2.4 GHz ZigBee™ Network Application Interface Programmer's Guide

1. Introduction

This document describes the Silicon Laboratories ZigBee Network Layer interface. It contains implementation details specific to the Network-layer interface software library included as part of the Silicon Laboratories ZigBee Development Kit.

This document should be used in conjunction with the ZigBee Alliance's Network Specification.

Current firmware releases do not support beacon-based networks or security.

2. Overview of Primitive Implementation

Messaging between the application layer and the network layer is implemented either by direct function calls or by using a shared buffer. Primitives transmitted from the application layer to the network layer are implemented using direct function calls.

In contrast, primitives sent from the network layer to the application layer are implemented differently. Indication primitives notifying an event to the application layer will be stored in a shared buffer. The application layer needs to poll this buffer for an incoming event.

Some confirmation primitives carry only one parameter, normally a status indicator corresponding to a request. The parameter is conveyed as a return value of the requesting function call. Thus, there is no explicit implementation of these primitives.

Other confirmation primitives contain more than one parameter. When a request is called, the function call of the request will store the confirmation data to the shared buffer. The caller of the request shall check the buffer for confirmation when the request returns.

Table 1. Primitive Implementation

Primitive	Implementation
Request	Direct function call
Confirm (one parameter)	Return value of the request function call
Confirm (multiple parameters)	Globally shared buffer
Indication	Globally shared buffer

3. Creating User Applications

The Application layer must start the ZigBee network in a specified sequence. This section describes initialization, network startup, and joining procedures.

3.1. System Initialization

These functions should be called in order as the node is initially powered up.

Disable Global Interrupts:

```
DISABLE_GLOBAL_INT();
```

Initialize System Hardware:

```
SystemInit();
```

Initialize Transceiver:

```
CC2420Init();
```

Initialize Transceiver Interrupt:

```
EINT_Init ();
```

Initialize MAC Internal Variables and Default PIB settings:

```
MAC_Init();
macInitEnv();
mlmeResetRequest(FALSE);
```

Initialize NWK layer:

```
netInit();
```

Enable Global Interrupts:

```
ENABLE_GLOBAL_INT();
```

3.2. Network-level Procedures

This section describes the processes of establishing, expanding, and dismantling a ZigBee network. Sections 4 and 5 describe each command primitive in detail.

A ZigBee network is established by the steps shown in Figure 1.

1. Reset and initialize each device as it is powered up.
2. Establish a ZigBee network by designating a Coordinator. The Coordinator calls specific primitives to form the network then permits other nodes to join. Refer to section 3.2.2.
3. Once a network is formed, other devices may join the network and transfer data to other nodes within the network. Refer to sections 3.2.3 and 3.2.4.
4. Devices may request removal from a network or a parent may force a node from the network. Refer to section 3.2.5.

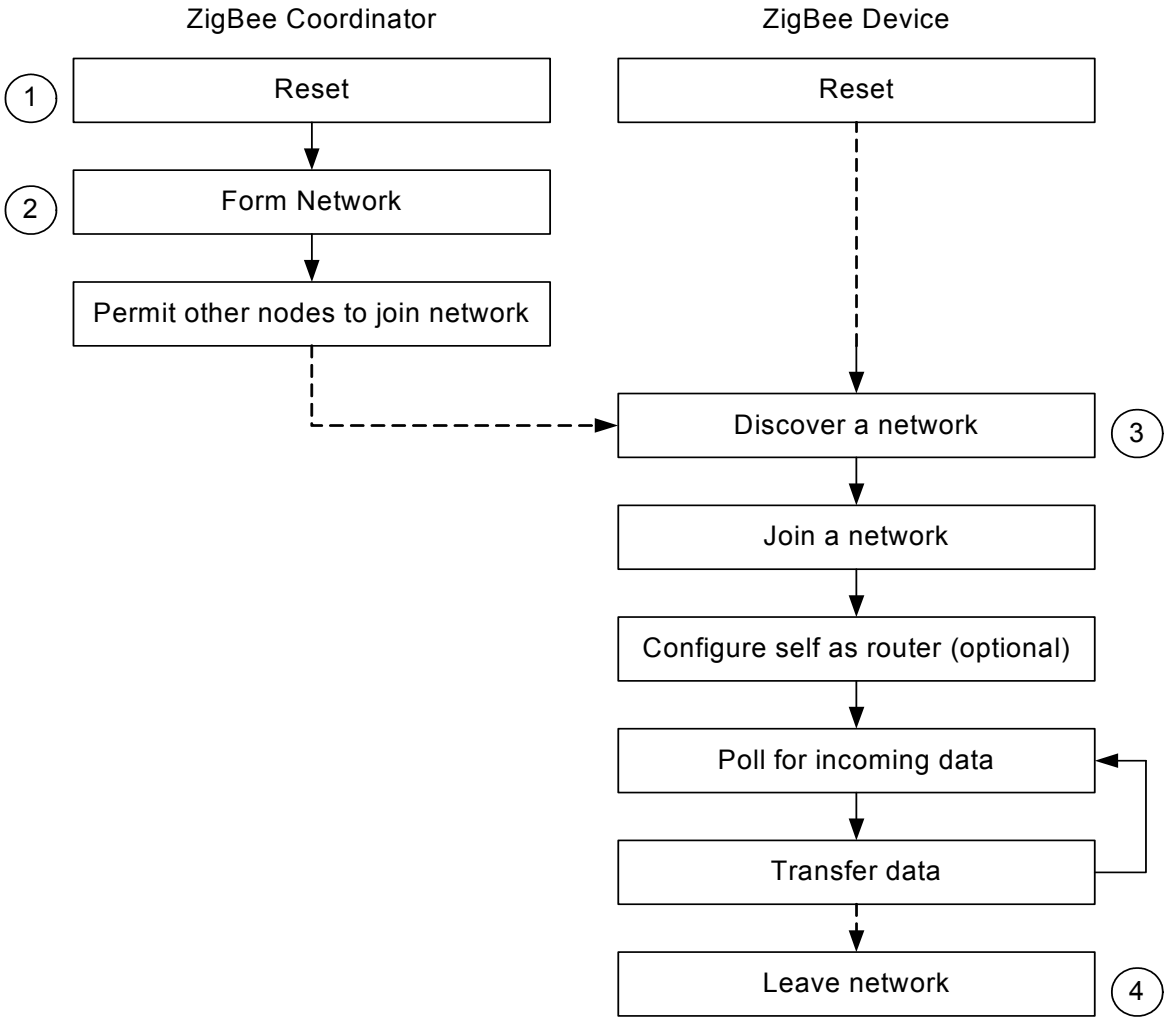

Figure 1. ZigBee Network Formation

3.2.1. Initialization

Each device must be reset via the NLME-RESET primitive immediately on powerup.

