2.4 GHz 802.15.4 MAC Application Interface Programmer's Guide

1. Introduction

This document describes the Silicon Laboratories 802.15.4 MAC Application Programming Interface (API). It serves as a reference for the implementation device on specific details of the API.

The purpose of this document is to link command primitives of the Silicon Laboratories MAC software implementation to the primitives defined in the IEEE 802.15.4 specification. The IEEE specification contains a substantive description of each primitive, and its parameters and should be read for a thorough understanding of the MAC functionality.

2. Application Procedures

The 802.15.4 MAC API provides the command primitives necessary to manage a link and transfer data. The next higher layer (the user application in this case, referred to as "the application") must manage the MAC through use of the provided primitives. This section provides a high-level description of the procedures.

The 802.15.4 specification defines links between devices within the same Personal Operating Space (POS). Networking is the responsibility of higher layers and is not addressed in this document. Refer to the Silicon Laboratories ZigBee Development Kit series for a complete 802.15.4 plus network solution.

An 802.15.4 network is started by first establishing a Personal Area Network (PAN) Coordinator. Other devices may be added to the network once the coordinator is running.

2.1. System Initialization

Each device should be initialized to a known state on power-up by the following steps:

Disable Global Interrupts:

```
DISABLE_GLOBAL_INT();
```

System Hardware Init:

```
SystemInit();
```

Initialize PHY PIB:

```
PHY_PIB_init();
```

Initialize MAC PIB:

```
SetDefult_macpib();
```

Initialize MAC Internal Variables:

```
init_macpremeter();
```

Initialize Transceiver:

```
CC2420Init();
```

Enable Global Interrupt:

```
ENABLE_GLOBAL_INT();
```

2.2. Starting a Node as a PAN Coordinator

An 802.15.4 network is made up of links between coordinators and lower-level devices. In terms of a parent-child relationship, the coordinators are parents, and devices are children. Each 802.15.4 topology has a top-level controller, or PAN Coordinator. Subsequent links are between lower-level controllers and devices.

The PAN coordinator startup is similar to that of other devices, with the exception that it must perform a few extra steps to establish the network. The first of these extra steps is to identify a suitable RF channel for the network using the MAC Scan primitives. Scan types include energy-detect, passive, and active scans. An energy-detect scan simply scans for RF power on all channels and is useful for detecting 802.15.4 systems, wireless LAN, Bluetooth, and ISM interferers. Passive and active scans search for 802.15.4 beacon frames from other 15.4 networks. A passive scan simply listens for a beacon, whereas an active scan actively requests a beacon on each channel. Non-beacon networks must always use an active scan. The application should interpret these results to initiate a network on the channel with the fewest number of interferers.

The application selects a PAN ID to identify the new network. The PAN ID should be unique within the POS.

The steps to start a PAN coordinator are as follows:

1. Reset the device: MLME-RESET.request.
2. Scan channels; select the best: MLME-SCAN.request.
3. Set the PAN coordinator's short MAC address: MLME-SET.request; set PIB macCoordShort-Address.
4. Start the PAN coordinator: MLME-START.request, panCoordinator = TRUE.
5. Allow other devices to associate: MLME-SET.request MacAssociationPermit = TRUE.
6. Transfer data once devices are associated.

2.3. Adding Nodes to the Network

A device may join the network once a coordinator is established and accepting association requests. This coordinator may be either the PAN coordinator or another lower-level coordinator.

The joining device first performs a scan to find networks within its Personal Operating Space (POS). For beacon-based networks, the device can listen on each channel for the periodic beacon using a passive scan. For non-beacon-based networks, the device must request a beacon on each channel using an active scan.

Once the scan is complete, the MLME-SCAN confirmation will include a table of PANs within the POS. The application layer can then select one of the PANs and request association.

The steps to start a device are:

1. Reset the device: MLME-RESET.request.
2. Scan to locate desired PAN: MLME-SCAN.request.
3. Select from available PANs: MLME-SCAN.confirm.
4. Request association with selected PAN: MLME-ASSOCIATE.

   Optional: if starting as a coordinator

   5.Configure, start Beacon: MLME-START.request, panCoordinator = FALSE.

   6.Allow other devices to associate: MLME-SET set PIB MacAssociationPermit = TRUE.
7. Transfer data.

2.4. Transferring Data Across the Network

Data may be exchanged between devices in a POS once they are associated.

Note: 802.15.4 only provides a transfer mechanism between devices. If needed, networking and routing functionality should be implemented in a layer above the 802.15.4 MAC.

2.4.1. Transmitting to Coordinator

This section describes how a device transmits data to a coordinator.

2.4.1.1. Beacon Network

The device is synchronized to the coordinator beacon after association.

The device communicates to its coordinator using the MCPS-DATA request in beacon mode. The MCPS-DATA confirm indicates the status of a transmission attempt. All transmissions occur within the CAP or GTS period.

2.4.1.2. Non-Beacon Network

The device communicates to its coordinator using the MCPS-DATA request in non-beacon mode. The MCPS-DATA confirm indicates the status of a transmission attempt.

In a non-beacon network, the transmission request is processed immediately via CSMA-CA.

2.4.2. Receiving from Coordinator

This section describes how a device receives data from a coordinator.

2.4.2.1. Beacon Network

In a beacon-based network, the device is normally synchronized to its coordinator's beacon. Each beacon received will generate a MLME-BEACON-NOTIFY indication to the application. The application should check the `AddrList` parameter of the indication for its own address to determine if a message is pending.

If a message is pending, the device should issue a MLME-POLL request. The device will then send a data request frame to the coordinator. The coordinator will then transmit the data which will be conveyed to the application via an MCPS-DATA indication.

2.4.2.2. Non-Beacon Network

In a non-beacon-based network, the device must proactively check with its coordinator for pending data by issuing a MLME-POLL request. The coordinator will then transmit the data which will be periodically conveyed to the application via an MCPS-DATA indication. The coordinator will return a zero-length data response (`msduLength=0`) if no data are pending.

Note: In a non-beacon network, there is no scheduling mechanism to dictate when communication should occur. As such, the coordinator's receiver should be on continuously, or the application should schedule the communication.

2.5. Dissolving the Network

A device may disassociate from its coordinator. A coordinator may also proactively initiate disassociation of a device.

The application initiating the disassociation issues an MLME-DISASSOCIATE request. Arguments of this request include the long address of the disassociating device and a reason. These arguments determine whether a device is removing itself or an associated device. If removing itself, the arguments include its own long address and a reason of "The device wishes to leave the PAN". If removing an associated device, the arguments include the long address of the device to disassociate and a reason for "The coordinator wishes the device to leave the PAN." The request will be followed by a MLME-DISASSOCIATE confirm indicating the status of the disassociation.

A disassociate request at one device will create a MLME-DISASSOCIATE indication in the other device to notify the application of the disassociation.

3. Application Programming Interface Overview

This section provides notes specific to the implementation of the Silicon Laboratories MAC API.

Communication between the application and MAC layers is through two shared buffers, one for each direction. This method was selected to minimize the RAM required for primitive arguments. All primitives in the 802.15.4 specification may be accessed through this method.

All addresses are in little endian format.

Structures for each primitive are described in detail in Sections 4 and 5.

3.1. Application to MAC Communication

The `n_m_msg_t` buffer conveys messages from the application to the MAC layer. This buffer is one message deep. Each primitive has a corresponding message data structure. The primitive message data structure defines all of the parameter data types and the order in which they will be stored in the buffer. The network to MAC message data type, `msg`, is defined as a union of all the primitive message data structures. This conserves RAM by using a common block of memory to store all primitive parameters. The network to MAC message buffer `n_m_msg_t` is a structure consisting of the primitive type primitive and the message data union `msg`.

