shall return with a passed status.

7.6.8.2.9 Incoming key usage policy checking procedure

The inputs to this procedure are the KeyDescriptor, the frame type, and the command frame identifier. The output from this procedure is a passed or failed status.

The incoming key usage policy checking procedure involves the following steps as applicable:

a) For each KeyUsageDescriptor in the KeyUsageList of the KeyDescriptor:

   1) If the frame type is not equal to 0x03 and the frame type is equal to the FrameType element of the KeyUsageDescriptor, the procedure shall return with a passed status.

   2) If the frame type is equal to 0x03, the frame type is equal to the FrameType element of the KeyUsageDescriptor, and the command frame identifier is equal to the CommandFrame-Identifier element of the KeyUsageDescriptor, the procedure shall return with a passed status.

b) The procedure shall return with a failed status.

7.7 Security suite specifications

[Editor's Note: this section and the security mechanism needs to be reviewed for its applicability to VLC]

7.7.1 PIB security material

The PIB security-related attributes are presented in Table 69, Table 70, Table 71, Table 72, Table 73, Table 74, and Table 75.

Table 69— Security-related MAC PIB attributes

Attribute	Identifier	Type	Range	Description	Default
macKeyTable	0x71	List of Key-Descriptor entries (see Table 70)	—	A table of KeyDescriptor entries, each containing keys and related information required for secured communications.	(empty)
macKeyTableEntries	0x72	Integer	Implementa-tion specific	The number of entries in *macKeyTable*.	0
macDeviceTable	0x73	List of Device-Descriptor entries (see Table 74)	—	A table of Device-Descriptor entries, each indicating a remote device with which this device securely communicates.	(empty)
macDeviceTable-Entries	0x74	Integer	Implementa-tion specific	The number of entries in *macDeviceTable*.	0
macSecurity-LevelTable	0x75	Table of SecurityLevel Descriptor entries (see Table 73)	—	A table of SecurityLevel-Descriptor entries, each with information about the minimum security level expected depending on incoming frame type and subtype.	(empty)
macSecurity-LevelTableEntries	0x76	Integer	Implementa-tion specific	The number of entries in *macSecurityLevelTable*.	0
macFrameCounter	0x77	Integer	0x00000000–0xffffffff	The outgoing frame counter for this device.	0x00000000
macAutoRequest-SecurityLevel	0x78	Integer	0x00–0x07	The security level used for automatic data requests.	0x06

Table 69— Security-related MAC PIB attributes *(continued)*

Attribute	Identifier	Type	Range	Description	Default
macAutoRequest-KeyIdMode	0x79	Integer	0x00–0x03	The key identifier mode used for automatic data requests. This attribute is invalid if the *macAuto-RequestSecurityLevel* attribute is set to 0x00.	0x00
macAutoRequest-KeySource	0x7a	As specified by the *mac-AutoRequest-KeyIdMode* parameter	—	The originator of the key used for automatic data requests. This attribute is invalid if the *macAuto-RequestKeyIdMode* element is invalid or set to 0x00.	All octets 0xff
macAutoRequest-KeyIndex	0x7b	Integer	0x01–0xff	The index of the key used for automatic data requests. This attribute is invalid if the *macAuto-RequestKeyIdMode* attribute is invalid or set to 0x00.	All octets 0xff
macDefaultKey-Source	0x7c	Set of 8 octets	—	The originator of the default key used for key identifier mode 0x01.	All octets 0xff
macVANCoord-ExtendedAddress	0x7d	IEEE address	An extended 64-bit IEEE address	The 64-bit address of the VAN coordinator.	—
macVANCoordShort-Address	0x7e	Integer	0x0000–0xffff	The 16-bit short address assigned to the VAN coordinator. A value of 0xfffe indicates that the VAN coordinator is only using its 64-bit extended address. A value of 0xffff indicates that this value is unknown.	0x0000

Table 70—Elements of KeyDescriptor

Name	Type	Range	Description
KeyIdLookupList	List of KeyId-LookupDescriptor entries (see Table 75)	—	A list of KeyIdLookupDescriptor entries used to identify this KeyDescriptor.
KeyIdLookupListEntries	Integer	Implementation specific	The number of entries in KeyIdLookupList.
KeyDeviceList	List of KeyDevice-Descriptor entries (see Table 72)	—	A list of KeyDeviceDescriptor entries indicating which devices are currently using this key, including their blacklist status.

Table 70—Elements of KeyDescriptor *(continued)*

Name	Type	Range	Description
KeyDeviceListEntries	Integer	Implementation specific	The number of entries in KeyDeviceList.
KeyUsageList	List of KeyUsage-Descriptor entries (see Table 71)	—	A list of KeyUsageDescriptor entries indicating which frame types this key may be used with.
KeyUsageListEntries	Integer		The number of entries in KeyUsageList.
Key	Set of 16 octets	—	The actual value of the key.

Table 71—Elements of KeyUsageDescriptor

Name	Type	Range	Description
FrameType	Integer	0x00–0x03	See 7.2.1.1.1.
CommandFrameIdentifier	Integer	0x00–0x09	See Table 63.

Table 72—Elements of KeyDeviceDescriptor

Name	Type	Range	Description
DeviceDescriptorHandle	Integer	Implementa-tion specific	Handle to the DeviceDescriptor corresponding to the device (see Table 74).
UniqueDevice	Boolean	TRUE or FALSE	Indication of whether the device indicated by DeviceDescriptorHandle is uniquely associated with the KeyDescriptor, i.e., it is a link key as opposed to a group key.
Blacklisted	Boolean	TRUE or FALSE	Indication of whether the device indicated by DeviceDescriptorHandle previously communicated with this key prior to the exhaustion of the frame counter. If TRUE, this indicates that the device shall not use this key further because it exhausted its use of the frame counter used with this key.

7.7.2 Auxiliary security header

The Auxiliary Security Header field has a variable length and contains information required for security processing, including a Security Control field, a Frame Counter field, and a Key Identifier field. The Auxiliary Security Header field shall be present only if the Security Enabled subfield of the Frame Control field is set to one. The Auxiliary Security Header field shall be formatted as illustrated in Figure 62.

7.7.2.1 Integer and octet representation

The auxiliary security header is a MAC frame field (see 7.2.1.7) and, therefore, uses the representation conventions specified in 7.2.

Table 73—Elements of SecurityLevelDescriptor

Name	Type	Range	Description
FrameType	Integer	0x00–0x03	See 7.2.1.1.1.
CommandFrameIdentifier	Integer	0x00–0x09	See Table 63.
SecurityMinimum	Integer	0x00–0x07	The minimal required/expected security level for incoming MAC frames with the indicated frame type and, if present, command frame type (see Table 76 in 7.7.2.2.1).
DeviceOverrideSecurity-Minimum	Boolean	TRUE or FALSE	Indication of whether originating devices for which the Exempt flag is set may override the minimum security level indicated by the SecurityMinimum element. If TRUE, this indicates that for originating devices with Exempt status, the incoming security level zero is acceptable, in addition to the incoming security levels meeting the minimum expected security level indicated by the SecurityMinimum element.

Table 74—Elements of DeviceDescriptor

Name	Type	Range	Description
VANId	Device VAN ID	0x0000–0xffff	The 16-bit VAN identifier of the device in this DeviceDescriptor.
ShortAddress	Device short address	0x0000–0xffff	The 16-bit short address of the device in this DeviceDescriptor. A value of 0xfffe indicates that this device is using only its extended address. A value of 0xffff indicates that this value is unknown.
ExtAddress	IEEE address	Any valid 64-bit device address	The 64-bit IEEE extended address of the device in this DeviceDescriptor. This element is also used in unsecuring operations on incoming frames.
FrameCounter	Integer	0x00000000–0xffffffff	The incoming frame counter of the device in this DeviceDescriptor. This value is used to ensure sequential freshness of frames.
Exempt	Boolean	TRUE or FALSE	Indication of whether the device may override the minimum security level settings defined in Table 73.

