Airbee-ZNS Lite Version TI2.04

Programmer's Reference Manual

Airbee Wireless, Inc.
9400 Key West Avenue
Rockville, Maryland 20850 USA

REVISION HISTORY

No	Date	Description
V1.0	May 15, 2005	Initial Creation
V2.0	July 05, 2005	Primitives, Data structures added
V3.0	July 06, 2005	Application notes added
V4.0	July 07, 2005	Format Changed
V5.0	July 09, 2005	Updated and reformatted
V6.0	July 11,2005	Format changes
V6.1	July 13, 2005	Incorporate format changes
V6.2	Sep 9, 2005	Flowchart Modified
V6.3	Sep 13, 2005	General edits
V6.4	Sep15, 2005	General edits
V6.5	Sep18, 2005	Changes to Application Development, Stack Architecture, Overview, APIs etc.
V6.6	Sep 23, 2005	Software version change
V6.7	Sep 23, 2005	General Edits

Table of Contents

Table of Figures

ACRONYMS

API	Application Program Interface
ASL	Application Support Layer
DSL	Device Support layer
MAC	Medium Access Control
PAN	Personal Area Network
PHY	Physical Layer
SAP	Service Access Point
APP-SAP	Application SAP
APP- HW-SAP	Application Hardware SAP
NLDE- SAP	Network Layer Data Entity SAP
NLME- SAP	Network Layer Management Entity SAP
MLDE- SAP	MAC Layer Data Entity SAP
MLME- SAP	MAC Layer Management Entity SAP
PD- SAP	PHY Data SAP
PLME- SAP	PHY Layer Management Entity SAP
RF- SAP	Radio Frequency SAP

1 Purpose

The purpose of this document is to provide the developer, comprehensive API information to develop and deploy ZigBee networks using Airbee-ZNS Lite Version_TI2.04.

2 Scope of Airbee-ZNS Lite Programmer's Reference Manual

The Airbee-ZNS Lite API Programmer's Reference Manual enumerates the APIs available for developers of mesh network applications using four TI ZigBee-ready devices. The document further gives an overview of:

1. Airbee Network Stack Architecture
2. Airbee-ZNS Lite software
3. Device types in a network
4. Application development
5. Limitations of the software

This document provides a brief guide for writing custom applications in C using Airbee-ZNS Lite.

3 Airbee Network Stack Architecture

Airbee Network stack Architecture is derived from ZigBee specification with usability-enhanced extensions. The stack architecture of the Airbee Network is shown in the following illustration:

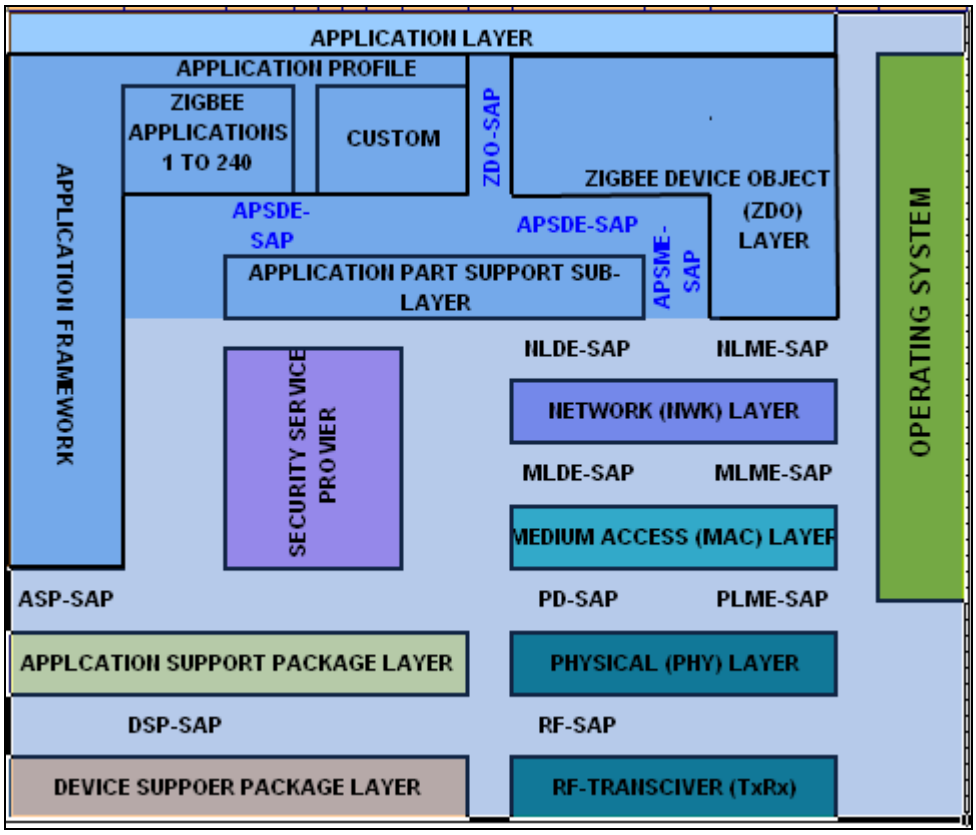


Figure 1. Airbee Stack Architecture

Note: Airbee-ZNS Lite software implements only the Physical (PHY), Medium Access (MAC) and Network (NWK) layers of the ZigBee-ready Airbee stack architecture. So only the PHY, MAC and NWK layers are explained in the forthcoming section.

The software adopts a layered architecture, where each layer performs a specific set of services for the layer above. Service Access Points (SAP) provides the interface between layers. The data services and management services are accessed by the upper layers through the respective SAPs. The SAPs support a number of service primitives to achieve the required functionality.

3.1 Physical (PHY) Layer

The PHY layer mainly consists of a radio transceiver and a micro controller. The PHY layer accepts commands from the MAC layer, transfers the data from radio to MAC and vice versa. The radio can operate in one of the three frequency bands, viz. 2.4 GHz (Global), 915 MHz (Americas) and 868 MHz (Europe). In the frequency band of 2.4 GHz, the radio can operate with sixteen channels. During data reception, PHY layer collects the link quality indication and forwards to upper layers through the MAC layer.

3.2 Medium Access Control (MAC) Layer

The MAC sub-layer provides the interface between Network Layer and the PHY layer. The MAC data and Management services are accessed through their respective service points. The essential features of MAC sub-layer are Beacon Management, Channel access through slotted / Un-slotted CSMA mechanism. In addition, MAC sub-layer provides Frame-validation, Association, Disassociation and reliable data delivery. MAC offers security as an option.

3.3 Network (NWK) Layer

The NWK layer contains mechanisms that are used to associate and disassociate a device from a network and transmit data frames to the intended destinations. In addition, the NWK layer of an Airbee coordinator is responsible for starting a new network and assigning addresses to newly associated devices. The routing is an important function of the coordinator device. Based on the network topology, the PAN Coordinator and Router implement either hierarchical or table based routing.

3.3.1 Operating System

Airbee Operating System (OS) is generic software, specially designed to manage embedded system resources and implement tasks. Some of the advantages the OS presents are:

- Occupies very little memory space
- Optimum support for management and implementation of the tasks
- Supports compile time configuration resulting in minimum use of system resources

4 Airbee-ZNS Lite - Overview

Airbee-ZNS, the ZigBee-ready stack software, saves precious development time while achieving higher return on investment upon deployment of ZigBee networks. Airbee-ZNS has been developed to achieve maximum portability and adaptability so that stack binaries can be made available on any microcontroller, transceiver and operating system platform.

The possibility of developing a full ZigBee stack or parts of it on new or modified hardware platforms offers a phenomenal advantage to ZigBee application developers, ZigBee hardware developers, tool developers, and manufacturers of ZigBee compatible products. This protocol stack versatility also enables custom implementations.

In a typical embedded software design and implementation, the extent of portability is limited by factors such as undefined interaction between the radio and the medium, hardware- dependent optimization needs, the ordering of bytes in a multi-byte packet, and numerous other details with few or no specifications.