3.2.2. Starting a ZigBee Coordinator

A network is established by designating a node as the ZigBee Coordinator. The Application layer must first instruct the Coordinator to form a network then must permit other ZigBee devices to join the network. This is illustrated in Figure 2.

For a more robust design, the Coordinator's `nlmeNetworkFormationRequest()` will usually include an active scan to detect its neighboring environment. One possible SCAN outcome will be PAN ID conflict. This is a very important exception that programmers should be aware of so any problem can be handled and detected before the Coordinator starts.

"Permit Join" can be used creatively. Many examples have been discussed suggesting that "Permit Join" is toggled on and off by a simple push button. This is one idea how the Coordinator can fend off unsolicited JOIN requests and protect the integrity of the subject network.

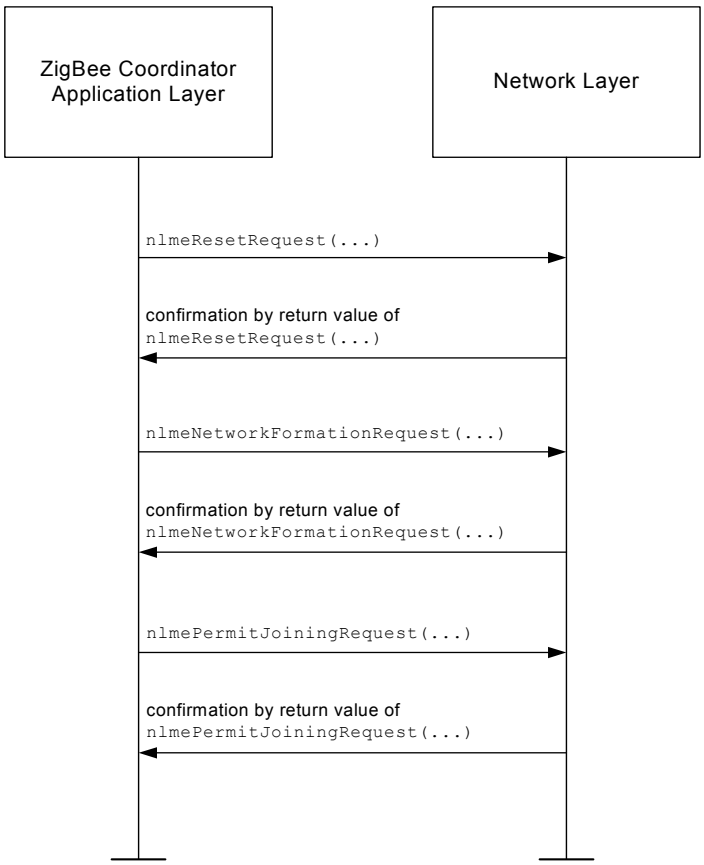

Figure 2. Establishing a Network and Enabling Nodes to Join

3.2.3. Constructing the Network

Remote devices may join once a core network has been established. New devices (children) can connect to existing devices (parents) through either association or direct connection.

3.2.3.1. Joining Through Association

In association, the child proactively discovers the network. Once discovered, the child requests a connection as shown in Figure 3.

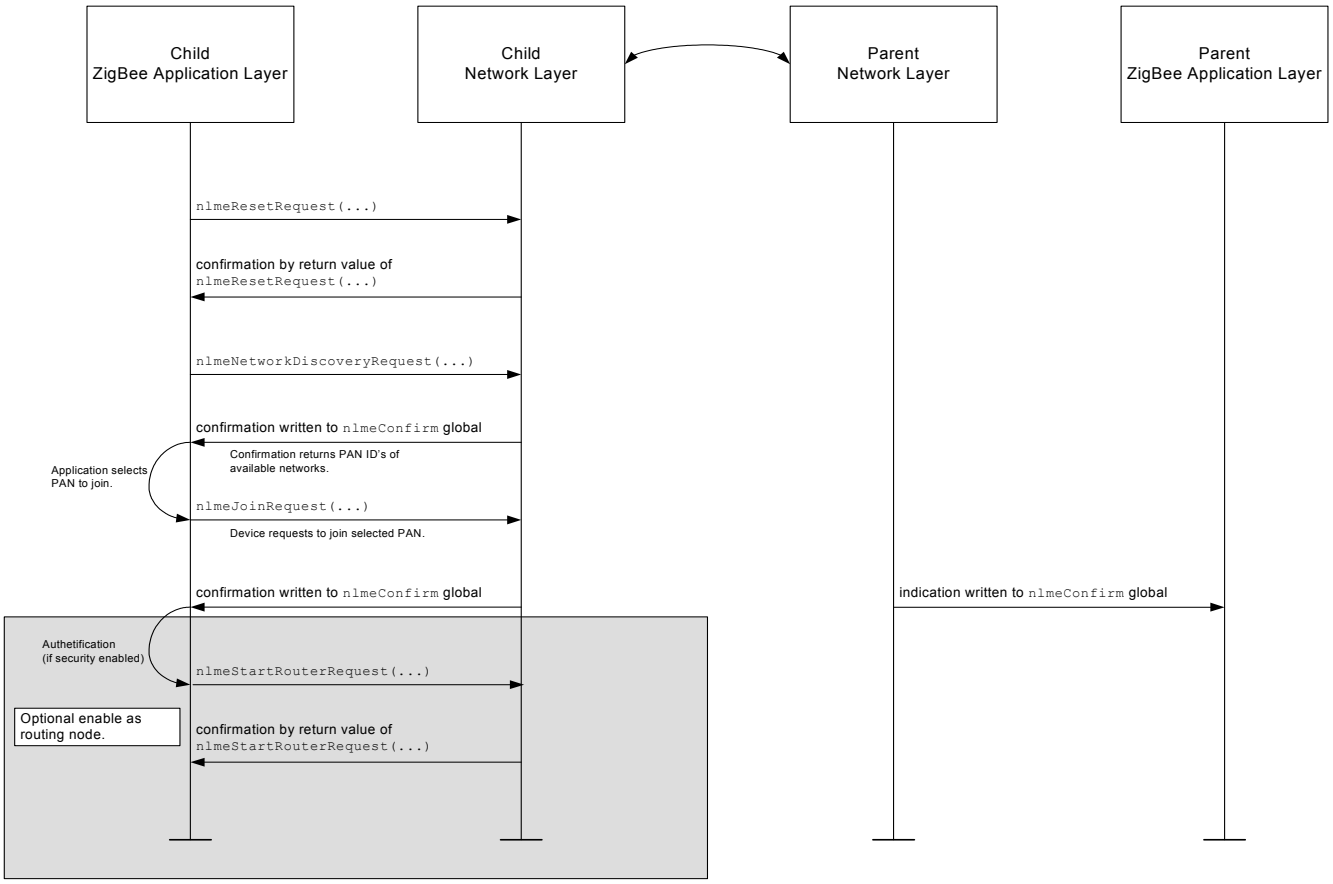

Figure 3. Child Joins Network Through Discovery and Association (Optional Configuration as a Routing Node After Association)