7.7.2.2 Security Control field

The Security Control field is 1 octet in length and is used to provide information about what protection is applied to the frame. The Security Control field shall be formatted as shown in Figure 63.

Table 75—Elements of KeyIdLookupDescriptor

Name	Type	Range	Description
LookupData	Set of 5 or 9 octets	—	Data used to identify the key.
LookupDataSize	Integer	0x00–0x01	A value of 0x00 indicates a set of 5 octets; a value of 0x01 indicates a set of 9 octets.

Octets: 1	4	0/1/5/9
Security Control	Frame Counter	Key Identifier

Figure 62—Format of the auxiliary security header

Bit: 0–2	3–4	5–7
Security Level	Key Identifier Mode	Reserved

Figure 63—Security Control field format

7.7.2.2.1 Security Level subfield

The Security Level subfield is 3 bits in length and indicates the actual frame protection that is provided. This value can be adapted on a frame-by-frame basis and allows for varying levels of data authenticity (to allow minimization of security overhead in transmitted frames where required) and for optional data confidentiality. The cryptographic protection offered by the various security levels is shown in Table 76. When nontrivial protection is required, replay protection is always provided.

Table 76—Security levels available to the MAC sublayer

Security level identifier	Security Control field (Figure 63) b_2 b_1 b_0	Security attributes	Data confidentiality	Data authenticity (including length *M* of authentication tag, in octets)
0x00	'000'	None	OFF	NO (*M* = 0)
0x01	'001'	MIC-32	OFF	YES (*M* = 4)
0x02	'010'	MIC-64	OFF	YES (*M* = 8)
0x03	'011'	MIC-128	OFF	YES (*M* = 16)
0x04	'100'	ENC	ON	NO (*M* = 0)
0x05	'101'	ENC-MIC-32	ON	YES (*M* = 4)
0x06	'110'	ENC-MIC-64	ON	YES (*M* = 8)
0x07	'111'	ENC-MIC-128	ON	YES (*M* = 16)

Security levels can be ordered according to the corresponding cryptographic protection offered. Here, a first security level SEC1 is greater than or equal to a second security level SEC2 if and only if SEC1 offers at least the protection offered by SEC2, both with respect to data confidentiality and with respect to data authenticity. The statement "SEC1 is greater than or equal to SEC2" shall be evaluated as TRUE if both of the following conditions apply:

a) Bit position b_2 in SEC1 is greater than or equal to bit position b_2 in SEC2 (where Encryption OFF < Encryption ON).

b) The integer value of bit positions b_1 b_0 in SEC1 is greater than or equal to the integer value of bit positions b_1 b_0 in SEC2 (where increasing integer values indicate increasing levels of data authenticity provided, i.e., message integrity code (MIC)-0 < MIC-32 < MIC-64 < MIC-128).

Otherwise, the statement shall be evaluated as FALSE.

For example, ENC-MIC-64 ≥ MIC-64 is TRUE because ENC-MIC-64 offers the same data authenticity protection as MIC-64, plus confidentiality. On the other hand, MIC-128 ≥ ENC-MIC-64 is FALSE because even though MIC-128 offers stronger data authenticity than ENC-MIC-64, it offers no confidentiality.

7.7.2.2.2 Key Identifier Mode subfield

The Key Identifier Mode subfield is 2 bits in length and indicates whether the key that is used to protect the frame can be derived implicitly or explicitly; furthermore, it is used to indicate the particular representations of the Key Identifier field (see 7.7.2.4) if derived explicitly. The Key Identifier Mode subfield shall be set to one of the values listed in Table 77. The Key Identifier field of the auxiliary security header (see 7.7.2.4) shall be present only if this subfield has a value that is not equal to 0x00.

Table 77—Values of the key identifier mode

Key identifier mode	Key Identifier Mode subfield b_1 b_0	Description	Key Identifier field length (octets)
0x00	'00'	Key is determined implicitly from the originator and receipient(s) of the frame, as indicated in the frame header.	0
0x01	'01'	Key is determined from the 1-octet Key Index subfield of the Key Identifier field of the auxiliary security header in conjunction with *macDefaultKeySource*.	1
0x02	'10'	Key is determined explicitly from the 4-octet Key Source subfield and the 1-octet Key Index subfield of the Key Identifier field of the auxiliary security header.	5
0x03	'11'	Key is determined explicitly from the 8-octet Key Source subfield and the 1-octet Key Index subfield of the Key Identifier field of the auxiliary security header.	9

7.7.2.3 Frame Counter field

The Frame Counter field is 4 octets in length and represents the *macFrameCounter* attribute of the originator of a protected frame. It is used to provide semantic security of the cryptographic mechanism used to protect a frame and to offer replay protection.

7.7.2.4 Key Identifier field

The Key Identifier field has a variable length and identifies the key that is used for cryptographic protection of outgoing frames, either explicitly or in conjunction with implicitly defined side information. The Key Identifier field shall be present only if the Key Identifier Mode subfield of the Security Control field of the auxiliary security header (see 7.7.2.2.2) is set to a value different from 0x00. The Key Identifier field shall be formatted as illustrated in Figure 64.

Octets: 0/4/8	1
Key Source	Key Index

Figure 64—Format for the Key Identifier field, if present

7.7.2.4.1 Key Source subfield

The Key Source subfield, when present, is either 4 octets or 8 octets in length, according to the value specified by the Key Identifier Mode subfield of the Security Control field (see 7.7.2.2.2), and indicates the originator of a group key.

7.7.2.4.2 Key Index subfield

The Key Index subfield is 1 octet in length and allows unique identification of different keys with the same originator.

It is the responsibility of each key originator to make sure that actively used keys that it issues have distinct key indices and that the key indices are all different from 0x00.

7.7.3 Security operations

This subclause describes the parameters for the CCM* security operations, as specified in B.3.2.

7.7.3.1 Integer and octet representation

The integer and octet representation conventions specified in B.2 are used throughout 7.7.3.

7.7.3.2 CCM* Nonce

The CCM* nonce is a 13-octet string and is used for the advanced encryption standard (AES)-CCM* mode of operation (see B.2.2). The nonce shall be formatted as shown in Figure 65, with the leftmost field in the figure defining the first (and leftmost) octets and the rightmost field defining the last (and rightmost) octet of the nonce.

The source address shall be set to the extended address *aExtendedAddress* of the device originating the frame, the frame counter to the value of the respective field in the auxiliary security header (see 7.7.2), and

Octets: 8	4	1
Source address	Frame counter	Security level

Figure 65—CCM* nonce

the security level to the security level identifier corresponding to the Security Level subfield of the Security Control field of the auxiliary security header as defined in Table 76.

The source address, frame counter, and security level shall be represented as specified in 7.7.3.1.

7.7.3.3 CCM* prerequisites

Securing a frame involves the use of the CCM* mode encryption and authentication transformation, as described in B.4.1. Unsecuring a frame involves the use of the CCM* decryption and authentication checking process, as described in B.4.2. The prerequisites for the CCM* forward and inverse transformations are as follows:

— The underlying block cipher shall be the AES encryption algorithm as specified in B.3.1.

— The bit ordering shall be as defined in 7.7.3.1.

— The length in octets of the Length field *L* shall be 2 octets.

— The length of the Authentication field *M* shall be 0 octets, 4 octets, 8 octets, or 16 octets, as required.

7.7.3.3.1 Authentication field length

The length of the Authentication field *M* for the CCM* forward transformation and the CCM* inverse transformation is determined from Table 76, using the Security Level subfield of the Security Control field of the auxiliary security header of the frame.