Airbee-ZNS Lite software is designed and structured to perform optimally within these limitations. Airbee-ZNS Lite is built to address complete portability across all microcontrollers. Each layer in the ZigBee standard has its own Application Programming Interface (API).

Airbee-ZNS Lite, as the name suggests, is a limited but powerful implementation from the Airbee-ZNS Lite family of ZigBee stack software.

4.1 Scope of Airbee-ZNS Lite Software

Airbee-ZNS Lite implements PHY, MAC and Network layers of the ZigBee-ready Airbee stack architecture. Refer to section 3 for an overview of the Airbee stack architecture

Note: The Application Support Sub-layer (APS), ZigBee Device Objects (ZDO) layer, Application Framework (AF) and Security services are not implemented.

5 Device Types in a Network

A ZigBee network is made up of three types of devices, viz. a PAN Coordinator, Routers and End Devices. Each type of device plays a specific role in the network.

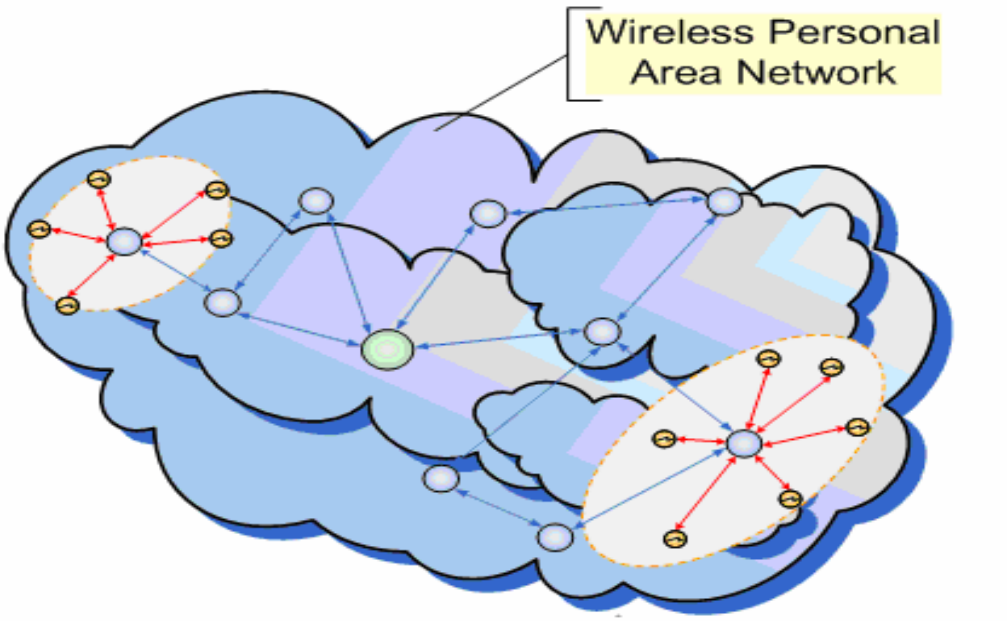


Figure 2. Typical Airbee network

Every device needs to be configured for its assigned role in the network through programmable options. The Airbee Network stack provides a set of parameters for configuring the capability and other network operational characteristics.

The configuration parameters for each of the devices are defined as a part of network initialization. These parameters are programmable, either at the factory or in the field.

5.1 PAN Coordinator

The PAN Coordinator is responsible for forming the network. There can be only ONE PAN Coordinator in a PAN. This device scans and chooses an operational channel within its range of frequencies in the ISM band (Channel 11 to 26). Each Personal Area Network has a unique identification number, viz. PAN Identity (PAN ID). After starting as a PAN Coordinator, it allows other devices to join to form a network. Other devices can join the PAN Coordinator as Routers or End Devices.

5.2 Router

The router plays a vital role in extending the network diameter. Routers allow other routers and end devices to join the network. Routers also route data from other network devices to its destination.

5.3 End device

The End Device is restricted to communicate only with its parent, which could be a Router or a PAN Coordinator. However, it can communicate data to other remote devices through its parent, which could be a Router or a PAN coordinator or another end device in its Personal Area Network.