3.2.3.2. Direct Joining

Direct joining is used to reestablish a previous connection. The parent device first adds the child device back into its network. The child must then attempt reconnection. This is most commonly used when a child device is temporarily disconnected ("orphaned") from the network. This is illustrated in Figure 4.

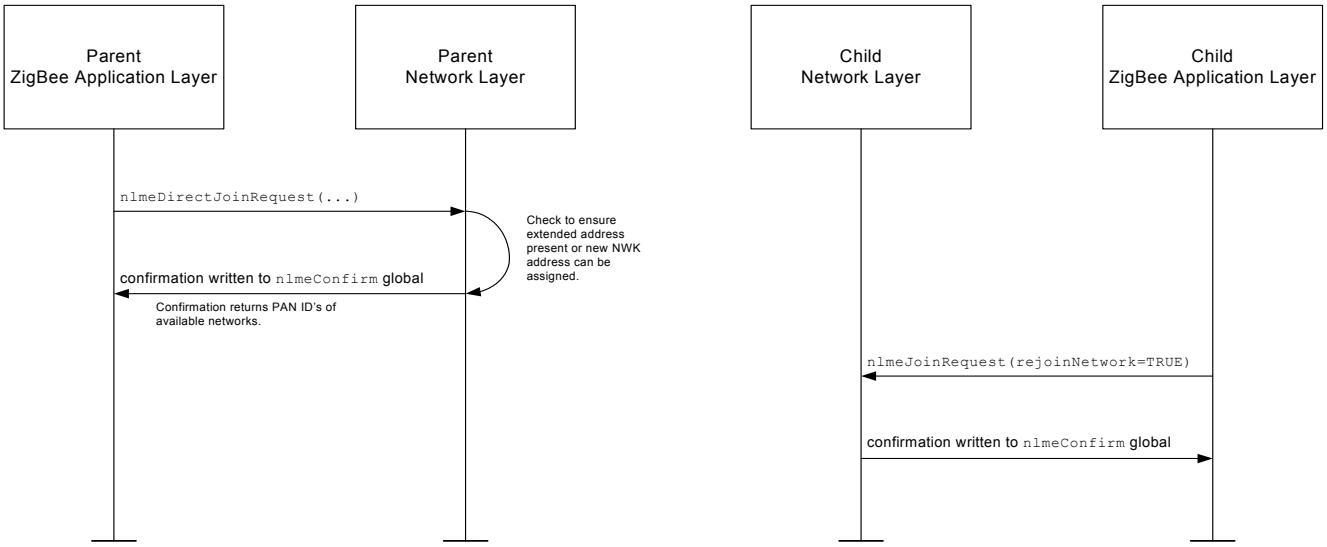

Figure 4. Rejoining a Network

3.2.4. Data Transfer

ZigBee networks can be categorized into two basic operating types, beacon-enabled networks and non-beacon networks. The network type is determined when the PAN Coordinator forms the network. The data transfer operation is different for the two types.

3.2.4.1. Uploading Data

In non-beacon networks, the Coordinator's receiver is always enabled. Thus, devices can send data to the Coordinator at any time*.

In beacon-enabled networks, devices need to synchronize with a beacon first, locate the appropriate timeslot, then send data in the designated periods of a superframe.

3.2.4.2. Downloading Data

When the Coordinator wants to send data to its Child devices, it needs to follow different procedures depending upon the receiver state of the destination device. Normally, if the receiver of the destination device is always enabled while idle, data will be sent out during the active periods of a superframe. Indirect transmission may also be used.

If the end device disables its receiver when idle, the Coordinator needs to use indirect data transmission. The firmware will put the data in an indirect queue for devices to poll.

3.2.4.3. Synchronizing and Polling for Data

It is the Application layer's responsibility to call `nlmeSyncRequest()` periodically to sync with its parent for pending data. The calling period is dependent upon specific applications.

In non-beacon networks, the sync request will trigger the lower layer to send a command requesting pending data from the Coordinator. In beacon-enabled networks, this request will enable a search for the next beacon and automatically request pending data if pending data are indicated in the beacon.

***Note:** It is possible that in non-beacon networks the whole system goes to sleep for a period in which devices cannot send data to Coordinators. Nevertheless, devices should normally be in a sleep state in that time too.

3.2.5. Dissolving the Network

Devices may be disconnected from the network one-by-one. A child may proactively disconnect from its parent, or the parent may break the network connection to a child, shown in Figure 5 and Figure 6, respectively.