7.7.3.4 CCM* transformation data representation

This subclause describes how the inputs and output of the CCM* forward transformation, as described in B.4.1, are formed:

The inputs are

— Key

— Nonce

— *a* data

— *m* data

The output is *c* data.

7.7.3.4.1 Key and nonce data inputs

The Key data for the CCM* forward transformation is passed by the outgoing frame security procedure described in 7.6.8.2.1. The Nonce data for the CCM* transformation is constructed as described in 7.7.3.2.

7.7.3.4.2 *a* data and *m* data

In the CCM* transformation process, the data fields shall be applied as in Table 78.

Table 78—*a* data and *m* data for all security levels

Security level identifier	*a* data	*m* data
0x00	None	None
0x01	MHR || Auxiliary security header || Nonpayload fields || Unsecured payload fields	None
0x02	MHR || Auxiliary security header || Nonpayload fields || Unsecured payload fields	None
0x03	MHR || Auxiliary security header || Nonpayload fields || Unsecured payload fields	None
0x04	None	Unsecured payload fields
0x05	MHR || Auxiliary security header || Nonpayload fields	Unsecured payload fields
0x06	MHR || Auxiliary security header || Nonpayload fields	Unsecured payload fields
0x07	MHR || Auxiliary security header || Nonpayload fields	Unsecured payload fields

7.7.3.4.3 *c* data output

In the CCM* transformation process, the data fields that are applied, or right-concatenated and applied, represent octet strings.

The secured payload fields right-concatenated with the authentication tag shall substitute the unsecured payload field in the original unsecured frame to form the secured frame (see Table 79).

Table 79—*c* data for all security levels

Security level identifier	*c* data
0x00	None
0x01	MIC-32
0x02	MIC-64
0x03	MIC-128
0x04	Secured payload fields
0x05	Secured payload fields || MIC-32
0x06	Secured payload fields || MIC-64
0x07	Secured payload fields || MIC-128

7.7.3.5 CCM* inverse transformation data representation

This subclause describes how the inputs and output of the CCM* inverse transformation, as described in B.4.2, are formed.

The inputs are

— Key

— Nonce
— *c* data
— *a* data

The output is *m* data.

7.7.3.5.1 Key and nonce data inputs

The Key data for the CCM* inverse transformation is passed by the incoming frame security procedure described in 7.6.8.2.3. The Nonce data for the CCM* transformation is constructed as described in 7.7.3.2.

7.7.3.5.2 *c* data and *a* data

In the CCM* inverse transformation process, the data fields shall be applied as in Table 80.

Table 80—*c* data and *a* data for all security levels

Security level identifier	*c* data	*a* data
0x00	None	None
0x01	MIC-32	MHR || Auxiliary security header || Nonpayload fields ||Secured payload fields
0x02	MIC-64	MHR || Auxiliary security header || Nonpayload fields || Secured payload fields
0x03	MIC-128	MHR || Auxiliary security header || Nonpayload fields || Secured payload fields
0x04	Secured payload fields	MHR || Auxiliary security header || Nonpayload fields
0x05	Secured payload fields || MIC-32	MHR || Auxiliary security header || Nonpayload fields
0x06	Secured payload fields || MIC-64	MHR || Auxiliary security header || Nonpayload fields
0x07	Secured payload fields || MIC-128	MHR || Auxiliary security header || Nonpayload fields

7.7.3.5.3 *m* data output

The *m* data shall then substitute secured payload fields and authentication tag in the original secured frame to form the unsecured frame.

7.8 Message sequence charts illustrating MAC-PHY interaction

[This section has not been updated to reflect changes made for VLC]

This subclause illustrates the main tasks specified in this standard. Each task is described by use of a message sequence chart to illustrate the chronological order, rather than the exact timing, of the primitives required for each task.

The primitives necessary for the VAN coordinator to start a new VAN are shown in Figure 66. The first action the next higher layer takes after resetting the MAC sublayer is to initiate a scan to search for other VANs in the area. An active scan is required, and an ED scan may optionally be performed. The steps for performing an active scan and an ED scan are shown in Figure 71 and Figure 67, respectively.

Once a new VAN is established, the VAN coordinator is ready to accept requests from other devices to join the VAN. Figure 68 shows the primitives issued by a device requesting association, while Figure 69 illustrates the steps taken by a coordinator allowing association. In the process of joining a VAN, the device requesting association will perform either a passive or an active scan to determine which VANs in the area are allowing association; Figure 70 and Figure 71 detail the primitives necessary to complete a passive scan and an active scan, respectively.

The primitives necessary for transmitting and receiving a single data packet are shown next. The actions taken by the originator of the packet are shown in Figure 72, while the actions taken by the recipient are shown in Figure 73.

When a device becomes unable to communicate to its coordinator any longer, the device can use an orphan scan to rediscover its coordinator. The primitives necessary for the realignment of an orphaned device are shown in Figure 74.