Parameters are first loaded into the `n_m_msg_t.msg` buffer. Once configured, `n_m_msg_t.primtype` is set with the appropriate enumerated primitive and `NWK_Buffer_Valid` (MSB of primitive buffer) set indicating that the message is complete and ready to be processed.

Type declarations for the buffer, message structure, and enumerated primitive types are in `nwk2mac.h`, shown in " Appendix A—Application to MAC Shared Data Structures" on page 36 .

3.2. MAC to Application Communication

The `m2n_msg` buffer relays messages from the MAC up to the application. This buffer may contain up to `TransactionBuffLenth` messages.

The MAC to network message is also defined as a union of primitive message data structures. The `ProcessMAC2NWK()` function processes the buffer and handles the resulting confirmation where applicable. This function should be called regularly from the main loop.

The MAC layer will set `NWK_Buffer_Valid = 1` in a buffer to signify a message pending for the application. (`NWK_Buffer_Valid` is simply the most significant bit of `m2n_msg.primtype`.) Once the message has been read, the application should set `m2n_msg.primtype = emptyBuffer`. Note that `ProcessMAC2NWK()` services this buffer for a confirmation after a primitive has been called.

Type declarations for the buffer, message structure, and enumerated primitive types are in mac2nwk.h, shown in " Appendix B—MAC to Application Shared Data Structures" on page 39 .

4. MAC Common Part Sublayer (MCPS-) Primitives

The MAC Common Part Sublayer includes data transfer services.

Name	Request	Confirm	Indication
MCPS-DATA	See “4.1.1. Request”	See “4.1.2. Confirm”	See “4.1.3. Indication”
MCPS-PURGE	See “4.1.1. Request”	See “4.1.2. Confirm”	—

4.1. MCPS-DATA

4.1.1. Request

The Data-Request structure includes the actual data in the buffer itself. The payload should be copied to the buffer before setting the NWK_Buffer_Valid bit.

```
// Type: gMcpsDataReq_c
typedef struct mcpsDataReq_tag
{
                  unsigned char dstAddr[8]; //Address as defined by dstAddrMode
                  unsigned short dstPanId;
                  unsigned char dstAddrMode;
                  unsigned char srcAddr[8]; //Address as defined by srcAddrMode
                  unsigned short srcPanId;
                  unsigned char srcAddrMode;
                  unsigned char msduLength; // 0-102
                  unsigned char msduHandle;
                  unsigned char txOptions;
                  unsigned char SecurityLevel;
                  unsigned char msdu[aMaxMACFrameSize]; // Data will start at this
byte
}mcpsDataReq_t;
```

Each entry in the MAC-to-network buffer is a structure consisting of the primitive type, `primitive`, and the message union.

Name	Type	Valid range	Description
dstAddr	unsigned char	As specified by the DstAddrMode parameter	The individual device address of the entity to which the MSDU is being transferred.
dstPanId	unsigned short	0x0000–0xFFFF	The 16-bit PAN identifier of the entity to which the MSDU is being transferred.
dstAddrMode	unsigned char	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). 0x01 = reserved. 0x02 = 16-bit short address. 0x03 = 64-bit extended address.
srcAddr	unsigned char	As specified by the SrcAddrMode parameter	The individual device address of the entity from which the MSDU is being transferred.
srcPanId	unsigned short	0x000–0xFFFF	The 16-bit PAN identifier of the entity from which the MSDU is being transferred.
srcAddrMode	unsigned char	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). 0x01 = Reserved. 0x02 = 16-bit short address. 0x03 = 64-bit extended address.
msduLength	unsigned char	aMaxMACFrameSize	The number of octets contained in the MSDU to be transmitted by the MAC sublayer entity.
msduHandle	unsigned char	0x00–0xFF	The handle associated with the MSDU to be transmitted by the MAC sublayer entity.
txOptions	bitmap	0000 xxxx (where x can be 0 or 1)	The transmission options for this MSDU. These are a bitwise OR of one or more of the following: 0x01 = acknowledged transmission. 0x02 = GTS transmission. 0x04 = indirect transmission. 0x08 = security-enabled transmission.
msdu	set of octets		The set of octets forming the MSDU to be transmitted by the MAC sublayer entity.

4.1.2. Confirm

```
// Type: gMcpsDataCnf_c,
typedef struct mcpsDataCnf_tag
{
                  unsigned char msduHandle;
                  unsigned char status;
}mcpsDataCnf_t;
```

Name	Type	Valid range	Description
msduHandle	unsigned char	0x00–0xFF	The handle associated with the MSDU being confirmed.
status	unsigned char	SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, CHANNEL_ACCESS_FAILURE, INVALID_GTS, NO_ACK, UNAVAILABLE_KEY, FRAME_TOO_LONG, FAILED_SECURITY_CHECK, or INVALID_PARAMETER	The status of the last MSDU transmission.

4.1.3. Indication

The data should be copied from the buffer before clearing the `NWK Buffer_valid` flag.

```
// Type: gMcpsDataInd_c,
typedef struct mcpsDataInd_tag
{
                  unsigned char srcAddrMode;
                  unsigned char srcPanId[2];
                  unsigned char srcAddr[8]; //Address as defined by srcAddrMode
                  unsigned char dstAddrMode;
                  unsigned char dstPanId[2];
                  unsigned char dstAddr[8]; //Address as defined by dstAddrMode
                  unsigned char mpduLinkQuality;
                  unsigned char securityUse;
                  unsigned char aclEntry;
                  unsigned char msduLength; // 0-102
                  unsigned char msdu[aMaxMACFrameSize]; // Data will start at this
byte
}mcpsDataInd_t;
```

Name	Type	Valid range	Description
srcAddrMode	unsigned char	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.
srcPANId	unsigned char	0x0000–0xFFFF	The 16-bit PAN identifier of the entity from which the MSDU was received.
srcAddr	unsigned char	As specified by the *SrcAddrMode* parameter	The individual device address of the entity from which the MSDU was received.
dstAddrMode	unsigned char	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). 0 x 01 = Reserved. 0x02 = 16-bit short device address. 0x03 = 64-bit extended device address.
dstPANId	unsigned char	0x0000–00xFFFF	The 16-bit PAN identifier of the entity to which the MSDU is being transferred.
dstAddr	unsigned char	As specified by the *DstAddrMode* parameter	The individual device address of the entity to which the MSDU is being transferred.
mpduLinkQuality	unsigned char	0x00–0xFF	LQ value measured during reception of the MPDU. Lower values represent lower LQ.
securityUse	unsigned char	TRUE or FALSE	An indication of whether the received data frame is using security. This value is set to TRUE if the security enable subfield was set to 1 or FALSE if the security enabled subfield was set to 0.
aclEntry	unsigned char	0x00–0x08	The *macSecurityMode* parameter value from the ACL entry associated with the sender of the data frame. This value is set to 0x08 if the sender of the data frame was not found in the ACL.
msduLength	unsigned char	*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.