6 Airbee-ZNS Lite Network Stack Architecture

Airbee-ZNS Lite Network stack provides API for starting and managing the network formation, discovery, data transmission and reception. The sub-sections define the functionality of API. The criteria for invoking the API and the response of the Airbee Network stack are defined in the "When generated" and "On Receipt" section, respectively.

The design of Network stack API is based on the task-based architecture. Each layer in the stack is defined as a separate task. The NWK stack API consists of Application, Network, and MAC combined with PHY and IDLE tasks. The application task is exposed to the application developers to build custom applications.

6.1 Data Types

The commonly used data types in this API document are tabulated below:

Mnemonics	Data Types
UINT8	unsigned char
UINT16	unsigned int
UINT32	unsigned long
REAL	float
UINT64	unsigned long long
BOOL	unsigned char
TRUE	Number Constant
FALSE	Number Constant

6.2 Airbee-ZNS Lite Network API

The Network stack APIs are explained with examples in the following section.

6.2.1 V_ABZB_NWK_Init

Function Prototype

`V_ABZB_NWK_Init (UINT8 DeviceRole, UINT8 StackProfile, BOOL Security)`

When generated:

This API is called by the application task to define the role of the device in the network, type of application profile used in this application, and security in the network. This run-time configuration of the device is initialized based on the set initial values.

- The device can be configured as Coordinator, Router or End Device in a network. In a ZigBee network, there can be only one Coordinator.
- The stack profile value for a network is defined by ZigBee Alliance for each profile.
- The security of the network device can be enabled / disabled during initialization.

On Receipt:

The network stack initializes the dependent parameters of the device.

Parameters:

Element Name	Data Type	Data Size	Value	Return Type	Purpose/Remarks
DeviceRole	UINT8	1	0 to 2	Void	DeviceRole: 0-Coordinator 1- router, 2- end device
StackProfile	UINT8	1	5		StackProfile: Stack profile which is running.
Security	BOOL	1	0 or 1		Security: 1 - MAC security is TRUE 0- MAC security is set to FALSE

Example:

Refer to the API V_ABZB_NWK_Init which is called in the function V_ABZB_APP_COORDINATOR_INIT in the file ABZB_APP_TASK.c

```
File Name – ABZB_APP_TASK.c

Function Name - V_ABZB_APP_COORDINATOR_INIT

void V_ABZB_APP_COORDINATOR_INIT()

{

//NWK init for coordinator

V_ABZB_NWK_Init (COORDINATOR, M_ABZB_STACK_PROFILE, FALSE);

……

……

}
```

6.2.2 V_ABZB_NWK_NLMERESETRequest

Function Prototype

V_ABZB_NWK_NLMERESETRequest()

When generated:

This API is invoked during initialization of the application program. The device either starts a network as PAN Coordinator or joins the network as a Router or an end device, depending on its configuration. All the layers in the stack get initialized with this primitive call based on the defined role.

On Receipt:

AIRBEE Network Stack performs channel scanning, network identification and network formation in the case of a PAN coordinator. In the case of Router or end-device, it performs discovery process and joins a target network.

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
NLMERESETRequest()	void	1	void	All layers in the stack get initialized.

Example:

Refer to the API V_ABZB_NWK_NLMERESETRequest, which is called in the function V_ABZB_APP_NetworkResetRequest in the file ABZB_APP_TASK.c

```
File Name – ABZB_APP_TASK.c

Function Name - V_ABZB_APP_NetworkResetRequest0

void V_ABZB_APP_NetworkResetRequest0()

{

……..

// Sending NLME-RESET REQUEST

V_ABZB_NWK_NLMERESETRequest();

// waiting till the NLME_RESET_CONFIRM is received

while(Msg.MACMsg.MsgId != NLME_RESET_CONFIRM)

// Getting message from the APP mailbox

GET_MESSAGE(APP_MBX, &(Msg.Msg));

// free the message pointer sent by the NWK layer

free(Msg.MACMsg.Message.MsgPtr);

……..

}

……..)
```

6.2.3 V_ABZB_NWK_NLMENETWORKFORMATIONRequest

Function Prototype

V_ABZB_NWK_NLMENETWORKFORMATIONRequest (stNLMENWKFORMATIONRequest * NLME_NWK_Formation_Request)

When generated:

This primitive is generated only by the coordinator of the network. This is the first step in forming a network. If a coordinator operates in the 2.4GHZ frequency, it can scan up to sixteen different channels in the network to select a particular to channel to form the network. Scanning all the channels before starting the network is not mandatory. Any channel can be selected to form the network. For an example, to start a network in the eleventh channel, a value of 2048(2^{11}) is passed in this parameter.