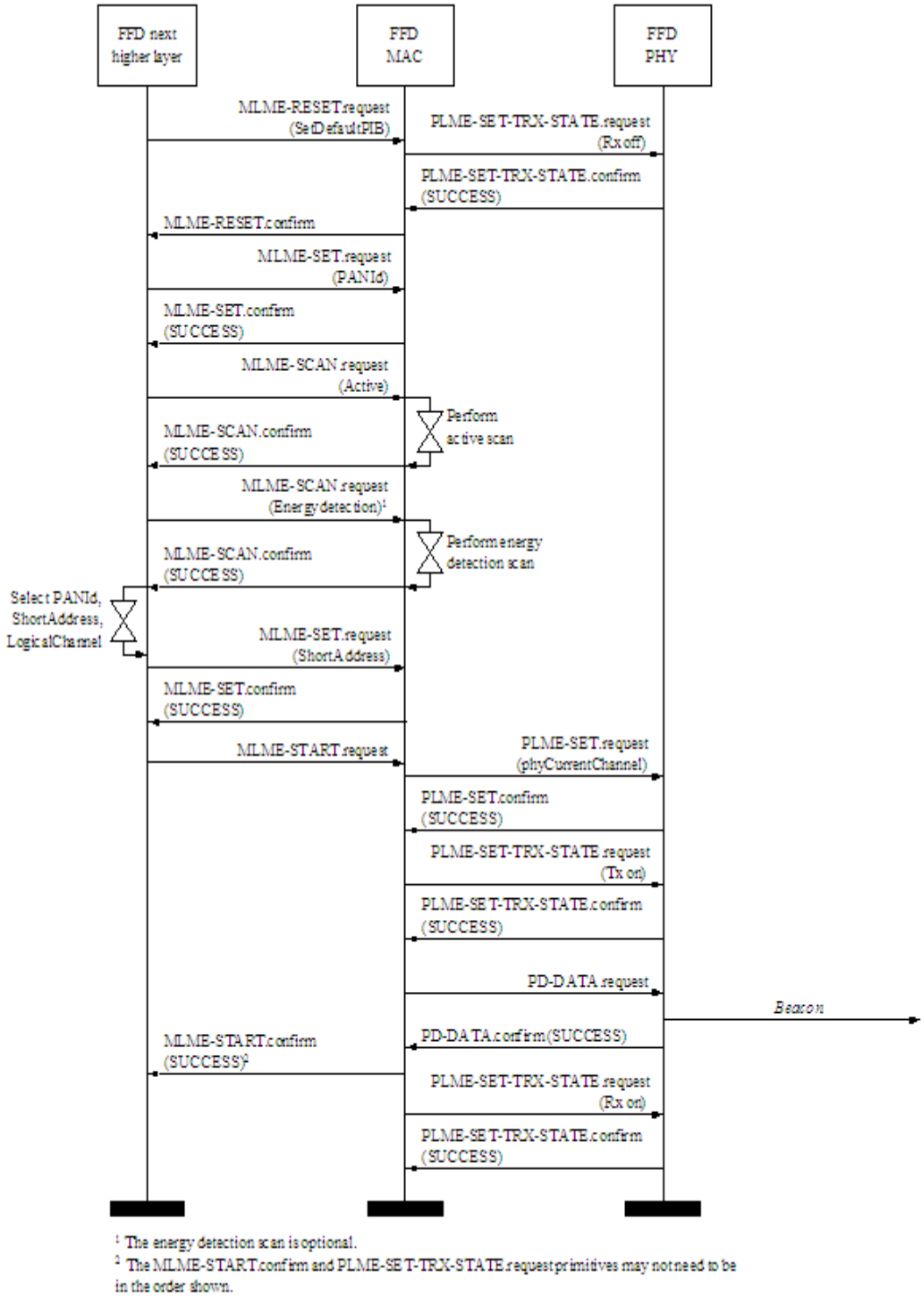


Figure 66—VAN start message sequence chart—VAN coordinator

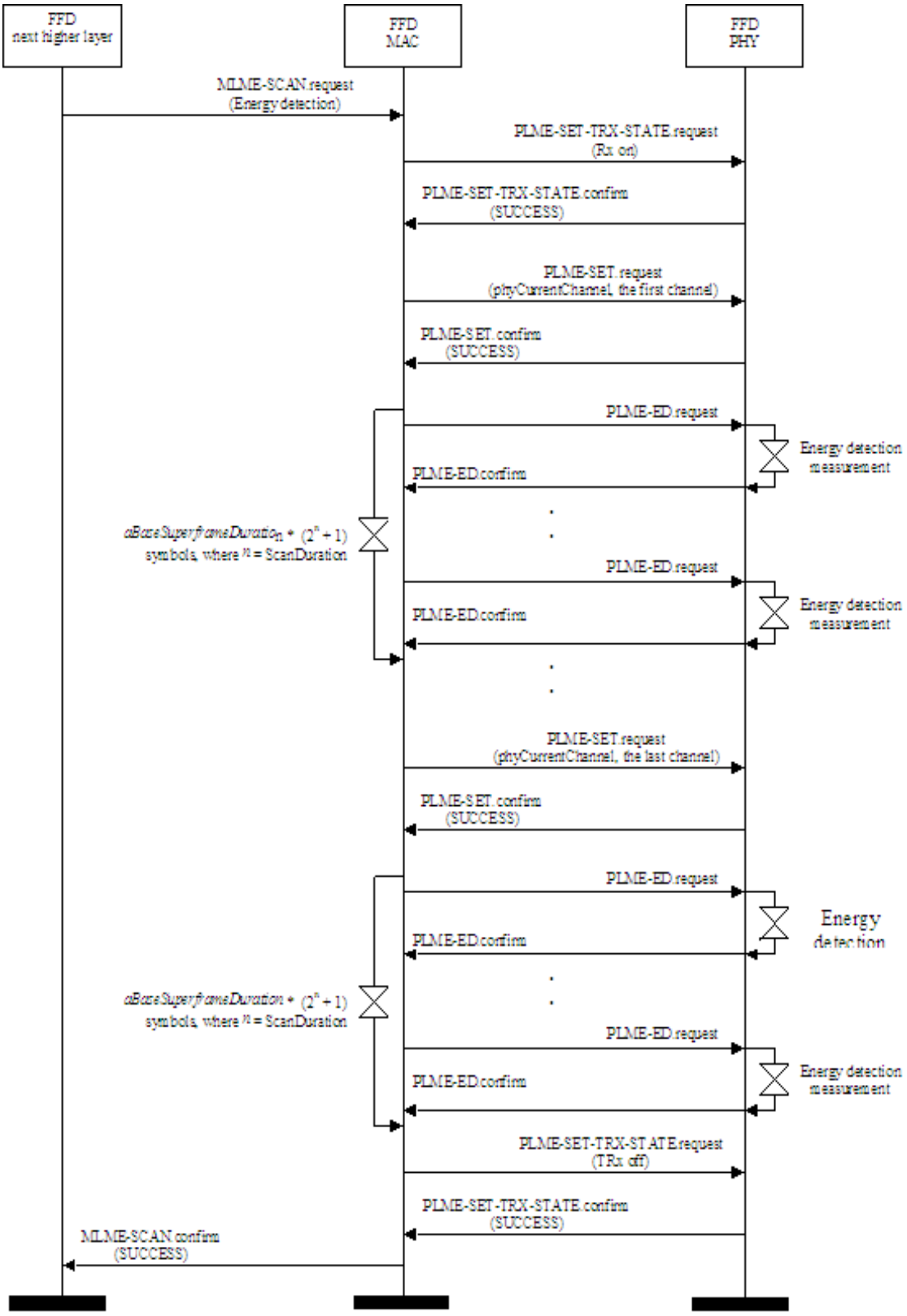


Figure 67—ED scan message sequence chart

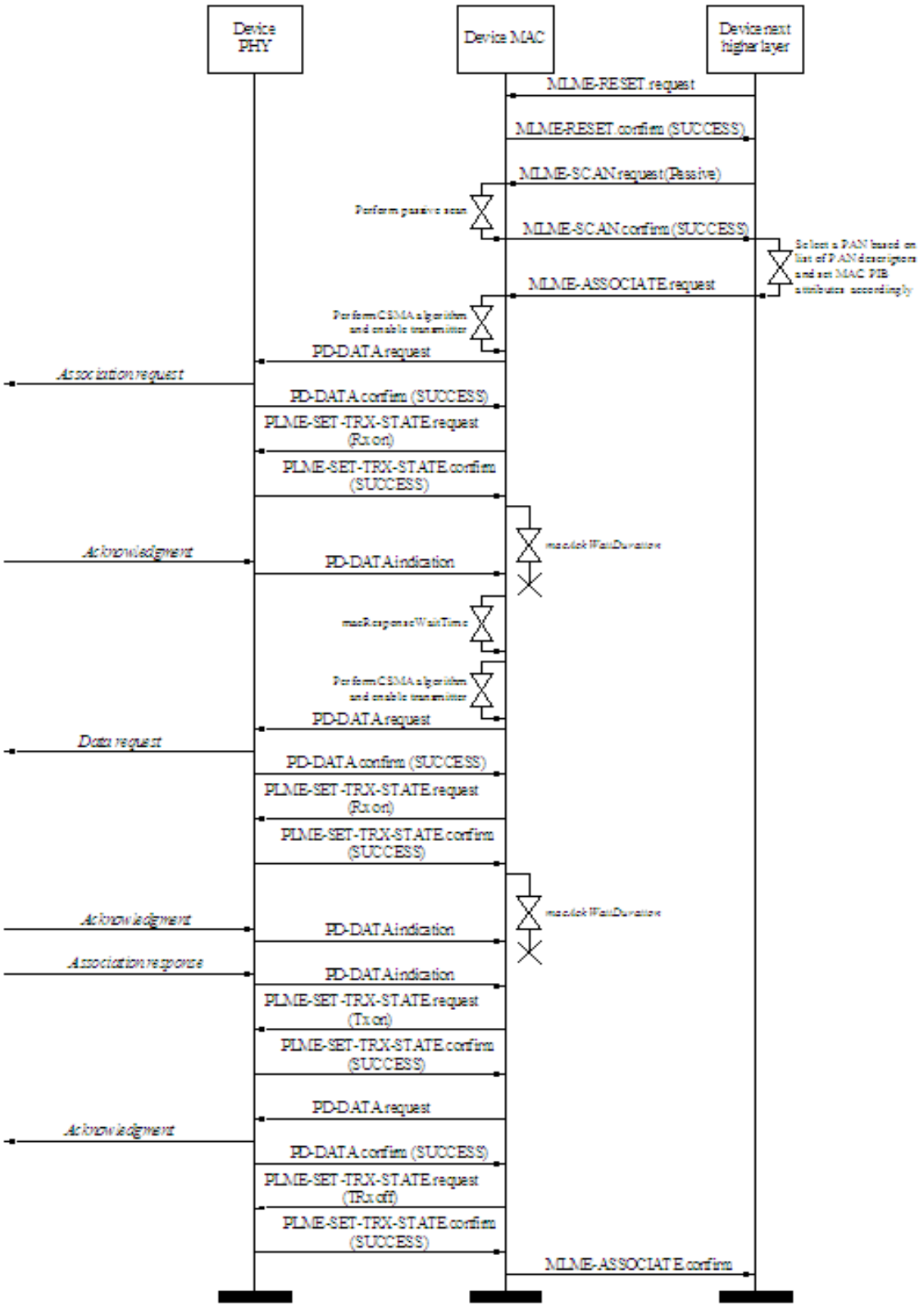