4.2. MCPS-PURGE

4.2.1. Request

```
// Type: gMcpsPurgeReq_c,
typedef struct mcpsPurgeReq_tag
{
                  unsigned char msduHandle;
}mcpsPurgeReq_t;
```

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

4.2.2. Confirm

```
// Type: gMcpsPurgeCnf_c
typedef struct mcpsPurgeCnf_tag
{
                  unsigned char msduHandle;
                  unsigned char status;
}mcpsPurgeCnf_t;
```

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

5. MAC Sublayer Management Entity (MLME-) Primitives

Name	Request	Indication	Response	Confirm
MLME-ASSOCIATE	See “5.1.1. Request”	See “5.1.2. Indication”	See “5.1.3. Response”	See “5.1.4. Confirm”
MLME-DISASSOCIATE	See “5.2.1. Request”	See “5.2.2. Indication”	—	See “5.2.3. Confirm”
MLME-BEACON-NOTIFY	—	See “5.3.1. Indication”	—	—
MLME-GET	See “5.4.1. Request”	—	—	See “5.4.2. Confirm”
MLME-GTS	See “5.5.1. Request”	See “5.5.2. Indication”	—	See “5.5.3. Confirm”
MLME-ORPHAN	—	See “5.6.1. Indication”	See “5.6.2. Response”	—
MLME-RESET	See “5.7.1. Request”	—	—	See “5.7.2. Confirm”
MLME-RX-ENABLE	See “5.8.1. Request”	—	—	See “5.8.2. Confirm”
MLME-SCAN	See “5.9.1. Request”	—	—	See “5.9.2. Confirm”
MLME-COMM-STATUS	—	See “5.10.1. Indication”	—	—
MLME-SET	See “5.11.1. Request”	—	—	See “5.11.2. Confirm”
MLME-START	See “5.12.1. Request”	—	—	See “5.12.2. Confirm”
MLME-SYNC	See “5.13.1. Request”	—	—	—
MLME-SYNC-LOSS	—	See “5.14.1. Indication”	—	—
MLME-POLL	See “5.15.1. Request”	—	—	See “5.15.2. Confirm”

5.1. MLME-ASSOCIATE

5.1.1. Request

Before sending the Associate-Request primitive, the 802.15.4 specification states that a Reset-Request must have been sent and an Active or Passive Scan performed.

```
// Type: gMlmeAssociateReq_c
typedef struct mlmeAssociateReq_tag
{
                  unsigned char coordAddress[8];
                  unsigned short coordPanId;
                  unsigned char coordAddrMode;
                  unsigned char logicalChannel;
                  unsigned char SecurityLevel;
                  unsigned char capabilityInfo;
                  unsigned char ChannelPage;
                  unsigned char KeyIdAddrMode;
}mlmeAssociateReq_t;
```

Name	Type	Valid range	Description
coordAddress	unsigned char	As specified by the *CoordAddrMode* parameter.	The address of the coordinator with which to associate.
coordPANId	unsigned short	0x0000–0xFFFF	The identifier of the PAN with which to associate.
coordAddrMode	unsigned char	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.
logicalChannel	unsigned char	Selected from the available channels supported by the PHY	The logical channel on which to attempt association.
SecurityLevel	unsigned char		
capabilityInfo	Bitmap	See 802.15.4 specification	Specifies the operational capabilities of the associating device.
ChannelPage	unsigned char		
KeyIdAddrMode	unsigned char		

5.1.2. Indication

```
// Type: gNwkAssociateInd_c
      typedef struct nwkAssociateInd_tag
      {
      unsigned char deviceAddress[8];
      unsigned char capabilityInfo;
      unsigned char securityUse;
      unsigned char AclEntry;
      }nwkAssociateInd_t;
```

Name	Type	Valid range	Description
deviceAddress	unsigned char	An extended 64-bit IEEE address.	The address of the device requesting association.
capabilityInfo	Bitmap		The operational capabilities of the device requesting association.
securityUse	unsigned char	TRUE or FALSE	An indication of whether the received MAC command frame is using security. This value is set to TRUE if the security enable subfield was set to 1 or FALSE if the security enabled subfield was set to 0.
AclEntry	unsigned char	0x00– 0x08	The *macSecurityMode* parameter value from the ACL entry associated with the sender of the data frame. This value is set to 0x08 if the sender of the data frame was not found in the ACL.

5.1.3. Response

```
// Type: gMlmeAssociateRes_c
typedef struct mlmeAssociateRes_tag
{
                  unsigned char deviceAddress[8];
                  unsigned char assocShortAddress[2];
                  unsigned char securityEnable;
                  unsigned char status;
}mlmeAssociateRes_t;
```

Name	Type	Valid range	Description
deviceAddress	unsigned char	An extended 64-bit IEEE address	The address of the device requesting association.
assocShortAddress	unsigned char	0x0000–0xFFFF	The short device address allocated by the coordinator on successful association. This parameter is set to 0xffff if the association was unsuccessful.
securityEnable	unsigned char	TRUE or FALSE	TRUE if security is enabled for this transfer or FALSE otherwise.
status	unsigned char	See “ Appendix B—MAC to Application Shared Data Structures”	The status of the association attempt.

5.1.4. Confirm

```
// Type: gNwkAssociateCnf_c
typedef struct nwkAssociateCnf_tag
{
                  unsigned char assocShortAddress[2];//WORD assocShortAddress;
                  unsigned char status;
}nwkAssociateCnf_t;
```

Name	Type	Valid range	Description
assocShortAddress	unsigned char	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	unsigned char	The value of the status field of the associate response command SUCCESS, CHANNEL_ACCESS_FAILURE, NO_ACK, NO_DATA, UNAVAILABLE_KEY, FAILED_SECURITY_CHECK, or INVALID_PARAMETER.	The status of the association attempt.

5.2. MLME-DISASSOCIATE

5.2.1. Request

```
// Type: gMlmeDisassociateReq_c
typedef struct mlmeDisassociateReq_tag
{
                  unsigned char deviceAddress[8];
                  unsigned char securityEnable;
                  unsigned char disassociateReason;
}mlmeDisassociateReq_t;
```

Name	Type	Valid range	Description
deviceAddress	unsigned char	An extended 64-bit IEEE address.	The address of the device to which to send the disassociation notification command.
securityEnable	unsigned char	TRUE or FALSE	TRUE if security is enabled for this transfer or FALSE otherwise.
disassociateReason	unsigned char	0x00–0xFF	The reason for the disassociation (see below).

Table 1. Disassociation Reasons

Disassociate Reason	Description
0x00	Reserved.
0x01	The coordinator wishes the device to leave the PAN.
0x02	The device wishes to leave the PAN.
0x03–0x7F	Reserved.
0x80–0xFF	Reserved for MAC primitive enumeration values.

5.2.2. Indication

```
// Type: gNwkDisassociateInd_c
typedef struct nwkDisassociateInd_tag
{
   unsigned char deviceAddress[8];
unsigned char disassociateReason;
                  unsigned char securityUse;
   unsigned char aclEntry;
}nwkDisassociateInd_t;
```

Name	Type	Valid range	Description
deviceAddress	unsigned char	An extended 64-bit IEEE address	The address of the device requesting disassociation.
disassociateReason	unsigned char	0x00–0xFF	The reason for the disassociation.
securityUse	unsigned char	TRUE or FALSE	An indication of whether the received MAC command frame is using security. This value is set to TRUE if the security enable subfield was set to 1 or FALSE if the security enabled subfield was set to 0.
aclEntry	unsigned char	0x00–0x08	The *macSecurityMode* parameter value from the ACL entry associated with the sender of the data frame. This value is set to 0x08 if the sender of the data frame was not found in the ACL.

5.2.3. Confirm

```
// Type: gNwkDisassociateCnf_c
typedef struct nwkDisassociateCnf_tag
{
                  unsigned char status;
}nwkDisassociateCnf_t;
```

Name	Type	Valid range	Description
Status	Enumeration	SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, NO_ACK, CHANNEL_ACCESS_FAILURE, UNAVAILABLE_KEY, FAILED_SECURITY_CHECK, or INVALID_PARAMETER	The status of the disassociation attempt.