3.2.5.1. Disconnection Initiated by Child

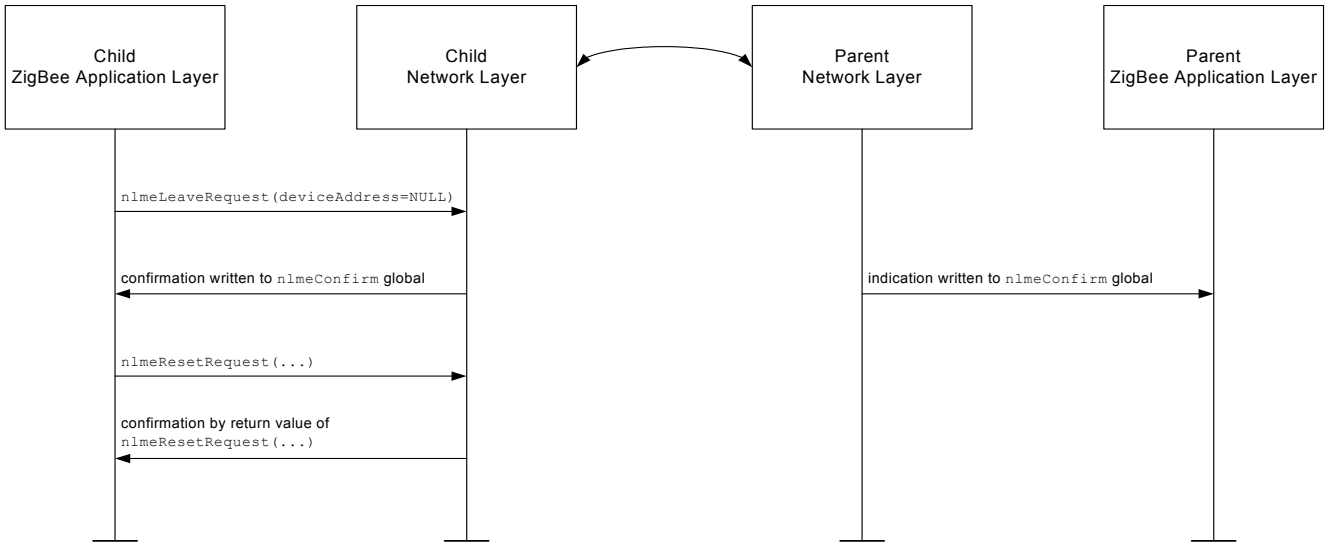

Figure 5. Child Initiates Disconnection from Network

3.2.5.2. Disconnection Initiated by Parent

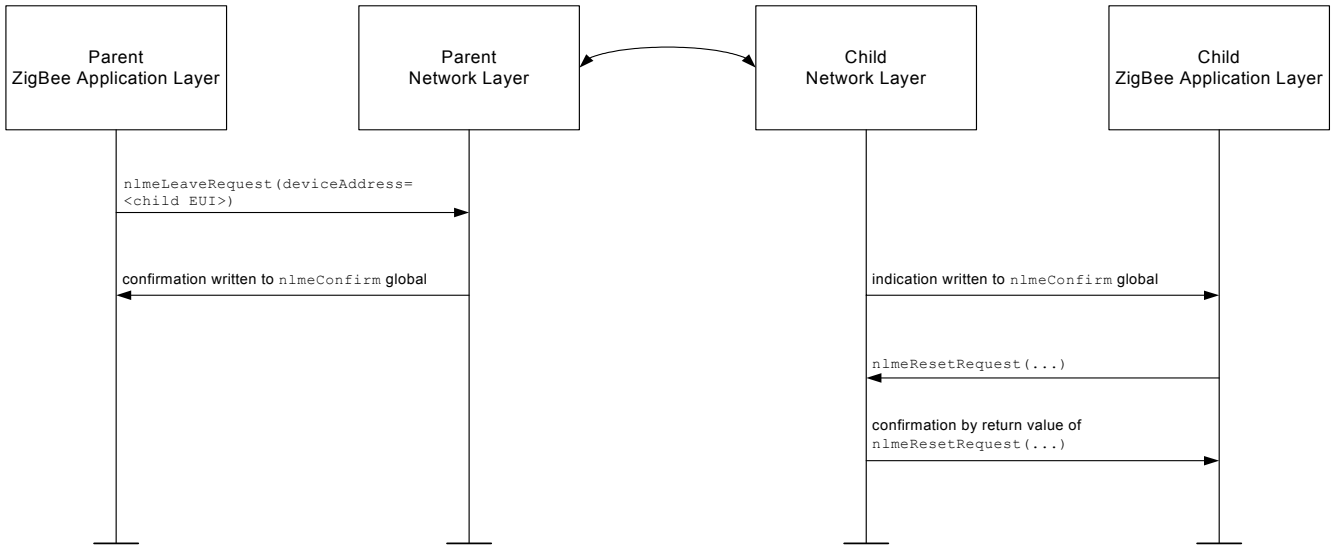

Figure 6. Parent Initiates Disconnection of Child

4. Network Layer Data Entity (NLDE-) Commands

Name	Request	Indication	Response	Confirm
NLDE-DATA	4.1.2	4.1.3		4.1.4

4.1. NLDE-DATA

4.1.1. Description

Applicability: All devices.

Prerequisites: Device must be associated.

4.1.2. Request

Description: This primitive requests the transfer of a data PDU (NSDU) from the local APS sub-layer entity to a single or multiple peer APS sub-layer entity.

Function Prototype

```
void nldeDataRequest(NLDE_DATA_REQUEST
*pNldeDataRequest);
```

Parameters:

```
typedef struct{
   WORD  dstAddr;
   BYTE  nsduHandle;
   BYTE  broadcastRadius;
   BOOL  discoverRoute;
   BOOL  securityEnable;
   BYTE  nsduLength;
   BYTE  *pNsdu;
}NLDE_DATA_REQUEST;
```

(defined in HS_NET.h)

`NLDE_DATA_REQUEST *pNldeDataRequest`
A pointer to the data structure of `NLDE_DATA_REQUEST` where all the arguments are available for the function.

`WORD dstAddr`
The network address of the entity or entities to which the NSDU is being transferred.

`BYTE nsduHandle`
The handle associated with the NSDU to be transmitted by the NWK layer entity.

`BYTE broadcastRadius`
The distance, in hops, that a broadcast frame will be allowed to travel through the network.

`BOOL discoverRoute`
The `DiscoverRoute` parameter may be used to enable route discovery operations for the transit of this frame.

`BOOL securityEnable`
The `SecurityEnable` parameter may be used to enable NWK layer security processing for the current frame.

`BYTE nsduLength`
The number of octets comprising the NSDU to be transferred.

`BYTE * pNsdu`
A pointer to the packet payload.

4.1.3. Indication

Description: Indication written to `nlmeConfirm` global.

Results:

```
nlmeConfirm.confirmId = N_DATA_IND;
nlmeConfirm.buffer structure:

typedef struct
{
   WORD    srcAddress;
   BYTE    linkQuality;
   BYTE    nsduLength;
   BYTE    *pNsdu;
}NLDE_DATA_INDICATION;   (defined in HS_NET.h)
```

`WORD srcAddress`
The individual device address from which the NSDU originated.

`BYTE linkQuality`
The link quality indication delivered by the MAC on receipt of this frame as a parameter of the `MCPS-DATA.indication` primitive.

BYTE nsduLength
The number of octets comprising the NSDU being indicated.

`BYTE* pNsdu`
The pointer to the set of octets comprising the NSDU being indicated.

4.1.4. Confirm

Description: Confirmation written to `nlmeConfirm` global.