Figure 68—Association message sequence chart—device

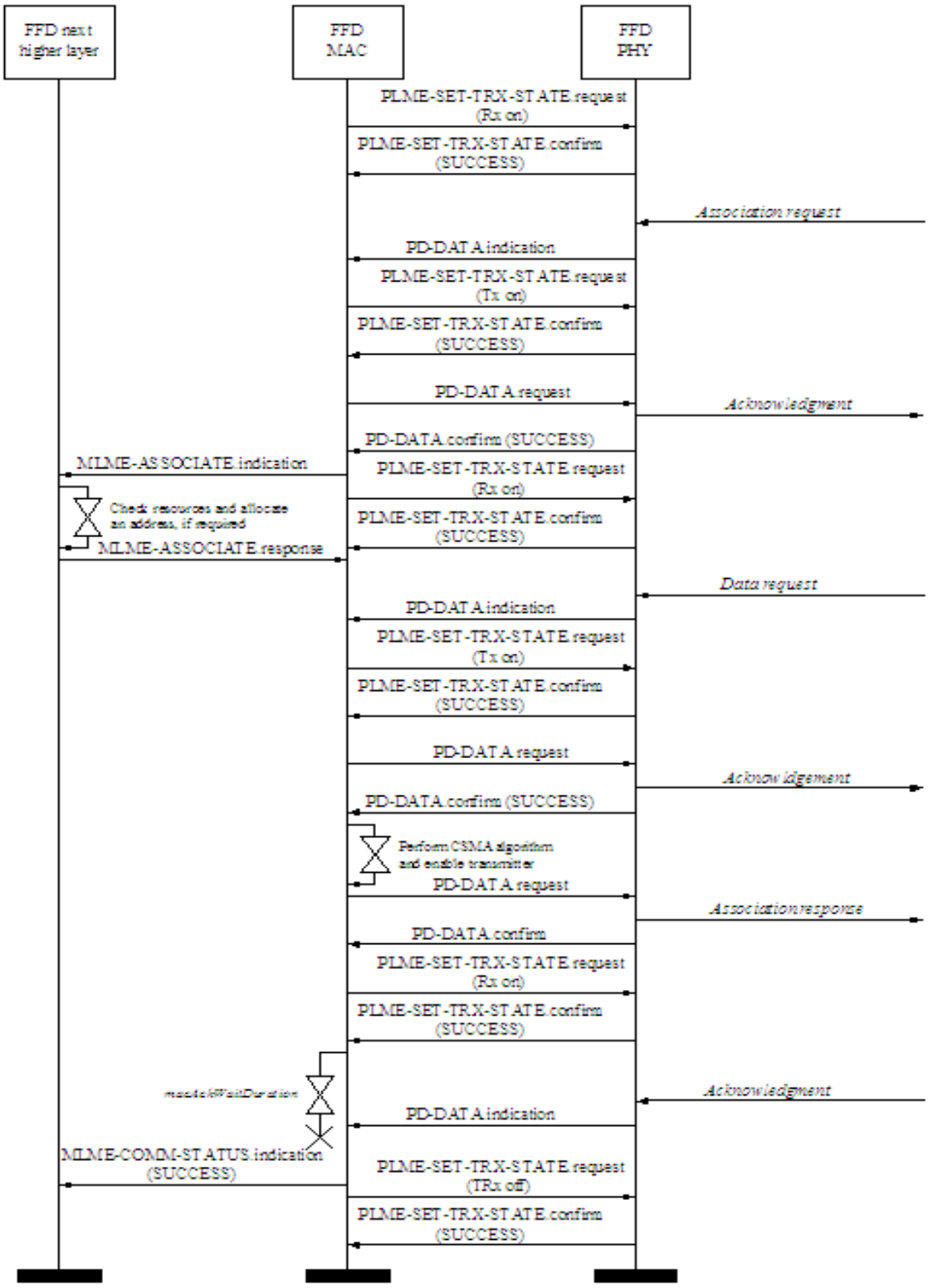


Figure 69—Association message sequence chart—coordinator

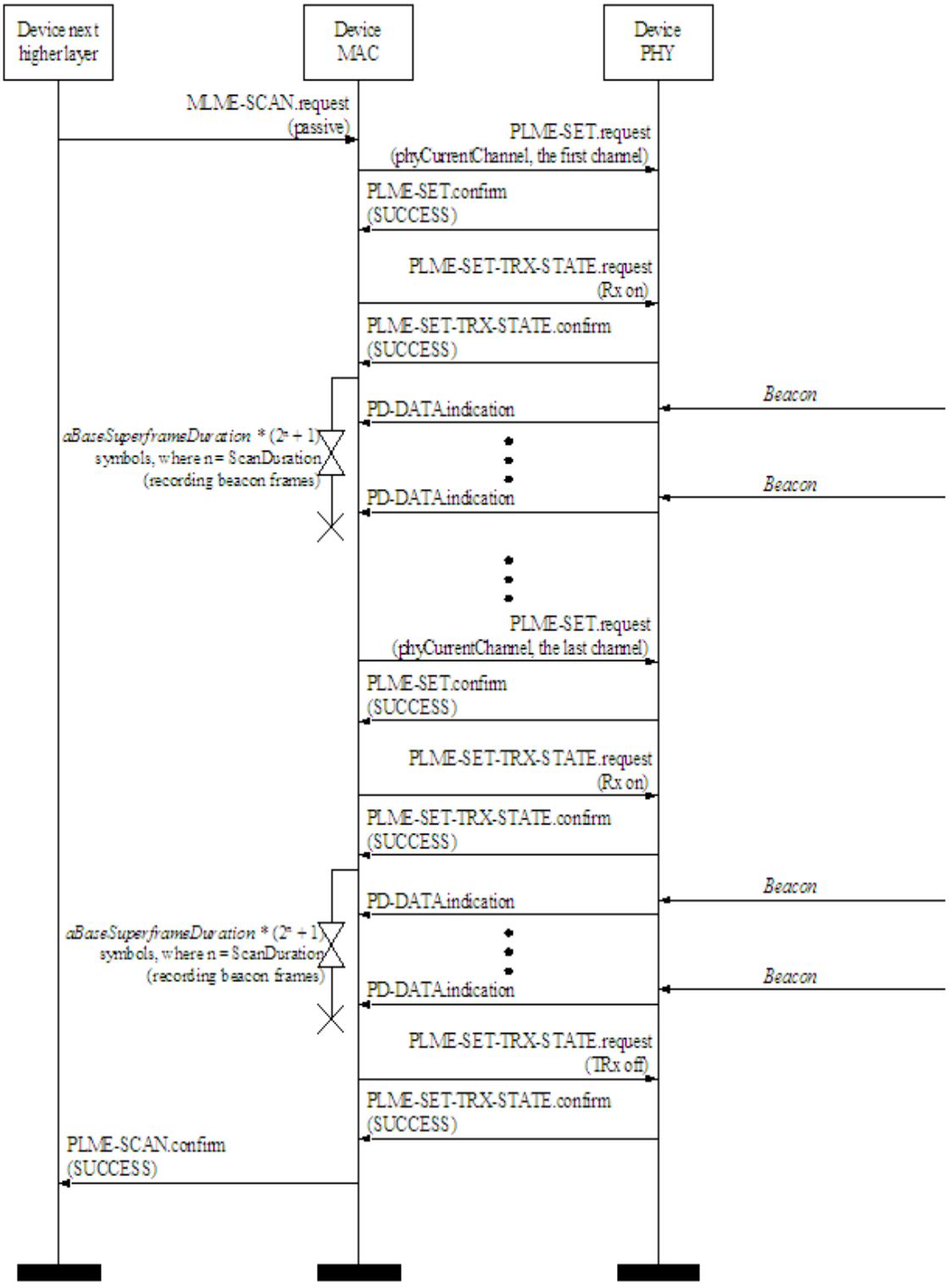