5.3. MLME-BEACON-NOTIFY

5.3.1. Indication

The Beacon-Notify-Indication message is one of the special messages that contains pointers. `pAddrList` points to the address list, which is formatted according to IEEE 802.15.4 standard, Sec 7.2.2.1.6/7. `pPanDescriptor` points to the pan descriptor of the indication message. `psdu` is the beacon payload buffer. The `pBufferRoot` pointer contains the data fields pointed to by the other pointers.

The `pBufferRoot` must be freed before freeing the indication message. As shown in this example, `MSG_Free(pBeaconInd->pBufferRoot); MSG_Free(pBeaconInd);` otherwise, the MAC memory pools will be exhausted after just a few beacons.

```
typedef struct nwkBeaconNotifyInd_tag
{
                  unsigned char bsn;
                  PAN_DESCRIPTOR pPanDescriptor;
                  unsigned char pendAddrSpec;
                  unsigned char *pAddrList;
                  unsigned char psduLength;
                  unsigned char *psdu;
                  unsigned char *pBufferRoot;
}nwkBeaconNotifyInd_t;
```

Name	Type	Valid range	Description
bsn	unsigned char	0x00–0xFF	The beacon sequence number.
pPANDescriptor	PAN_DESCRIPTOR	See below.	The PANDescriptor for the received beacon.
pendAddrSpec	unsigned char	See 802.15.4 specification	The beacon pending address specification.
pAddrList	unsigned char	—	The list of addresses of the devices for which the beacon source has data.
psduLength	unsigned char	0 – *aMaxBeaconPayloadLength*	The number of octets contained in the beacon payload of the beacon frame received by the MAC sublayer.
psdu	Set of octets	—	The set of octets comprising the beacon payload to be transferred from the MAC sublayer entity to the next higher layer.
pBufferRoot			

Table 2. PAN Descriptor

Name	Type	Valid range	Description
byCoordAddrMode	unsigned char	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.
dwCoordPANId	unsigned short	0x0000–0xFFFF	The PAN identifier of the coordinator as specified in the received beacon frame.
CoordAddress	unsigned char	As specified by the CoordAddrMode parameter	The address of the coordinator as specified in the received beacon frame.
byLogicalChannel	unsigned char	Selected from the available logical channels supported by the PHY	The current logical channel occupied by the network.
dwSuperframeSpec	unsigned short	See 802.15.4 specification.	The superframe specification as specified in the received beacon frame.
byGTSPermit	unsigned char	TRUE or FALSE	TRUE if the beacon is from a PAN coordinator that is accepting GTS requests.
byLinkQuality	unsigned char	0x00–0xFF	The LQ at which the network beacon was received. Lower values represent lower LQ.
dgTimeStamp	unsigned long	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. The precision of this value is a minimum of 20 bits, with the lowest 4 bits being the least significant.
bySecurityUse	unsigned char	TRUE or FALSE	An indication of whether the received beacon frame is using security. This value is set to TRUE if the security enable subfield was set to 1 or FALSE if the security enabled subfield was set to 0.
byACLEntry	unsigned char	0x00–0x08	The *macSecurityMode* parameter value from the ACL entry associated with the sender of the data frame. This value is set to 0x08 if the sender of the data frame was not found in the ACL.
bySecurityFailure	unsigned char	TRUE or FALSE	TRUE if there was an error in the security processing of the frame or FALSE otherwise.

5.4. MLME-GET

5.4.1. Request

The Get-Request message contains a pointer to a buffer where data from the MAC PIB will be copied. The pointer must be supplied by the NWK or APP. Attributes with a size of more than one byte are little-endian and are given as byte arrays.

```
// Type: gMlmeGetReq_c,
typedef struct mlmeGetReq_tag
{
                  unsigned char pibAttribute;
                  unsigned char *pibAttributeValue; // Pointer supplied by NWK
}mlmeGetReq_t;
```

Name	Type	Valid range	Description
PIBAttribute	unsigned char	See Tables 3 and 4.	The identifier of the PIB attribute to read.
pibAttributeValue	unsigned char	See Tables 3 and 4.	Attribute Value.

Table 3. MAC PIB Attributes

Attribute	Identifier	Type	Range	Description	Default
macAckWaitDuration	0x40	Integer	54 or 120	The maximum number of symbols to wait for an acknowledgment frame to arrive following a transmitted data frame. This value is dependent on the currently-selected logical channel. For 0 *phyCurrentChannel* 10, this value is equal to 120. For 11 *phyCurrentChannel* 26, this value is equal to 54.	54
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.	TRUE

Table 3. MAC PIB Attributes (Continued)

Attribute	Identifier	Type	Range	Description	Default
macBattLifeExt	0x43	Boolean	TRUE or FALSE	Indication of whether battery life extension, by reduction of coordinator receiver operation time during the CAP, is enabled. A value of TRUE indicates that it is enabled.	FALSE
macBattLifeExtPeriods	0x44	Integer	6 or 8	The number of backoff periods during which the receiver is enabled following a beacon in battery life extension mode. This value is dependent on the currently-selected logical channel. For 0 *phyCurrentChannel* 10, this value is equal to 8. For 11 *phyCurrentChannel* 26, this value is equal to 6.	6
macBeaconPayload_	0x45	Set of octets	—	The contents of the beacon payload.	NULL
macBeaconPayload-Length_	0x46	Integer	0 – *aMax-Beacon-Payload-Length*	The length, in octets, of the beacon payload.	0
macBeaconOrder_	0x47	Integer	0–15	Specification of how often the coordinator transmits a beacon. The *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 will not transmit a beacon.	15
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. The precision of this value shall be a minimum of 20 bits, with the lowest four bits being the least significant.	0x000000

Table 3. MAC PIB Attributes (Continued)

Attribute	Identifier	Type	Range	Description	Default
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 with which the device is associated.	—
macCoordShort-Address	0x4b	Integer	0x0000–0xffff	The 16-bit short address assigned to the coordinator with which the device is associated. A value of 0 x fffe indicates that the coordinator is only using its 64-bit extended address. A value of 0 x ffff 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 PAN coordinator is to accept GTS requests. FALSE otherwise.	TRUE
macMaxCSMABack-offs	0x4e	Integer	0–5	The maximum number of back-offs the CSMA-CA algorithm will attempt before declaring a channel access failure.	4
macMinBE	0x4f	Integer	0–3	The minimum value of the backoff exponent in the CSMA-CA algorithm. Note that if this value is set to 0, collision avoidance is disabled during the first iteration of the algorithm. Also note that for the slotted version of the CSMA-CA algorithm with the battery life extension enabled, the minimum value of the backoff exponent will be the lesser of 2 and the value of *macMinBE*.	3

Table 3. MAC PIB Attributes (Continued)

Attribute	Identifier	Type	Range	Description	Default
macPANId	0x50	Integer	0x0000–0xffff	The 16-bit identifier of the PAN on which the device is operating. If this value is 0 x ffff, the device is not associated.	0xffff
macPromiscuous-Mode_	0x51	Boolean	TRUE or FALSE	This indicates 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
macRxOnWhenIdle	0x52	Boolean	TRUE or FALSE	This indicates whether the MAC sublayer is to enable its receiver during idle periods.	FALSE
macShortAddress	0x53	Integer	0x0000–0xffff	The 16-bit address that the device uses to communicate in the PAN. If the device is a PAN coordinator, this value shall be chosen before a PAN 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
macSuperframe-Order_	0x54	Integer	0–15	This specifies the length of the active portion of the superframe, including the beacon frame. The *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 will not be active following the beacon.	15
*macTransaction-PersistenceTime_*	0x55	Integer	0x0000–0xffff	The maximum time (in superframe periods) that a transaction is stored by a coordinator and indicated in its beacon.	0x01f4