Scan Channels: To scan 11th and 12th Channel, scan channel parameter needs to be 6144 (a sum of 2^{11} and 2^{12}). Channels from 11th to 26th can be scanned by passing appropriate parameters. To scan 11th and 13th channel, the value of 10240 (a sum of 2^{11} and 2^{13}) needs to be used. If a channel is free, the network will be successfully started in the channel.

PanID: A unique value for the network, as an identifier. The valid range is in between 0 to 0x3fff.

Scan Duration: The time duration for scanning each of the channels.

Beacon Order: In the Mesh network, the value is 15.

Super Frame Order: In the Mesh network, the value is 15.

On Receipt:

The network layer scans all the request channels for a given period of scan duration. The network layer analyses the scan results and forms the network in the first available channel with the PANID value passed.

Parameters: The input parameter - stNLMENWKFORMATIONRequest

. Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
NLME_NWK_Formation_Request	Pointer to stNLMENWKFORMATIONRequest *	1	Void	Formation of the network by the Coordinator

stNLMENWKFORMATIONRequest

Element Name	Data Type	Data Size	Purpose/Remarks
ScanChannels	UINT32	4	Each bit is set or cleared depending on the channel to be scanned
PanID	UINT16	2	The unique identifier value for a network The range 0 to 0x3ffff
ScanDuration	UINT8	1	The length of time to spend scanning each channel

Element Name	Data Type	Data Size	Purpose/Remarks
BeaconOrder	UINT8	1	The beacon order of the network. In this case, a default value of 15
SuperframeOrder	UINT8	1	The superframe order of the network. In this case, a default value of 15

Semantics:

```
struct stNLMENWKFORMATIONREQUEST
{
        UINT32          ScanChannels;
        UINT16          PANId;
      UINT8     ScanDuration;
      UINT8     BeaconOrder;
        UINT8  SuperframeOrder;
        BOOL  BatteryLifeExtension;
}
```

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.

Scan Duration – valid values between 0x00 to 0x0e. This value is 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.

Scan Channel - The five most significant bits (b27... b31) are reserved. The 16 least significant bits (b11 ... b26) indicate which channels are to be scanned in preparation for starting a network (1=scan, 0=do not scan) for each of the 16 valid channels

Beacon Order – defines how often the beacon order is transmitted; if Beacon Order = 15, the beacon order is not transmitted

Superframe Order – Superframe order defines the length of the active portion of the beacon

Logical Channel – The current logical channel occupied by the network.

Battery Life Extension - 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 the battery life extension mode.

Example:

Refer to the API V_ABZB_NWK_NLMENETWORKFORMATIONRequest(NLME_NWK_Formation_Request, which is called in the function V_ABZB_APP_NetworkFormationRequest in the file ABZB_APP_TASK.c