Figure 70—Passive scan message sequence chart

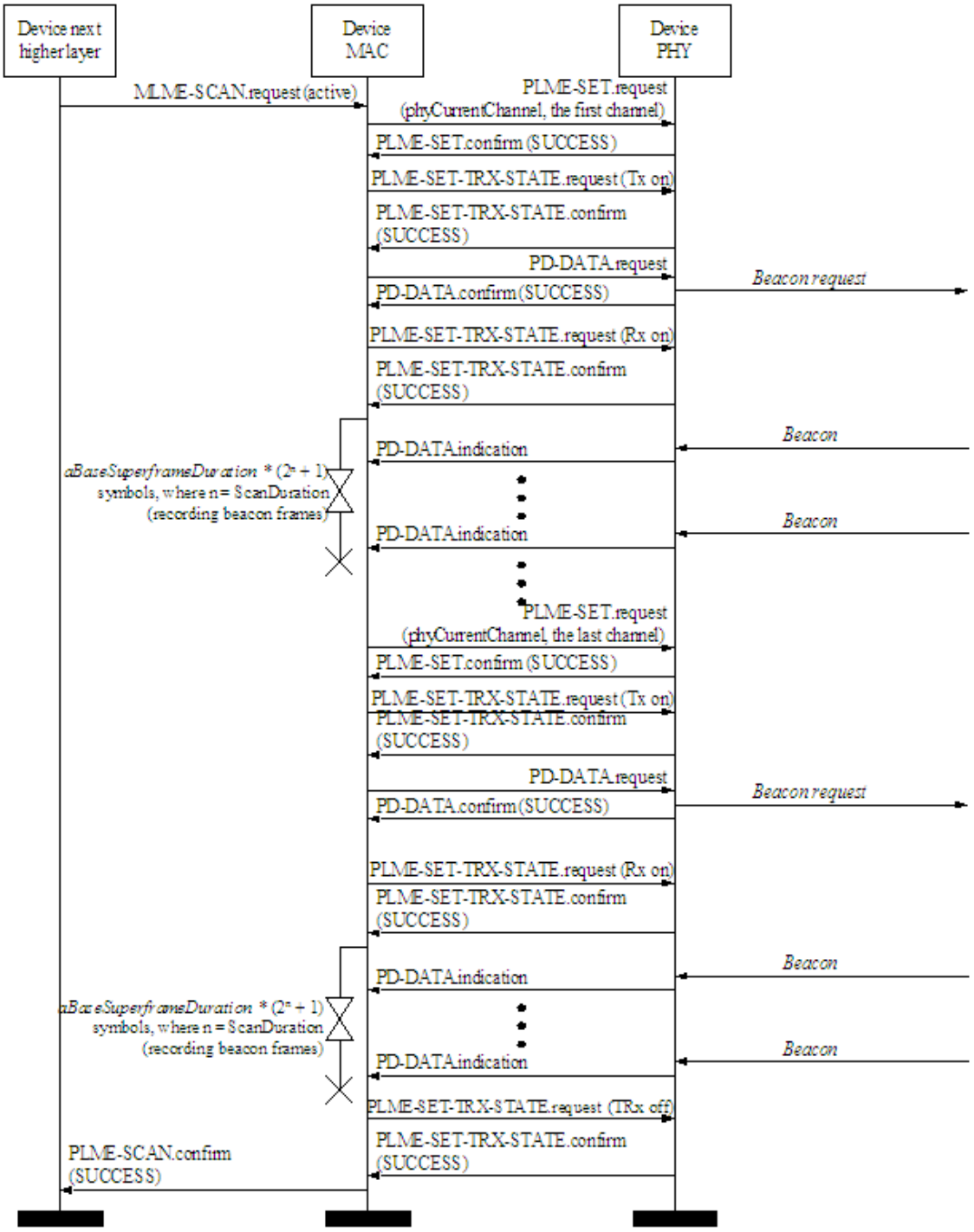


Figure 71—Active scan message sequence chart

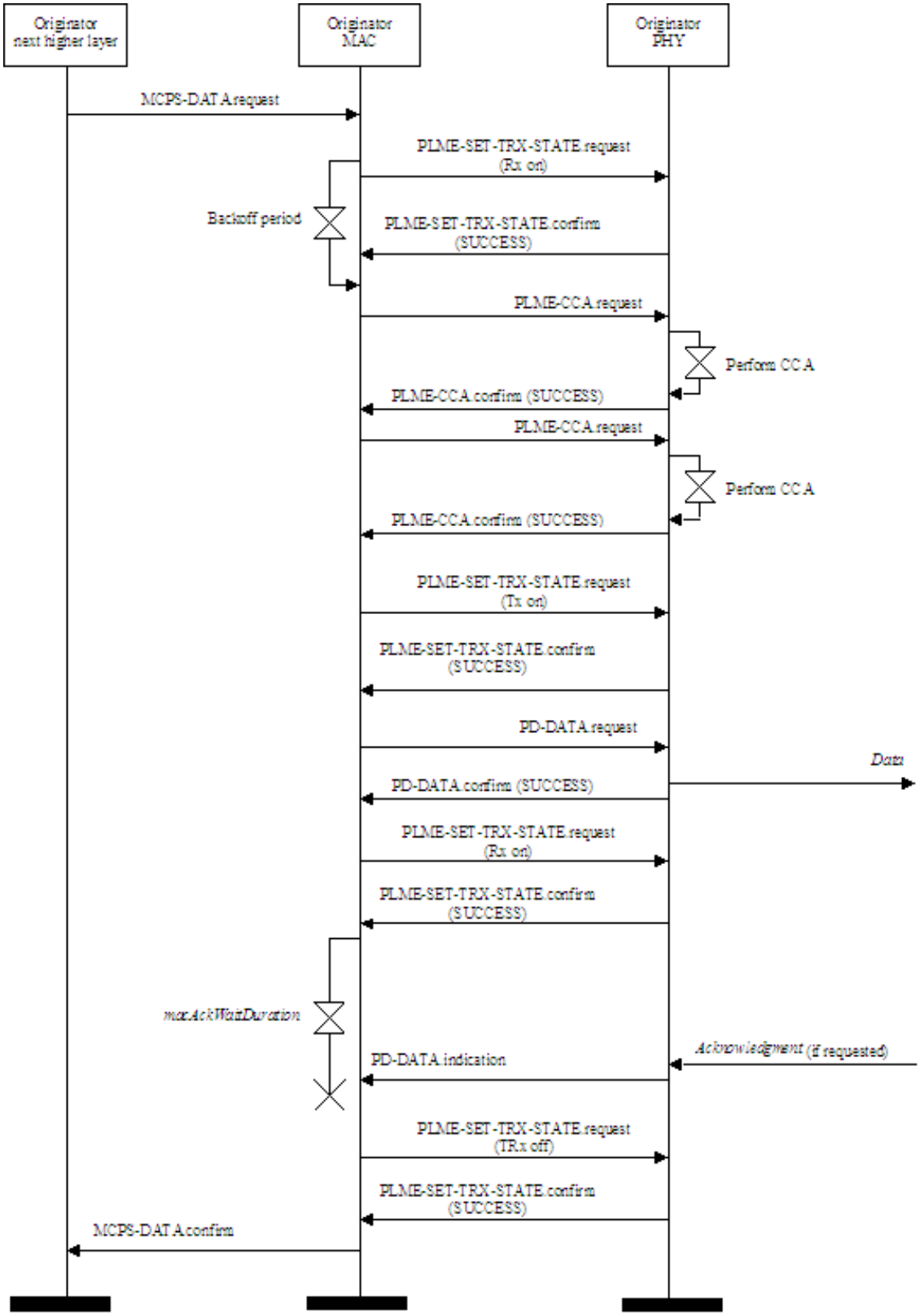


Figure 72—Data transmission message sequence chart—originator