Table 4. MAC PIB Security Attributes

Attribute	Identifier	Type	Range	Description	Default
macACLEntry-DescriptorSet	0x70	Set of ACL descriptor values	Variable	A set of ACL entries, each containing address information, security suite information, and security material to be used to protect frames between the MAC sublayer and the specified device.	Null set
macACLEntry-DescriptorSet-Size	0x71	Integer	0x00–0 x ff	The number of entries in the ACL descriptor set.	0x00
macDefault-Security	0x72	Boolean	TRUE or FALSE	Indication of whether the device is able to transmit secure frames to or accept secure frames from devices that are not explicitly listed in the ACL. It is also used to communicate with multiple devices at once. A value of TRUE indicates that such transmissions are permitted.	FALSE
macDefaultSe-curityMaterial-Length	0x73	Integer	0x00–0x1a	The number of octets contained in *ACLSecurityMaterial.*	0x15
macDefault-SecurityMate-rial	0x74	octet string	Variable	The specific security material to be used to protect frames between the MAC sublayer and devices not in the ACL.	Empty string
macDefault-SecuritySuite	0x75	Integer	0x00–0x07	The unique identifier of the security suite to be used to protect communications between the MAC and devices not in the ACL.	0x00
macSecurity-Mode	0x76	Integer	0x00–0x02	The identifier of the security mode in use. 0x00 = Unsecured mode. 0x01 = ACL mode. 0x02 = Secured mode.	0x00

5.4.2. Confirm

```
// Type: gNwkGetCnf_c,
typedef struct nwkGetCnf_tag
{
                  unsigned char status;
                  unsigned char pibAttribute;
                  unsigned char PIBAttributeLength;
                  unsigned char *pibAttributeValue;
}nwkGetCnf_t;
```

Name	Type	Valid Range	Description
status	unsigned char	SUCCESS or UNSUPPORTED_ATTRIBUTE	The result of the request for MAC PIB attribute information.
pibAttribute	unsigned char	See tables in Section "5.4.1. Request" on page 18.	The identifier of the MAC PIB attribute that was read.
PIBAttributeLength	unsigned char		
pibAttributeValue	Various	Attribute specific; see tables in Section "5.4.1. Request" on page 18.	The value of the indicated MAC PIB attribute that was read.

5.5. MLME-GTS

5.5.1. Request

```
// Type: gMlmeGtsReq_c,
typedef struct mlmeGtsReq_tag
{
                  unsigned char securityEnable;
                  unsigned char gtsCharacteristics;
}mlmeGtsReq_t;
```

Name	Type	Valid Range	Description
securityEnable	unsigned char	TRUE or FALSE	TRUE if security is enabled for this transfer or FALSE otherwise.
gtsCharacteristics	unsigned char	See Table 5	The characteristics of the GTS request.

Table 5. GTS Characteristics Field Format

Bits 0–3	4	5	6–7
GTS length	GTS direction	Characteristics type	Reserved

5.5.2. Indication

```
// Type: gNwkGtsInd_c,
typedef struct nwkGtsInd_tag
{
                  unsigned char devAddress[2];
                  unsigned char securityUse;
                  unsigned char AclEntry;
                  unsigned char gtsCharacteristics;
}nwkGtsInd_t;
```

Name	Type	Valid Range	Description
devAddress	unsigned char	0x0000–0xfffd	The short address of the device that has been allocated or deallocated by a GTS.
securityUse	unsigned char	TRUE or FALSE	An indication of whether the received frame is using security. This value is set to TRUE if the security enable subfield was set to 1 or FALSE if the security enabled subfield was set to 0.
AclEntry	unsigned char	0x00–0x08	The *macSecurityMode* parameter value from the ACL entry associated with the sender of the data frame. This value is set to 0x08 if the sender of the data frame was not found in the ACL.
gtsCharacteristics	GTS characteristics	See table in Section "5.5.1. Request" on page 23.	The characteristics of the GTS.

5.5.3. Confirm

```
// Type: gNwkGtsCnf_c,
typedef struct nwkGtsCnf_tag
{
                  unsigned char status;
                  unsigned char gtsCharacteristics;
}nwkGtsCnf_t;
```

Name	Type	Valid Range	Description
status	unsigned char	SUCCESS, DENIED, NO_SHORT_ADDRESS, CHANNEL_ACCESS_FAILURE, NO_ACK, NO_DATA, UNAVAILABLE_KEY, FAILED_SECURITY_CHECK, or INVALID_PARAMETER.	The status of the GTS request.
gtsCharacteristics	unsigned char	See table in Section "5.5.1. Request" on page 23.	The characteristics of the GTS.

5.6. MLME-ORPHAN

5.6.1. Indication

```
// Type: gNwkOrphanInd_c,
typedef struct nwkOrphanInd_tag
{
                  unsigned char orphanAddress[8];
                  unsigned char securityUse;
                  unsigned char AclEntry;
}nwkOrphanInd_t;
```

Name	Type	Valid Range	Description
orphanAddress	unsigned char	Extended 64-bit IEEE address	The address of the orphaned device.
securityUse	unsigned char	TRUE or FALSE	An indication of whether the received MAC command frame is using security. This value is set to TRUE if the security enable subfield was set to 1 or FALSE if the security enabled subfield was set to 0.
AclEntry	unsigned char	0x00–0x08	The *macSecurityMode* parameter value from the ACL entry associated with the sender of the data frame. This value is set to 0x08 if the sender of the data frame was not found in the ACL.

5.6.2. Response

```
// Type: gMlmeOrphanRes_c,

typedef struct mlmeOrphanRes_tag
{
                  unsigned char orphanAddress[8];
                  unsigned char shortAddress[2];
                  unsigned char securityEnable;
                  unsigned char associatedMember;
}mlmeOrphanRes_t;
```

Name	Type	Valid Range	Description
orphanAddress	unsigned char	Extended 64-bit IEEE address	The address of the orphaned device.
shortAddress	unsigned char	0x0000–0xFFFF	The 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.
securityEnable	Boolean	TRUE or FALSE	TRUE if security is enabled for this transfer or FALSE otherwise.
associatedMember	Boolean	TRUE or FALSE	TRUE if the orphaned device is associated with this coordinator or FALSE otherwise.

5.7. MLME-RESET

5.7.1. Request

The Reset-Request message is processed immediately. No confirm message is generated. Instead, the return code is used as the status value.

```
// Type: gMlmeResetReq_c,
typedef struct mlmeResetReq_tag
{
                    unsigned char setDefaultPib;
}mlmeResetReq_t;
```

Name	Type	Valid Range	Description
setDefaultPIB	unsigned char	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.

5.7.2. Confirm

The Reset-Confirm is not used because the Reset is carried out when called. The Confirm status code is returned by the SAP function that sends the Reset-Request message to the MLME.

```
// Type: gNwkResetCnf_c,
typedef struct nwkResetCnf_tag
{
                  unsigned char status;
}nwkResetCnf_t;
```

Name	Type	Valid Range	Description
status	unsigned char	SUCCESS or DISABLE_TRX_FAILURE	The result of the reset operation.