File Name – ABZB_APP_TASK.c

```
Function Name - V_ABZB_APP_NetworkFormationRequest0

void V_ABZB_APP_NetworkFormationRequest0()

{

// Create an instant to the tMACMsgDetails

tMACMsgDetails    Msg = {0};

// Create an instance to the stNLMENWKFORMATIONRequest

stNLMENWKFORMATIONRequest *NLME_NWK_Formation_Request;

……

// Allocate the memory to an instance of the stNLMENWKFORMATIONRequest

NLME_NWK_Formation_Request =
(stNLMENWKFORMATIONRequest*)malloc(sizeof(stNLMENWKFORMATIONRequest));

// checking whether the pointer is NULL, before use it.

if(NLME_NWK_Formation_Request != NULL_PTR)

{

// Stuff the values in the member of the instance

// Stuff the scan channels

NLME_NWK_Formation_Request->ScanChannels        = M_ABZB_SCAN_CHANNELS;

// Stuff the PANId

NLME_NWK_Formation_Request->PANId               = M_ABZB_PANID;


// Stuff the ScanDuration

NLME_NWK_Formation_Request->ScanDuration        = M_ABZB_SCAN_DURATION;

// Stuff the BeaconOrder

NLME_NWK_Formation_Request->BeaconOrder         = M_ABZB_BEACONORDER;

// Stuff the SuperframeOrder

NLME_NWK_Formation_Request->SuperframeOrder     = M_ABZB_SUPERFAMEORDER;

// Stuff the BatteryLifeExtension

NLME_NWK_Formation_Request->BatteryLifeExtension = FALSE;

//sending the NLME Formation Request

V_ABZB_NWK_NLMENETWORKFORMATIONRequest(NLME_NWK_Formation_Request);
```

```
//waiting till the NLME_NETWORK_FORMATION_CONFIRM is received

while(Msg.MACMsg.MsgId != NLME_NETWORK_FORMATION_CONFIRM)

//getting message from the APP mailbox

GET_MESSAGE(APP_MBX, &(Msg.Msg));

//free the message pointer sent by the NWK layer

free(Msg.MACMsg.Message.MsgPtr);

}

……

}
```

6.2.4 V_ABZB_NWK_NLMEPERMITJOININGRequest

Function Prototype

V_ABZB_NWK_NLMEPERMITJOININGRequest(stNLMEPERMITJOININGRequest* ABZB_Pst_PermitJoinReq);

When generated:

This primitive can be called by the next higher layer to allow or disallow devices from associating to this device. This can be called inside a ZigBee coordinator or a router.

On Receipt:

This API forms the permit joining request message and sends it to the Network layer mail box.

Parameters:

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_PermitJoinReq	Pointer to stNLMEPERMITJOININGRequest		Void	

stNLMEPERMITJOININGRequest

Element Name	Data Type	Data Size	Purpose/Remarks
PermitDuration	UINT8	1	The length of time in seconds during which the ZigBee PAN Coordinator or Router will allow association

Semantics:

struct stNLMEPERMITJOININGREQUEST

{

```
    UINT8 PermitDuration;
}
```

Permit Duration - 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.

Example:

Refer to the API V_ABZB_NWK_NLMEPERMITJOININGRequest(NLME_Permit_Joining_Request), which is called in the function V_ABZB_APP_Send_NLME_PermitJoin in the file ABZB_APP_TASK.c

File Name – ABZB_APP_TASK.c

Function Name - V_ABZB_APP_Send_NLME_PermitJoin

```
void V_ABZB_APP_Send_NLME_PermitJoin(UINT8 ABZB_Uch_PermitDuration)
{
// Create an instant to the tMACMsgDetails
 tMACMsgDetails    Msg= {0};
//Create an instance to the stNLMEPERMITJOININGRequest
stNLMEPERMITJOININGRequest *NLME_Permit_Joining_Request;
……
……
// Allocate the memory to an instance of the stNLMEPERMITJOININGRequest
NLME_Permit_Joining_Request =
(stNLMEPERMITJOININGRequest*)malloc(sizeof(stNLMEPERMITJOININGRequest));
// checking whether the pointer is NULL, before use it
if(NLME_Permit_Joining_Request != NULL_PTR)
{
// Stuff the values in the member of the instance
// Stuff the PermitDuration
NLME_Permit_Joining_Request->PermitDuration = ABZB_Uch_PermitDuration;
//sending the NLME permit join request to the NWK layer
V_ABZB_NWK_NLMEPERMITJOININGRequest(NLME_Permit_Joining_Request);
//waiting till the permit join confim is received
while(Msg.MACMsg.MsgId != NLME_PERMIT_JOINING_CONFIRM
//getting message from the APP mailbox
GET_MESSAGE(APP_MBX, &(Msg.Msg));
```

```
//free the message pointer sent by the NWL      free(Msg.MACMsg.Message.MsgPtr);...  }

……..

}
```

6.2.5 V_ABZB_NWK_NLMENETWORKDISCOVERYRequest

Function Prototype

V_ABZB_NWK_NLMENETWORKDISCOVERYRequest(stNLMENWKDISCOVERYRequest* ABZB_Pst_DiscoveryReq)

When generated:

This API is used by the next higher layer on the device to discover any other PANs currently operating in the vicinity or Personal operating space.