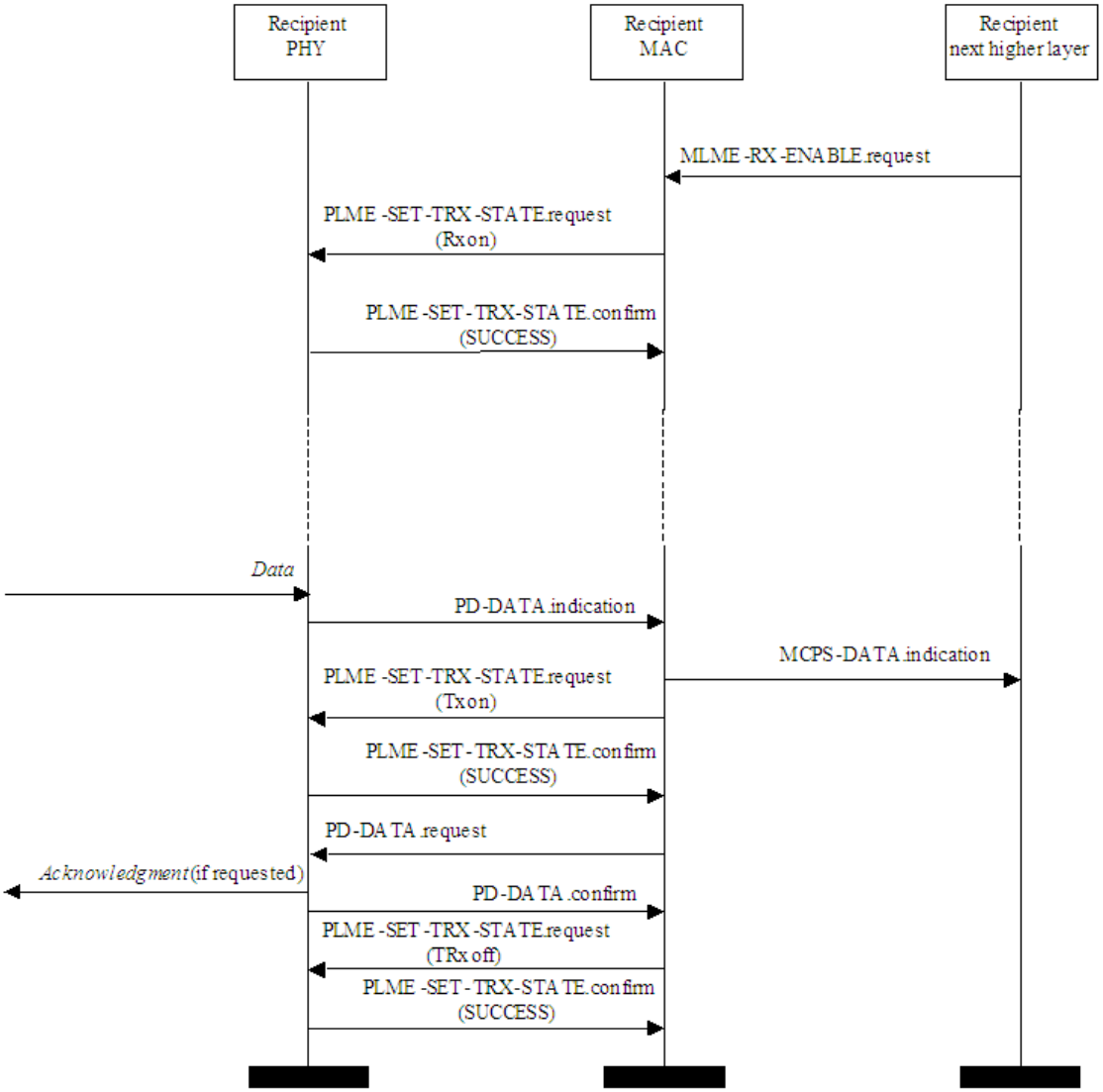


Figure 73—Data transmission message sequence chart—recipient

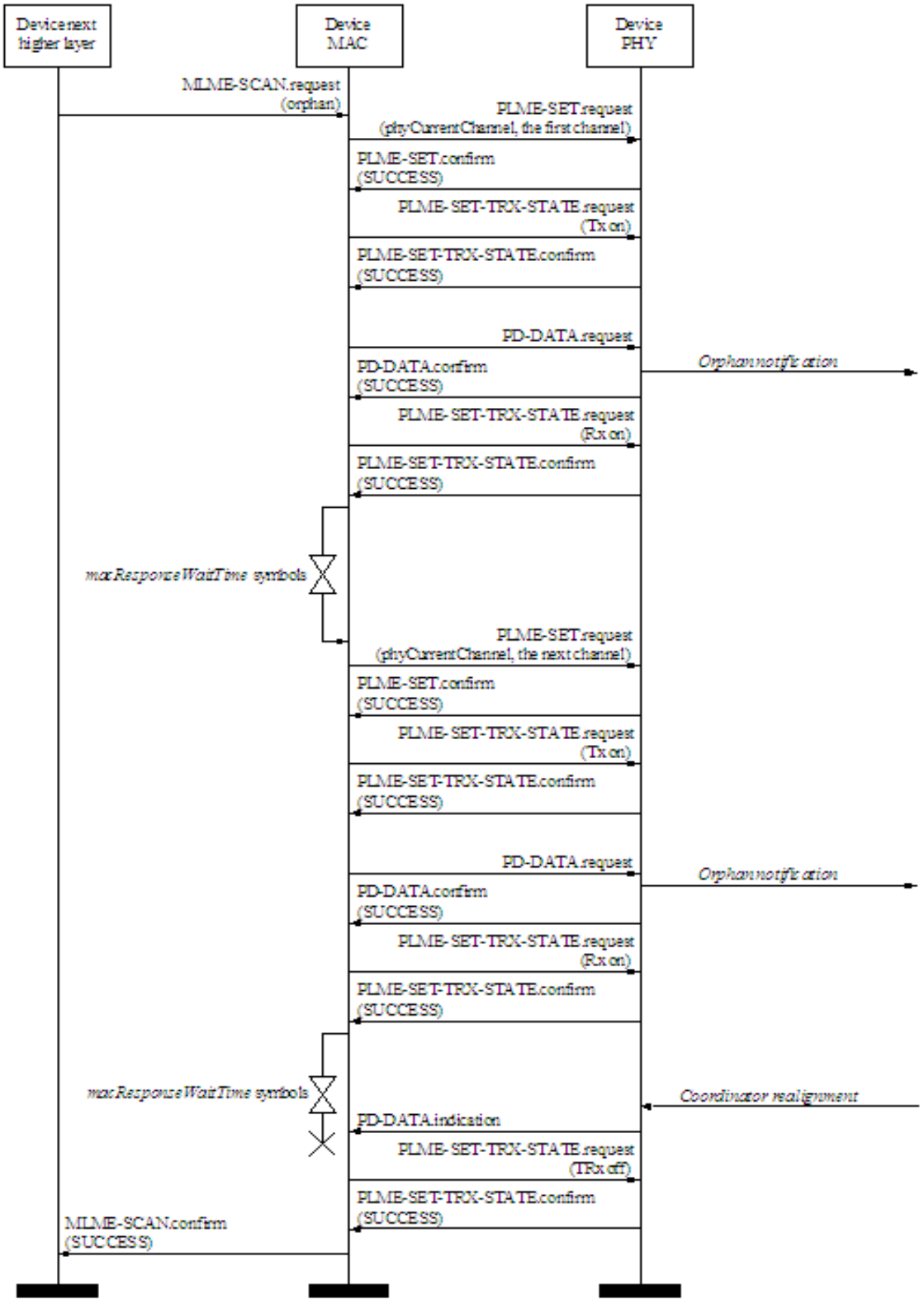


Figure 74—Orphaned device realignment message sequence chart

Annex A

(normative)