5.8. MLME-RX-ENABLE

5.8.1. Request

```
// Type: gMlmeRxEnableReq_c,
typedef struct mlmeRxEnableReq_tag
{
                  unsigned char deferPermit;
                  unsigned char rxOnTime[3];
                  unsigned char rxOnDuration[3];
}mlmeRxEnableReq_t;
```

Name	Type	Valid Range	Description
deferPermit	unsigned char	TRUE or FALSE	TRUE if the receiver enable can be deferred until the next superframe if the requested time has already passed. FALSE if the receiver enable is only to be attempted in the current superframe. This parameter is ignored for nonbeacon-enabled PANs.
rxOnTime	unsigned char	0x000000–0xFFFFFF	The number of symbols from the start of the superframe before the receiver is to be enabled. The precision of this value is a minimum of 20 bits, with the lowest four bits being the least significant. This parameter is ignored for non-beacon-enabled PANs.
rxOnDuration	unsigned char	0x000000–0xFFFFFF	The number of symbols for which the receiver is to be enabled.

5.8.2. Confirm

```
// Type: gNwkRxEnableCnf_c,
typedef struct nwkRxEnableCnf_tag
{
                  unsigned char status;
}nwkRxEnableCnf_t;
```

Name	Type	Valid Range	Description
status	unsigned char	SUCCESS, TX_ACTIVE, OUT_OF_CAP, or INVALID_PARAMETER	The result of the receiver enable request.

5.9. MLME-SCAN

5.9.1. Request

Ensure that `scanChannels` always indicates at least one valid channel and that channels outside the valid range [11:26] are not used. The value, 0x07FFF800, corresponds to all valid channels for 2.4 GHz radio. The valid range for `scanType` is [0:3]. The valid range for `scanDuration` is [0:14].

```
// Type: gMlmeScanReq_c,
typedef struct mlmeScanReq_tag
{
                  unsigned char scanType;
                  unsigned long scanChannels;
                  unsigned char scanDuration;
}mlmeScanReq_t;
```

Name	Type	Valid Range	Description
scanType	unsigned char	0x00–0x03	Indicates the type of scan performed: 0x00 = ED scan (FFD only). 0x01 = active scan (FFD only). 0x02 = passive scan. 0x03 = orphan scan.
scanChannels	unsigned long	32-bit field	The five MSBs (b27, ... , b31) are reserved. The 27 LSBs (b0, b1, ... b26) indicate which channels are to be scanned (1 = scan, 0 = do not scan) for each of the 27 valid channels. (See "5.1.2. Indication" on page 12.)
scanDuration	unsigned char	0–14	A value used to calculate the length of time to spend scanning each channel for ED, active, and passive scans. This parameter is ignored for orphan scans. The time spent scanning each channel is [*aBaseSuperframeDuration* * (2*n*+ 1)] symbols, where *n* is the value of the ScanDuration parameter.

5.9.2. Confirm

The Scan-Confirm structure contains a pointer to an array of descriptors. The energy detect list pointer is a fixed buffer that must not be freed. All other parameters map exactly as shown to the parameters listed and described in the 802.15.4 specification.

```
// Type: gNwkScanCnf_c,
typedef struct nwkScanCnf_tag
{
                  unsigned char scanType;
                  unsigned long unscannedChannels;
                  unsigned char resultListSize;
                  union
                  {
                        unsigned char *pEnergyDetectList;
                        PAN_DESCRIPTOR *pPanDescriptorList;

}resList;
                  unsigned char status;
}nwkScanCnf_t;
```

Name	Type	Valid Range	Description
scanType	unsigned char	0x00–0x03	Indicates the type of scan performed: 0x00 = ED scan (FFD only). 0x01 = active scan (FFD only). 0x02 = passive scan. 0x03 = orphan scan.
unscannedChannels	unsigned long	32-bit field	Indicates which channels given in the request were not scanned (1 = not scanned, and 0 = scanned or not requested). This parameter is only valid for passive or active scans.
resultListSize	unsigned char		
resList	union		
status	unsigned char	SUCCESS, NO_BEACON, or INVALID_PARAMETER	The status of the scan request.

5.10. MLME-COMM-STATUS

5.10.1. Indication

```
// Type: gNwkCommStatusInd_c,
typedef struct nwkCommStatusInd_tag //MLME_COMM_STATUS_indication
{
                  unsigned char panId[2];
                  unsigned char srcAddrMode;
                  unsigned char srcAddress[8];
                  unsigned char destAddrMode;
                  unsigned char destAddress[8];
                  unsigned char status;
}nwkCommStatusInd_t;
```

Name	Type	Valid Range	Description
panId	unsigned char	0x0000–0xFFFF	The 16-bit PAN identifier of the device from which the frame was received or to which the frame was being sent.
srcAddrMode	unsigned char	0x00–0x03	The source addressing mode for this primitive. This value can take one of the following values: 0 = No address (addressing fields omitted). 0x 01 = reserved. 0x02 = 16-bit short address. 0x 03 = 64-bit extended address.
srcAddress	unsigned char	As specified by the SrcAddrMode parameter	The individual device address of the entity from which the frame causing the error originated.
destAddrMode	unsigned char	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.
destAddress	unsigned char	As specified by the destAddrMode parameter	The individual device address of the device for which the frame was intended.
status	unsigned char	SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, CHANNEL_ACCESS_FAILURE, NO_ACK, UNAVAILABLE_KEY, FRAME_TOO_LONG, FAILED_SECURITY_CHECK or INVALID_PARAMETER	The communications status.

5.11. MLME-SET

5.11.1. Request

The Set-Request message contains a pointer to the data to be written to the MAC PIB. The pointer must be supplied by the NWK or APP. Attributes with a size of more than one byte must be little-endian and given as byte arrays. Because the Set-Request message is processed immediately, no confirm message is generated. Instead, the return code is used as the status value. When the Set-Request is used for setting the beacon payload, the beacon payload length attribute must be set first. Otherwise, the MLME has no way to tell how many bytes to copy.

```
// Type: gMlmeSetReq_c,
typedef struct mlmeSetReq_tag
{
                  unsigned char pibAttribute;
                  unsigned char *pibAttributeValue; // Pointer supplied by NWK
}mlmeSetReq_t;
```

Name	Type	Valid Range	Description
pibAttribute	unsigned char	See tables in Section "5.4.1. Request" on page 18.	The identifier of the MAC PIB attribute to write.
pibAttributeValue	Various	Attribute specific; see tables in Section "5.4.1. Request" on page 18.	The value to write to the indicated MAC PIB attribute.

5.11.2. Confirm

```
typedef struct nwkSetCnf_tag
{
                  unsigned char status;
                  unsigned char pibAttribute;
}nwkSetCnf_t;
```

Name	Type	Valid Range	Description
status	unsigned char	SUCCESS, UNSUPPORTED_ATTRIBUTE, or INVALID_PARAMETER	The result of the request to write the MAC PIB attribute.
PIBAttribute	unsigned char	See tables in Section "5.4.1. Request" on page 18.	The identifier of the MAC PIB attribute that was written.

5.12. MLME-START

5.12.1. Request

Before sending a Start-Request, the `macShortAddress` must be set to something other than 0xFFFF.

```
// Type: gMlmeStartReq_c,
typedef struct mlmeStartReq_tag
{
                  unsigned short panId;
                  unsigned char logicalChannel;
                  unsigned char beaconOrder;
                  unsigned char superFrameOrder;
                  unsigned char panCoordinator;
                  unsigned char batteryLifeExt;
                  unsigned char coordRealignment;
                  unsigned char securityEnable;
}mlmeStartReq_t;
```

Name	Type	Valid Range	Description
panId	unsigned short	0x0000–0xFFFF	The PAN identifier to be used by the beacon.
logicalChannel	unsigned char	Selected from the available logical channels supported by the PHY	The logical channel on which to start transmitting beacons.
beaconOrder	unsigned char	0–15	How often the beacon is to be transmitted. The beacon order, *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 will not transmit a beacon, and the SuperframeOrder parameter value is ignored.
superFrameOrder	unsigned char	0–*BO* or 15	The length of the active portion of the superframe, including the beacon frame. The superframe order, *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 will not be active after the beacon.
panCoordinator	unsigned char	TRUE or FALSE	If this value is TRUE, the device will become the PAN coordinator of a new PAN. If this value is FALSE, the device will begin transmitting beacons on the PAN with which it is associated.