Results:

```
nlmeConfirm.confirmId = N_DATA_CFM;
nlmeConfirm.buffer structure:

typedef struct
{
   NWK_ENUM   status;
   BYTE       nsduHandle;
}NLDE_DATA_CONFIRM;      (defined in HS_NET.h)
```

`NWK_ENUM status`
`INVALID_REQUEST` or any status values returned from security suite or the `MCPSDATA.confirm` primitive `(SUCCESS | TRANSACTION_OVERFLOW | TRANSACTION_EXPIRED | CHANNEL_ACCESS_FAILURE | INVALID_GTS | NO_ACK | UNAVAILABLE_KEY | FRAME_TOO_LONG | FAILED_SECURITY_CHECK)`

`BYTE nsduHandle`
A handle to this packet from the `mcpsDataRequest()` function.

5. Network Layer Management Entity (NLME-) Commands

Name	Request	Indication	Response	Confirm
NLME-NETWORK-DISCOVERY	5.1.2			5.1.3
NLME-NETWORK-FORMATION	5.2.2			5.2.3
NLME-PERMIT-JOINING	5.3.2			5.3.3
NLME-START-ROUTER	5.4.2			5.4.3
NLME-JOIN	5.5.2	5.5.3		5.5.4
NLME-DIRECT-JOIN	5.6.2			5.6.3
NLME-LEAVE	5.7.2	5.7.3		5.7.4
NLME-RESET	5.8.2			5.8.3
NLME-SYNC	5.9.2	5.9.3		5.9.4
NLME-GET	5.10.2			5.10.3
NLME-SET	5.11.2			5.11.3

5.1. NLME-NETWORK-DISCOVERY

5.1.1. Description

This primitive instructs the device's network layer to search for networks within connection range. The search operation will populate a list of available networks along with the characteristics of each.

Applicability: All device types.

Prerequisite: NLME-RESET

5.1.2. Request

Description: This function is called to request that the NWK layer discover networks currently operating within range.

Function Prototype:
```
void nlmeNetworkDiscoveryRequest(UINT32 scanChannels,
UINT8 scanDuration)
```

Parameters:

`UINT32 scanChannels`
32 bit long value. The five most significant bits (b27, ... ,b31) are reserved. The 27 least significant bits (b0, b1, ... b26) indicate which channels are to be scanned (1 = scan, 0 = do not scan) for each of the 27 valid channels.

`UINT8 scanDuration`
8 bit long unsigned character. Valid between 0 and 0x0E. A value used to calculate the length of time to spend scanning each channel. The time spent scanning each channel is (aBaseSuperframeDuration * (2n + 1)) symbols, where n is the value of the ScanDuration parameter. Constant aBaseSuperframeDuration is defined in the IEEE 802.15.4 Standard.

5.1.3. Confirm

Description: Confirmation written to `nlmeConfirm` global.

Results: `nlmeConfirm.confirmId = N_DISC_CFM;`
`nlmeConfirm.buffer` structure:

```
typedef struct{
   BYTE NetworkCount;
   NETWORK_DESCRIPTOR nwkDescriptor[
         MAX_USE_CHANNEL_COUNT];
   MAC_ENUM Status;
}NLME_NETWORK_DISCOVERY_CONFIRM;
```
(defined in HS_NET.h)

```
typedef struct {
   WORD  panID;
   BYTE  logicalChannel;
   BYTE  stackProfile;
   BYTE  zigBeeVersion;
   BYTE  beaconOrder;
   BYTE  superFrameOrder;
   BOOL  permitJoining;
   BYTE  securityLevel;
}NETWORK_DESCRIPTOR;
```
(defined in HS_NET.h)

`BYTE NetworkCount`
Number of networks discovered during the search.

`nwkDescriptor`
List of descriptors for each of the `NetworkCount` networks. One entry of type `NETWORK_DESCRIPTOR` for each network found.

`MAC_ENUM Status`
Status after the search.
`SUCCESS`: successful search (minimum 1 network found)
`NO_BEACON`: no beacons detected during active scan
`INVALID_PARAMETER`: unsupported parameter or parameter out of range.

List entry, one per discovered network:

`WORD panID`
The 16-bit PAN identifier of the discovered network. The 2 highest-order bits of this parameter are reserved and shall be set to 0.

`BYTE logicalChannel`
The current logical channel occupied by the network

`BYTE stackProfile`
A ZigBee stack profile identifier indicating the stack profile in use in the discovered network.

`BYTE zigBeeVersion`
The version of the ZigBee protocol in use in the discovered network.

`BYTE beaconOrder`
This specifies how often the MAC sub-layer beacon is to be transmitted by a given device on the network.

`BYTE superFrameOrder`
For beacon-enabled networks, i.e. beacon order < 15, this specifies the length of the active period of the superframe.

`BOOL permitJoining`
Value of `TRUE` indicates that at least one ZigBee router on the network currently permits joining, i.e. its NWK has been issued an NLME-PERMIT-JOINING primitive and the time limit, if given, has not yet expired.

`BYTE securityLevel`
The security level used in a security-enabled PAN. This parameter is not specified in the ZigBee v1.0 specification.

5.2. NLME-NETWORK-FORMATION

5.2.1. Description

This primitive instructs a device to initialize itself as the coordinator of a new ZigBee network.

Applicability: Applies to Coordinator only.

Prerequisite: Device is Coordinator-capable (FFD) and not already established in a network. NLME-RESET should be issued beforehand.

5.2.2. Request

Description: This primitive allows the next higher layer to request that the device start a new ZigBee network with itself as the coordinator.

Function Prototype:
```
NWK_ENUM nlmeNetworkFormationRequest(UINT32 scanChannels, BYTE
scanDuration, BYTE beaconOrder, BYTE superframeOrder, WORD panID,
BOOL batteryLifeExtension) large
```

Parameters:

`UINT32 scanChannels`
The five most significant bits (b27, ... ,b31) are reserved.
The 27 least significant bits (b0, b1, ... b26) indicate which channels are to be scanned (1 = scan, 0 = do not scan) for each of the 27 valid channels.

`UINT8 scanDuration`
A value used to calculate the length of time to spend scanning each channel.

`BYTE beaconOrder`
In star mode or tree mode this specifies the beacon order of the network that the higher layers wish to form. In MESH_MODE there are no beacons and this parameter should be set equal to 0x0F.

`BYTE superframeOrder`
In star mode or tree mode this specifies the superframe order of the network that the higher layers wish to form.
In MESH_MODE there are no beacons and this parameter may be omitted. If the parameter is supplied, it will be ignored.

`WORD panID`
An optional PAN identifier that may be supplied if higher layers wish to establish this network with a predetermined identifier. (0x0000 - 0x3FFF)
If PANId is not specified (i.e. `panID` = `NULL`) the NWK layer will choose a PAN ID.