Service-specific convergence sublayer (SSCS)

A.1 IEEE 802.2 convergence sublayer

The IEEE 802.2 convergence sublayer exists above the IEEE 802.15.4 MCPS. This sublayer provides an interface between an instance of an IEEE 802.2 LLC sublayer and the IEEE 802.15.4 MCPS.

A.1.1 MA-UNITDATA.request

The MA-UNITDATA.request primitive requests the transfer of a LLC protocol data unit (LPDU) (i.e., MSDU) from a local IEEE 802.2 Type 1 LLC sublayer entity to a single peer IEEE 802.2 Type 1 LLC sublayer entity or multiple peer IEEE 802.2 Type 1 LLC sublayer entities in the case of a group address.

A.1.1.1 Semantics of the service primitive

The semantics of the MA-UNITDATA.request primitive is as follows:

```
MA-UNITDATA.request     (
                        SrcAddr,
                        DstAddr,
                        RoutingInformation,
                        data,
                        priority,
                        ServiceClass
                        )
```

Table A.1 specifies the parameters for the MA-UNITDATA.request primitive.

Table A.1—MA-UNITDATA.request parameters

Name	Type	Valid range	Description
SrcAddr	IEEE address	Any valid IEEE address	The individual IEEE address of the entity from which the MSDU is being transferred.
DstAddr	IEEE address	Any valid IEEE address	The individual IEEE address of the entity to which the MSDU is being transferred.
RoutingInformation	—	null	This parameter is not used by the MAC sublayer and shall be specified as a null value.
data	Set of octets	—	The set of octets forming the MSDU to be transmitted by the MAC sublayer entity.
priority	—	null	This parameter is not used by the MAC sublayer and shall be specified as a null value.
ServiceClass	—	null	This parameter is not used by the MAC sublayer and shall be specified as a null value.

A.1.1.2 Appropriate usage

The MA-UNITDATA.request primitive is generated by a local IEEE 802.2 Type 1 LLC sublayer entity when an LPDU (MSDU) is to be transferred to a peer IEEE 802.2 Type 1 LLC sublayer entity or entities.

A.1.1.3 Effect on receipt

On receipt of the MA-UNITDATA.request primitive, the MAC sublayer entity shall begin the transmission of the supplied MSDU.

The MAC sublayer first builds an MPDU to transmit from the supplied arguments. The MPDU shall be transmitted using the CSMA-CA algorithm in the contention period of the frame and without requesting a handshake.

If the CSMA-CA algorithm indicates a busy channel, the MAC sublayer shall issue the MA-UNITDATA-STATUS.indication primitive with a status of CHANNEL_ACCESS_FAILURE. If the MPDU was successfully transmitted, the MAC sublayer shall issue the MA-UNITDATA-STATUS.indication primitive with a status of SUCCESS.

A.1.2 MA-UNITDATA.indication

The MA-UNITDATA.indication primitive indicates the transfer of an LPDU (i.e., MSDU) from the MAC sublayer to the local IEEE 802.2 Type 1 LLC sublayer entity.

A.1.2.1 Semantics of the service primitive

The semantics of the MA-UNITDATA.indication primitive is as follows:

```
MA-UNITDATA.indication          (
                                SrcAddr,
                                DstAddr,
                                RoutingInformation,
                                data,
                                ReceptionStatus,
                                priority,
                                ServiceClass
                                )
```

Table A.2 specifies the parameters for the MA-UNITDATA.indication primitive.

A.1.2.2 When generated

On receipt of a data packet at the local MAC sublayer entity, the FCS field is checked. If it is valid, the MAC sublayer shall issue the MA-UNITDATA.indication primitive to the IEEE 802.2 Type 1 LLC sublayer entity, indicating the arrival of a MSDU. If the FCS is not valid, the packet shall be discarded, and the IEEE 802.2 Type 1 LLC sublayer entity shall not be informed.

A.1.2.3 Appropriate usage

The appropriate usage of the MA-UNITDATA.indication primitive by the IEEE 802.2 Type 1 LLC sublayer entity is not specified in this standard.

Table A.2—MA-UNITDATA.indication parameters

Name	Type	Valid range	Description
SrcAddr	IEEE address	Any valid IEEE address	The individual IEEE address of the entity from which the MSDU has been received
DstAddr	IEEE address	Any valid IEEE address	The individual IEEE address of the entity to which the MSDU is being transferred.
RoutingInformation	—	null	This parameter is not used by the MAC sublayer and shall be specified as a null value.
data	Set of octets	—	The set of octets forming the MSDU received by the MAC sublayer entity.
ReceptionStatus	—	null	This parameter is not used by the MAC sublayer and shall be specified as a null value.
priority	—	null	This parameter is not used by the MAC sublayer and shall be specified as a null value.
ServiceClass	—	null	This parameter is not used by the MAC sublayer and shall be specified as a null value.

A.1.3 MA-UNITDATA-STATUS.indication

The MA-UNITDATA-STATUS.indication primitive reports the results of a request to transfer a LPDU (MSDU) from a local IEEE 802.2 Type 1 LLC sublayer entity to a single peer IEEE 802.2 Type 1 LLC sublayer entity or to multiple peer IEEE 802.2 Type 1 LLC sublayer entities.

A.1.3.1 Semantics of the service primitive

The semantics of the MA-UNITDATA-STATUS.indication primitive is as follows:

```
MA-UNITDATA-STATUS.indication   (
                                SrcAddr,
                                DstAddr,
                                status,
                                ProvPriority,
                                ProvServiceClass
                                )
```

Table A.3 specifies the parameters for the MA-UNITDATA-STATUS.indication primitive.

A.1.3.2 When generated

The MA-UNITDATA-STATUS.indication primitive is generated by the MAC sublayer entity in response to an MA-UNITDATA.request primitive issued by the IEEE 802.2 Type 1 LLC sublayer.

A.1.3.3 Appropriate usage

The receipt of the MA-UNITDATA-STATUS.indication primitive by the IEEE 802.2 Type 1 LLC sublayer entity signals the completion of the current data transmission.

Table A.3—MA-UNITDATA-STATUS.indication parameters

Name	Type	Valid Range	Description
SrcAddr	IEEE address	Any valid IEEE address	The individual IEEE address of the entity from which the MSDU has been transferred.
DstAddr	IEEE address	Any valid IEEE address	The individual IEEE address of the entity to which the MSDU has been transferred.
status	Enumeration	SUCCESS, TRANSMISSION_PENDING, NO_BEACON, or CHANNEL_ACCESS_FAILURE	The status of the last MSDU transmission.
ProvPriority	—	null	This parameter is not used by the MAC sublayer and shall be specified as a null value.
ProvServiceClass	—	null	This parameter is not used by the MAC sublayer and shall be specified as a null value.