Name	Type	Valid Range	Description
batteryLifeExt	unsigned char	TRUE or FALSE	If this value is TRUE, the receiver of the beaconing device is disabled *mac-BattLifeExtPeriods* full backoff periods after the interframe spacing (IFS) period of the beacon frame. If this value is FALSE, the receiver of the beaconing device remains enabled for the entire CAP.
coordRealignment	unsigned char	TRUE or FALSE	TRUE if a coordinator realignment command is to be transmitted prior to changing the superframe configuration or FALSE otherwise.
securityEnable	unsigned char	TRUE or FALSE	TRUE if security is enabled for beacon transmissions or FALSE otherwise.

5.12.2. Confirm

```
// Type: gNwkStartCnf_c,
typedef struct nwkStartCnf_tag
{
                  unsigned char status;
}nwkStartCnf_t;
```

Name	Type	Valid Range	Description
status	unsigned char	SUCCESS, NO_SHORT_ADDRESS, UNAVAILABLE_KEY, FRAME_TOO_LONG, FAILED_SECURITY_CHECK, or INVALID_PARAMETER	The result of the attempt to start using an updated superframe configuration.

5.13. MLME-SYNC

5.13.1. Request

```
// Type: gMlmeSyncReq_c,
typedef struct mlmeSyncReq_tag
{
                  unsigned char logicalChannel;
                  unsigned char trackBeacon;
}mlmeSyncReq_t;
```

Name	Type	Valid Range	Description
logicalChannel	unsigned char	Selected from the available logical channels supported by the PHY	The logical channel on which to attempt coordinator synchronization.
trackBeacon	unsigned char	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.

5.14. MLME-SYNC-LOSS

5.14.1. Indication

```
// Type: gNwkSyncLossInd_c,
typedef struct nwkSyncLossInd_tag
{
                  unsigned char lossReason;
}nwkSyncLossInd_t;
```

Name	Type	Valid Range	Description
lossReason	unsigned char	PAN_ID_CONFLICT, REALIGNMENT, or BEACON_LOST	The reason that synchronization was lost.

5.15. MLME-POLL

5.15.1. Request

```
// Type: gMlmePollReq_c,
typedef struct mlmePollReq_tag
{
                  unsigned char coordAddress[8];
                  unsigned char coordPanId[2];
                  unsigned char coordAddrMode;
                  unsigned char securityEnable;
}mlmePollReq_t;
```

Name	Type	Valid Range	Description
coordAddress	unsigned char	As specified by the CoordAddrMode parameter	The address of the coordinator to which the poll is intended.
coordPANId	unsigned char	0x0000–0xFFFE	The PAN identifier of the coordinator to which the poll is intended.
coordAddrMode	unsigned char	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.
SecurityEnable	unsigned char	TRUE or FALSE	TRUE if security is enabled for this transfer or FALSE otherwise.

5.15.2. Confirm

```
// Type: gNwkPollCnf_c,
typedef struct nwkPollCnf_tag
{
                  unsigned char status;
}nwkPollCnf_t;//
```

Name	Type	Valid Range	Description
status	unsigned char	SUCCESS, CHANNEL_ACCESS_FAILURE, NO_ACK, NO_DATA, UNAVAILABLE_KEY, FAILED_SECURITY_CHECK, or INVALID_PARAMETER	The status of the data request.

APPENDIX A—APPLICATION TO MAC SHARED DATA STRUCTURES

nwk2mac.h

Application to MAC

These data structures are defined in the nwk2mac.h header file.

The network to MAC message data type is a structure including the primitive type `primitive` and network to MAC message union.

```
typedef struct nwk2macMsg
{
                  //define primitive type
                  unsigned char primtype;
                  //primitive information
                  union n_m_msg        msg;
                  //primitive start time which indicated that the primitive perform
time in superframe in mac layer for tuv test only
                  unsigned char        Start_t;
};
extern struct nwk2macMsg EXTDATA n_m_msg_t;
```

The network to MAC message data type is a union of all the primitive message data structures.

```
union n_m_msg
{
                  //mlmeAssociateRequest
                  mlmeAssociateReq_t    MLME_ASSOCIATE_request;
                  //mlmeAssociateResponse
                  mlmeAssociateRes_t    MLME_ASSOCIATE_response;
                  //mlmeDisassociateRequest
                  mlmeDisassociateReq_t MLME_DISASSOCIATE_request;
                  //mlmeGetRequest
                  mlmeGetReq_t          MLME_GET_request;
                  //mlmeGtsRequest
                  mlmeGtsReq_t          MLME_GTS_request;
                  //mlmeOrphanResponse
                  mlmeOrphanRes_t       MLME_ORPHAN_response;
                  //mlmeResetRequest
                  mlmeResetReq_t        MLME_RESET_request;
                  //mlmeRxEnableRequest
                  mlmeRxEnableReq_t     MLME_RX_ENABLE_request;
                  //mlmeScanRequest
                  mlmeScanReq_t         MLME_SCAN_request;
                  //mlmeSetRequest
                  mlmeSetReq_t          MLME_SET_request;
                  //mlmeStartRequest
                  mlmeStartReq_t        MLME_START_request;
                  //mlmeSyncRequest
                  mlmeSyncReq_t         MLME_SYNC_request;
```

```
                    //mlmePollRequest
                    mlmePollReq_t          MLME_POLL_request;
                    //mcpsDataRequest
                    mcpsDataReq_t          MCPS_DATA_request;
                    //mcpsPurgeRequest
                    mcpsPurgeReq_t         MCPS_PURGE_request;
};
```

Message data structures for each individual primitive are described in detail in Sections "4. MAC Common Part Sublayer (MCPS-) Primitives" on page 5 and "5. MAC Sublayer Management Entity (MLME-) Primitives" on page 10. An enumerated type is used to define the contents of the primitive type “primtype” corresponding to each primitive.

```
enum n_m_primtype_t
{
                    //0 MLME-RESET.Request
                    gMlmeResetReq_c,
                    //1 MLME-SET.Request
                    gMlmeSetReq_c,
                    //2 MLME-GET.Request
                    gMlmeGetReq_c,
                    //3 MLME-START.Request
                    gMlmeStartReq_c,
                    //4 MLME-ASSOCIATE.Request
                    gMlmeAssociateReq_c,
                    //5 MLME-DISASSOCIATE.Request
                    gMlmeDisassociateReq_c,
                    //6 MLME-SCAN.Request
                    gMlmeScanReq_c,
                    //7 MLME-ASSOCIATE.Response
                    gMlmeAssociateRes_c,
                    //8 associate indication
                    tmpASSOCIATEINDICATION,
                    //9 disassociate indication
                    tmpDISASSOCIATEINDICATION,
                    //a orphan indication
                    tmpORPHANINDICATION,
                    //b MLME-ORPHAN.Response
                    gMlmeOrphanRes_c,
                    //c MLME-POLL.Request
                    gMlmePollReq_c,
                    //d MCPS-DATA.Request
                    gMcpsDataReq_c,
                    //e data indication
                    tmpDATAINDICATION,
                    //f //
                    tmp1,
                    //10
                    tmp2,
                    //11 MLME-RX-ENABLE.Request
                    gMlmeRxEnableReq_c,
```

```
                    //12
                    tmp11,
                    //13
                    gMcpsPurgeReq_c,
                    //14 MLME-GTS.Request
                    gMlmeGtsReq_c,
                    //15 MLME-GTS.Indication
                    tmpMLMEGTSINDICATION,
                    //16 MLME-SYNC.Request
                    gMlmeSyncReq_c,
                    //17
                     ntmp17,
                     //18
                    ntmp18,
                     //19
                    ntmp19,
                     //1a
                    gMcpsExdAddrReq_c,
                     //1b
                    fMasdftemp,
                    //1c
                    gMlmePowerMane_c
};
```