`BOOL batteryLifeExtension`
If this value is `TRUE`, the NLME will request that the ZigBee coordinator is started supporting battery life extension mode.
If this value is `FALSE`, the NLME will request that the ZigBee coordinator is started without supporting battery life extension mode.

5.2.3. Confirm

Description: Confirmation by return value of nlmeNetworkFormationRequest, type `NWK_ENUM` (See Section “6.1.1. NWK_ENUM” on page 28).

Returned Values: `SUCCESS`:

`INVALID_REQUEST`: Selected device is unable to start as a coordinator.

`STARTUP_FAILURE`: Device is unable to start as coordinator without conflicting with another existing Pan ID or channel assignment.

`NO_SHORT_ADDRESS`:

`UNAVAILABLE_KEY`: Key not found (secure mode)

`FRAME_TOO_LONG`:

`FAILED_SECURITY_CHECK`:

`INVALID_PARAMETER`: Unsupported parameter or parameter out of range.

5.3. NLME-PERMIT-JOINING

5.3.1. Description

This primitive opens a Coordinator or Router to accept other devices to its network.

Applicability: Applies to Coordinator or Routers only.

Prerequisite: Device already started as Coordinator or Router.
NLME-NETWORK-FORMATION (Coordinator), or
NLME-START-ROUTER (Router)

5.3.2. Request

Description: This function allows the next higher layer of a ZigBee coordinator or router to set its MAC sub-layer association permit flag for a fixed period during which it may accept devices onto its network.

Function Prototype: `NWK_ENUM nlmePermitJoiningRequest(BYTE permitDuration)`

Parameters: `BYTE permitDuration`
The length of time in seconds during which the ZigBee coordinator or router will allow associations.
The values 0x00 and 0xff indicate that permission is disabled or enabled, respectively, without a specified time limit.

5.3.3. Confirm

Description: Confirmation by return value of `nmlePermitJoiningRequest`, type `NWK_ENUM` (See Section “6.1.1. NWK_ENUM” on page 28).

Returned Values: `SUCCESS`:

`INVALID_REQUEST`: Occurs if issued to a ZigBee end device.

`UNSUPPORTED_ATTRIBUTE`:

`INVALID_PARAMETER`:

5.4. NLME-START-ROUTER

5.4.1. Description

5.4.2. Request

Description: This function allows the next higher layer of a ZigBee router to initialize or change its superframe configuration. It also allows the next higher layer of a ZigBee coordinator to change its superframe configuration.

Function Prototype: `NWK_ENUM nlmeStartRouterRequest(BYTE beaconOrder, BYTE superframeOrder, BOOL BatteryLifeExtension)`

Parameters:

`BYTE beaconOrder`
In star mode or tree mode this specifies the beacon order of the network that the higher layers wish to form. (0x00–0x0F)
In MESH_MODE there are no beacons and this parameter will be set equal to 0x0F.

`BYTE superframeOrder`
In star mode or tree mode this specifies the superframe order of the network that the higher layers wish to form. (0x00–0x0F)
In MESH_MODE there are no beacons and this parameter may be omitted. If the parameter is supplied, it will be ignored.

`BOOL BatteryLifeExtension`
If this value is `TRUE`, the NLME will request that the ZigBee coordinator is started supporting battery life extension mode.
If this value is `FALSE`, the NLME will request that the ZigBee coordinator is started without supporting battery life extension mode.

5.4.3. Confirm

Description: Confirmation by return value of `nlmeStartRouterRequest`, type `NWK_ENUM` (See Section “6.1.1. NWK_ENUM” on page 28).

Returned Values: `INVALID_REQUEST` or any status value `(SUCCESS, NO_SHORT_ADDRESS, UNAVAILABLE_KEY,FRAME_TOO_LONG, FAILED_SECURITY_CHECK or INVALID_PARAMETER)` returned from the `mlmeStartRequest` function.

5.5. NLME-JOIN

5.5.1. Description

Applicability: Allows Child to join network.

Prerequisites: NLME-RESET must occur before NLME-JOIN

5.5.2. Request

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

Function Prototype: `void nlmeJoinRequest(WORD panId, BOOL joinAsRouter, BOOL rejoinNetwork, UINT32 scanChannels, BYTE scanDuration, BYTE powerSource, BYTE rxOnWhenIdle, BYTE macSecurity) large`

Parameters:

`WORD panId`
The PAN identifier of the network to attempt to join or re-join. (0x0000–0x3FFF). Select from available networks shown in `nwkDescriptor` list from NLME-NETWORK-DISCOVERY request/confirmation.

`BOOL joinAsRouter`
The parameter is TRUE if the device is attempting to join the network in the capacity of a ZigBee router. It is FALSE otherwise.
The parameter is valid in requests to join through association and ignored in requests to join directly or to re-join through orphaning.

`BOOL rejoinNetwork`
`TRUE`: the device is joining directly or rejoining the network using the orphaning procedure.
`FALSE`: the device is requesting to join a network through association.

`UINT32 scanChannels`
The five most significant bits (b27, ... ,b31) are reserved. The 27 least significant bits (b0, b1, ... b26) indicate which channels are to be scanned (1 = scan, 0 = do not scan) for each of the 27 valid channels.

`BYTE scanDuration`
A value used to calculate the length of time to spend scanning each channel.

`BYTE powerSource`
This parameter becomes a part of the `CapabilityInformation` parameter passed to the `mlmeAssociateRequest` function that is generated as the result of a successful executing of a NWK join.
0x01: Mains-powered device,
0x00: other power source.

`BYTE rxOnWhenIdle`
This parameter indicates whether the device can be expected to receive packets over the air during idle portions of the active portion of its superframe.
0x01: The receiver is enabled when the device is idle.
0x00: The receiver may be disabled when the device is idle.

This parameter shall have a value of 0x01 for ZigBee coordinators and ZigBee routers operating in a nonbeacon-oriented network.

`BYTE macSecurity`
This parameter becomes a part of the `capabilityInformation` parameter passed to the mlmeAssociateRequest function that is generated as the result of a successful executing of a NWK join.
0x01: MAC security enabled.
0x00: MAC security disabled.

5.5.3. Indication

Description: This function allows the next higher layer of a ZigBee coordinator or ZigBee router to be notified when a new device has successfully joined its network by association. Indication written to `nlmeConfirm` global.

Results:
```
nlmeConfirm.confirmId = N_JOIN_IND;
nlmeConfirm.buffer structure:

typedef struct {
  WORD      shortAddress;
  ADDRESS   extendedAddress;
  BYTE      capabilityInformation;
}NLME_JOIN_INDICATION;
```
(defined in HS_NET.h)

Parameters:

`WORD shortAddress;`
The network address of an entity that has been added to the network.