APPENDIX B—MAC TO APPLICATION SHARED DATA STRUCTURES

These data structures are defined in the mac2nwk.h header file. The MAC to network data type includes the primitive type `primitive` and the MAC for network message union. The MAC to network buffer is an array of MAC-to-network message structures. The MAC-to-network message `mac_msg` is a union of all the primitive message data structures.

```
typedef struct mac2nwkMsg
{
                  //primitive type
                  unsigned char  primtype;
                  //primitive message
                  union mac_msg        msg;
};

//an interface array of mac and NWK layer
extern struct mac2nwkMsg EXTDATA m2n_msg[TransactionBuffLenth];

union mac_msg
{
                  //type of primitive
                  nwkAssociateInd_t    MLME_ASSOCIATE_indication;
                  nwkAssociateCnf_t    MLME_ASSOCIATE_confirm;
                  nwkDisassociateInd_t MLME_DISASSOCIATE_indication;
                  nwkDisassociateCnf_t MLME_DISASSOCIATE_confirm;
                  nwkBeaconNotifyInd_t MLME_BEACON_NOTIFY_indication;
                  nwkGtsInd_t          MLME_GTS_indication;
                  nwkGtsCnf_t          MLME_GTS_confirm;
                  nwkOrphanInd_t       MLME_ORPHAN_indication;
                  nwkResetCnf_t        MLME_RESET_confirm;
                  nwkRxEnableCnf_t     MLME_RX_ENABLE_confirm;
                  nwkScanCnf_t         MLME_SCAN_confirm;
                  nwkCommStatusInd_t   MLME_COMM_STATUS_indication;
                  nwkStartCnf_t        MLME_START_confirm;
                  nwkSetCnf_t          MLME_SET_confirm;
                  nwkSyncLossInd_t     MLME_SYNC_LOSS_indication;
                  nwkPollCnf_t         MLME_POLL_confirm;
                  mcpsDataCnf_t        MCPS_DATA_confirm;
                  mcpsDataInd_t        MCPS_DATA_indication;
                  mcpsPurgeCnf_t       MCPS_PURGE_confirm;
                  nwkGetCnf_t          MLME_GET_confirm; // Not used
                  unsigned char        ExtendAddress[8];
};
```

An enumerated type is used to define the contents of the primitive type “primtype” corresponding to each primitive.

```
enum m_n_primtype_t
{
                  //0 MLME-RESET.Confirm
                  gNwkResetCnf_c,
```

```
//1 Not supported for all Device Types
gNwkSetCnf_c,
//2 Not supported for all Device Types
gNwkGetCnf_c,
//3 MLME-START.Confirm
gNwkStartCnf_c,
//4 MLME-ASSOCIATE.Confirm
gNwkAssociateCnf_c,
//5 MLME-DISASSOCIATE.Confirm
gNwkDisassociateCnf_c,
//6 MLME-SCAN.Confirm
gNwkScanCnf_c,
//7for comm test
tmpASSOCIATERESPONSE,
//8 MLME-ASSOCIATE.Indication
gNwkAssociateInd_c,
//9 MLME-DISASSOCIATE.Indication
gNwkDisassociateInd_c,
//a Not supported for all Device Types
gNwkOrphanInd_c,
//b for comm test
tmpORPHANRESPONSE,
//c MLME-POLL.Confirm
gNwkPollCnf_c,
//d
gMcpsDataCnf_c,
//e
gMcpsDataInd_c,
//f MLME-BEACON-NOTIFY.Indication
gNwkBeaconNotifyInd_c,
//10 MLME-COMM-STATUS.Indication
 gNwkCommStatusInd_c,
//11 MLME-RX-ENABLE.Confirm
gNwkRxEnableCnf_c,
//12 MLME-SYNC-LOSS.Indication
gNwkSyncLossInd_c,
//13
gMcpsPurgeCnf_c,
//14 MLME-GTS.Confirm
gNwkGtsCnf_c,
//15 MLME-GTS.Indication
gNwkGtsInd_c,
//16
tmp16,
//17
 tmp17,
 //18
 tmp18,
//19
tmp19,
```

```
                  //1a
                   gMcpsExdAddrInd_c
};
```

Macros are used to define the values returned by the status parameter in all confirmed primitives. These values are defined by the 802.15.4 specification.

```
//mac enumerations description
#define SUCCESS                     0x00
#define BEACON_LOSS                 0xE0
#define CHANNEL_ACCESS_FAILURE      0xE1
#define DENIED                      0xE2
#define DISABLE_TRX_FAILURE         0xE3
#define FAILED_SECURITY_CHECK       0xE4
#define FRAME_TOO_LONG              0xE5
#define INVALID_GTS                 0xE6
#define INVALID_HANDLE              0xE7
#define INVALID_PARAMETER           0xE8
#define NO_ACK                      0xE9
#define NO_BEACON                   0xEA
#define NO_DATA                     0xEB
#define NO_SHORT_ADDRESS            0xEC
#define OUT_OF_CAP                  0xED
#define PAN_ID_CONFLICT             0xEE
#define REALIGNMENT                 0xEF
#define TRANSACTION_EXPIRED         0xF0
#define TRANSACTION_OVERFLOW        0xF1
#define TX_ACTIVE                   0xF2
#define UNAVAILABLE_KEY             0xF3
#define UNSUPPORTED_ATTRIBUTE       0xF4
#define MISSING_ADDRESS             0xF5
#define PAST_TIME                   0xF6
```

APPENDIX C—GLOBAL MAC SHARED DATA STRUCTURES

These data structures are defined in the mac.h header file. The Pan Descriptor structure is a list of information describing each PAN located during a passive or active scan.

```
typedef struct tag_PAN_DESCRIPTOR
{
                  //coordinator address mode
                  unsigned char  byCoordAddrMode;
                  //coordinator pan id
                  unsigned short dwCoordPANId;
                  //coordinator address
                  unsigned char  CoordAddress[8];
                  //logical channel
                  unsigned char  byLogicalChannel;
                  //superframe specific
                  unsigned short dwSuperframeSpec;
                  //gts permit tag
                  unsigned char  byGTSPermit;
                  //link quality
                  unsigned char  byLinkQuality;
                  //time stamp
                  unsigned long  dqTimeStamp;
                  //not support in this version
                  unsigned char  bySecurityUse;
                  //for security, not support in this version
                  unsigned char  byACLEntry;//
                  //not support in this version
                  unsigned char  bySecurityFailure;
}PAN_DESCRIPTOR;
```

NOTES:

CONTACT INFORMATION

Silicon Laboratories Inc.

4635 Boston Lane
Austin, TX 78735
Email:MCUinfo@silabs.com
Internet: www.silabs.com