`ADDRESS extendedAddress;`
The EUI of the an entity that has been added to the network.

`BYTE capabilityInformation`
Bitwise description of the device.
b0: Alternate PAN coordinator:Always 0 in ZigBee v1.0
b1: Device Type:
1: Joining device is a router, and joining with `joinAsRouter=TRUE`.
0: End device or router joining as an end device.
b2: Power Source: Set to the lowest order bit of the `powerSource` parameter passed to the `nlmeJoinRequest` primitive.
1: mains powered
0: other
b3: Receiver on when idle. Set to the lowest order bit of the `rxOnWhenIdle` parameter passed to the `nlmeJoinRequest` primitive.
1: receiver enabled when device in idle.
0: receiver may be disabled when device is idle.
b4: Reserved. Always 0.
b5: Reserved. Always 0
b6: Security Capability. This field shall be set to the value of lowest-order bit of the `macSecurity` parameter passed to the NLME-JOIN-request primitive.
1: MAC security enabled
0: MAC security disabled
b7: Allocate address: Always 1 in ZigBee v1.0. Always allocate the joining device a 16-bit short address.

5.5.4. Confirm

Description: Confirmation written to `nlmeConfirm` global.

Results:

```
nlmeConfirm.confirmId = N_JOIN_CFM;
nlmeConfirm.buffer structure:

typedef struct{
   WORD PANId;
   NWK_ENUM Status;
}NLME_JOIN_CONFIRM;      (defined in HS_NET.h)
```

Parameters:

`WORD PANId;`
The PAN identifier from the NLME-JOIN.request to which this is a confirmation. The 2 highest-order bits of this parameter are reserved and should be set to 0.

`NWK_ENUM Status`
`INVALID_REQUEST, NOT_PERMITTED` or any status value returned from the MLME-ASSOCIATE.confirm primitive or the MLME-SCAN.confirm primitive `(SUCCESS, CHANNEL_ACCESS_FAILURE, NO_ACK,_NO_DATA, UNAVAILABLE_KEY, FAILED_SECURITY_CHECK`).

5.6. NLME-DIRECT-JOIN

5.6.1. Description

This primitive manually adds a child device to its neighbor table. It does not communicate or handshake with the added child device.

Applicability: Ability to request applies only to Coordinator or Router-type devices.
All end devices are able to accept a direct join request from a parent.

Prerequisites: Device must be a Coordinator or Router to initiate.
Requesting device must know 64-bit address of device to add.
Child device must proactively initiate an `nlmeJoinRequest(rejoinNetwork=TRUE)` to complete re-join.

5.6.2. Request

Function Prototype:
```
void nlmeDirectJoinRequest(ADDRESS deviceAddress,
CAPABILITY_INFORMATION_FIELD capabilityInformation)
```

Parameters:

`ADDRESS deviceAddress`
The IEEE address of the device to be directly joined.

`BYTE capabilityInformation`
The operating capabilities of the device being directly joined. Refer to sections 5.5.3 and 6.2.1.

5.6.3. Confirm

Description: Confirmation written to `nlmeConfirm` global.

Results:
`nlmeConfirm.confirmId = N_DJOIN_CFM;`
`nlmeConfirm.buffer` structure:
```
typedef struct{
   ADDRESS deviceAddress;
   NWK_ENUM status;
}NLME_DIRECT_JOIN_CONFIRM;
```
(defined in HS_NET.h)

Parameters:

`ADDRESS deviceAddress;`
IEEE address of the device joined.

`NWK_ENUM status;`

`SUCCESS:`

`ALREADY_PRESENT:` Device already exists in table.

`TABLE_FULL:` No capacity available for additional devices.

5.7. NLME-LEAVE

5.7.1. Description

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

Applicability: Both child and parent-type devices.

Prerequisites: Device to be disconnected is currently connected to network.

5.7.2. Request

Description: The Function is used to request that it or another device leaves the network.

Function Prototype: `void nlmeLeaveRequest(ADDRESS deviceAddress)`

Parameters: `ADDRESS deviceAddress`
Parent: 64-bit IEEE address of child device to remove from network.
Child: `NULL` to remove itself from network.

5.7.3. Indication

Description: If the device is a child, a leave indication with a null address argument that the device has been forced to disconnect by its parent.
If the devices is a parent, a leave indication shows that a child has proactively removed itself from the network.

Results: Indication written to `nlmeConfirm` global.
`nlmeConfirm.confirmId = N_LEAVE_IND;`
`nlmeConfirm.buffer` structure:

```
typedef struct {
   ADDRESS   extendedAddress;
}NLME_LEAVE_INDICATION;
```

Parameters: `ADDRESS extendedAddress`

`NULL` if this device was removed by a parent device.

`<IEEE address>` if a child device has proactively disassociated itself from this parental device.

5.7.4. Confirm

Description: Confirmation written to `nlmeConfirm` global.

Results: nlmeConfirm.confirmId = N_LEAVE_CFM;

```
nlmeConfirm.buffer structure:
typedef struct{
   ADDRESS deviceAddress;
   NWK_ENUM status;
}NLME_LEAVE_CONFIRM;      (defined in HS_NET.h)
```

Parameters: `ADDRESS deviceAddress`

`NULL` if device removed itself from a parent.

`<IEEE address>` if device is a parent and has removed a child.

`NWK_ENUM status`
`SUCCESS:`

`INVALID_REQUEST`: Device is not in a network.

`UNKNOWN_DEVICE:` Issued if leave request made to a coordinator or router to remove an unknown device.

5.8. NLME-RESET

5.8.1. Description

The function is called to request that the NWK layer performs a reset operation. This operation sets NIB values to defaults, resets the MAC layer, and clears network-level parameters such as discovered routes.

NLME-RESET must be called immediately on power-up.

5.8.2. Request

Function Prototype: `NWK_ENUM nlmeResetRequest(void)`

Parameters: None.

5.8.3. Confirm

Description: Confirmation by return value of `mlmeResetRequest`, type `NWK_ENUM` (See Section "6.1.1. NWK_ENUM" on page 28).

Returned Values: Status value returned from the mlmeResetRequest function.
`SUCCESS:`

`DISABLE_TRX_FAILURE:`

5.9. NLME-SYNC

5.9.1. Description

The NLME-SYNC primitive is used by devices in a network to synchronize to a parent node and to request data from the Coordinator or Router.

In a non-beacon network, this primitive is simply used by a device to request pending data from the PAN coordinator. The `track` parameter should always be set to `FALSE` in non-beacon mode.

In a beacon-based network, this primitive serves multiple functions. First, it directs the device's MAC layer to synchronize to the beacon from its parent. The node will continuously track beacons if the `track` parameter is set to `TRUE`. Second, it instructs the device to automatically send a data request to the PAN coordinator each time a beacon frame is received indicating that data are waiting for the device.

Applicability: Applies to both beacon-based and non-beacon-based networks.
Applies to all devices other than Coordinators.

Prerequisites: Device associated with a network.

5.9.2. Request

Description: The function is called to synchronize or extract data from its ZigBee coordinator or router.

Function Prototype: `NWK_ENUM nlmeSyncRequest(BOOL track)`

Parameters: `BOOL track`
Whether the synchronization should be maintained for future beacons or not.

5.9.3. Indication

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

Indication written to `nlmeConfirm` global. According to the NWK specification, this primitive will be generated only when `nlmeSyncRequest` is called. This primitive will be generated when beacon can not be detected after several beacon periods.

Results: nlmeConfirm.confirmId = `N_SYNC_IND`

5.9.4. Confirm

Description: Confirmation by return value of `mlmeSyncRequest`, type `NWK_ENUM` (See Section “6.1.1. NWK_ENUM” on page 28).

Returned Values: `SUCCESS`:

`SYNC_FAILURE`: If unable to synchronize to a parent's beacon.

`INVALID_PARAMETER`: Occurs when `track = TRUE` on a nonbeacon network.

5.10. NLME-GET

5.10.1. Description

This function allows the application layer to read the value of an attribute from the NIB. Attributes are listed in Section "6.1.2. NWK_NIB_ATTR" on page 28.

5.10.2. Request

Function Prototype: `void nlmeGetRequest(NWK_NIB_ATTR NIBAttribute)`

Parameters: `NWK_NIB_ATTR NIBAttribute`
The identifier of the NIB attribute to read.

5.10.3. Confirm

Description: Confirmation written to `nlmeConfirm` global.

Results: `nlmeConfirm.confirmId = N_GET_CFM;`
`nlmeConfirm.buffer` structure:

```
typedef struct{
   NWK_ENUM      status;
   NWK_NIB_ATTR  NIBAttribute;
   WORD          NIBAttributeLength;
   BYTE          *pNIBAttributeValue;
}NLME_GET_CONFIRM;      (defined in HS_NET.h)
```

Parameters:

`NWK_ENUM status`
`SUCCESS:`
`UNSUPPORTED_ATTRIBUTE:`

`NWK_NIB_ATTR  NIBAttribute`
See attributes, Section 6.1.2.

`WORD NIBAttributeLength`
Length in octets (0x0000 - 0xFFFF)

`BYTE *pNIBAttributeValue`

5.11. NLME-SET

5.11.1. Description

This function allows the application layer to write the value of an attribute from the NIB. Attributes are listed in Section "6.1.2. NWK_NIB_ATTR" on page 28.

5.11.2. Request

Description: This function allows the next higher layer to write the value of an attribute into the NIB.

Function Prototype:
```
void nlmeSetRequest(NWK_NIB_ATTR NIBAttribute, BYTE
NIBAttributeLength, void *pNIBAttributeValue)
```

Parameters:

`NWK_ENUM NIBAttribute`
The identifier of the NIB attribute to be written.

`WORD NIBAttributeLength`
The length, in octets, of the attribute value being set.

`void *pNIBAttributeValue`
Pointer to the value of the NIB attribute that should be written.

5.11.3. Confirm

Description: Confirmation written to `nlmeConfirm` global.

Results: `nlmeConfirm.confirmId = N_SET_CFM;`
`nlmeConfirm.buffer` structure:

```
typedef struct{
   NWK_ENUM        status;
   NWK_NIB_ATTR    NIBAttribute;
}NLME_SET_CONFIRM;
```
(defined in HS_NET.h)

6. Shared Type Definitions, Structures and Defines

6.1. HS_Net.h

6.1.1. NWK_ENUM

```
BYTE NWK_ENUM;

#define SUCCESS                          0x00
#define NWK_INVALID_PARAMETER            0xc1
#define INVALID_REQUEST                  0xc2
#define NOT_PERMITTED                    0xc3
#define STARTUP_FAILURE                  0xc4
#define ALREADY_PRESENT                  0xc5
#define SYNC_FAILURE                     0xc6
#define TABLE_FULL                       0xc7
#define UNKNOWN_DEVICE                   0xc8
#define NWK_UNSUPPORTED_ATTRIBUTE        0xc9
```

6.1.2. NWK_NIB_ATTR

```
typedef enum {
    NWK_BSCN = 0x81,
    NWK_PASSIVE_ACK_TIMEOUT,
    NWK_MAX_BROADCAST_RETRIES,
    NWK_MAX_CHILDREN,
    NWK_MAX_DEPTH,
    NWK_MAX_ROUTERS,
    NWK_NEIGHBOR_TABLE,
    NWK_NETWORK_BROADCAST_DELIVERY_TIME,
    NWK_REPORT_CONSTANT_COST,
    NWK_ROUTE_DISCOVERY_RETRIES_PERMITTED,
    NWK_ROUTE_TABLE,
    NWK_SECURE_ALL_FRAMES,
    NWK_SECURITY_LEVEL,
    NWK_SYM_LINK,
    NWK_CAPABILITY_INFORMATION
} NWK_NIB_ATTR;
```

6.2. mac.h

6.2.1. CAPABILITY_INFORMATION_FIELD

```
typedef struct tag_CAPABILITY_INFORMATION_FIELD
{
  unsigned char   AlternatePANcoordiantor    :1;
  unsigned char   DeviceType                 :1;
  unsigned char   PowerSource                :1;
  unsigned char   ReceiverOnWhenIdle         :1;
  unsigned char   Reserved                   :2;
  unsigned char   SecurityCapability         :1;
  unsigned char   AllocateAddress            :1;
}CAPABILITY_INFORMATION_FIELD;
```

6.2.2. MAC_ENUM

```
typedef BYTE MAC_ENUM;

#define SUCCESS                 0
#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 RX_DEFERRED             0xF5
```

6.3. mac_headers.h

6.3.1. ADDRESS

```
typedef union {
  BYTE Extended[8];
  WORD Short[4];
  }ADDRESS;
```

CONTACT INFORMATION

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735

Internet: www.silabs.com

SILICON LABORATORIES