On Receipt:

The discovery request is processed and forwarded to the network mail box.

Parameters:

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_DiscoveryReq	Pointer to stNLMENWKDISCOVERYRequest		Void	

stNLMENWKDISCOVERYRequest

Element Name	Data Type	Data Size	Purpose/Remarks
Scan Channels	UINT32	2	Each bit is set or cleared depending on the channel to be scanned
ScanDuration	UINT8	1	A value used to calculate the length of time to spend scanning each channel

Semantics:

Struct stNLMENWKDISCOVERYRequest

{

UNIT 32 Scan Channels;

UNIT 8 ScanDuration;

}

Example:

Refer to the API V_ABZB_NWK_NLMENETWORKDISCOVERYRequest(NLME_NWK_DISCOVERY_Request, which is called in the function V_ABZB_APP_NetworkDiscoveryRequest in the file ABZB_APP_TASK.c

File Name – ABZB_APP_TASK.c

Function Name - V_ABZB_APP_NetworkDiscoveryRequest0

```
 void V_ABZB_APP_NetworkDiscoveryRequest0()
{
……
// Create an instant to the tMACMsgDetails
tMACMsgDetails    Msg ={0};
//Create an instance to the stNLMENWKDISCOVERYRequest
stNLMENWKDISCOVERYRequest *NLME_NWK_DISCOVERY_Request
……
// Allocate the memory to an instance of the stNLMENWKDISCOVERYRequest
NLME_NWK_DISCOVERY_Request =
(stNLMENWKDISCOVERYRequest*)malloc(sizeof(stNLMENWKDISCOVERYRequest));
// checking whether the pointer is NULL, before use it.
if(NLME_NWK_DISCOVERY_Request != NULL_PTR)
{
     // Stuff the values in the member of the instance
// Stuff the scan channels
NLME_NWK_DISCOVERY_Request->ScanChannels = M_ABZB_SCAN_CHANNELS;
// Stuff the scan duration
NLME_NWK_DISCOVERY_Request->ScanDuration = M_ABZB_SCAN_DURATION;
     // sending the NLME Discovery request
V_ABZB_NWK_NLMENETWORKDISCOVERYRequest(NLME_NWK_DISCOVERY_Request);
// waiting till the Network discovery confirm is received
while(Msg.MACMsg.MsgId != NLME_NETWORK_DISCOVERY_CONFIRM)
// getting the message from the APP mailbox
GET_MESSAGE(APP_MBX, &(Msg.Msg));
// free the message pointer sent by the NWK layer
free(Msg.MACMsg.Message.MsgPtr);
}
……}
```

6.2.6 V_ABZB_NWK_NLMEJOINRequest

Function Prototype

V_ABZB_NWK_NLMEJOINRequest(stNLMEJOINRequest * ABZB_Pst_JoinReq)

When Generated

The next higher layer of a device generates this primitive to request to join a new network. The new device can join a new network directly using orphaning procedure or to locate and re-join a network after being orphaned. It validates the input parameters and sends orphan scan request or associate request to the MAC layer depending on the rejoin parameter set to FALSE/TRUE.

On Receipt

This API forms the join Request message and sends it to the Network layer mail box.

Parameters: The return type for this API is void.

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_JoinReq	Pointer to stNLMEJOINRequest		Void	

stNLMEJOINRequest

Element Name	Data Type	Data Size	Purpose/Remarks
PANId	UINT8	1	The PAN identifier of the network to attempt to join or rejoin
JoinAsRouter	BOOL	1	The parameter is TRUE/FALSE depending on the device wants to join as a Router or not
RejoinNetwork	BOOL	1	The parameter is TRUE/FALSE depending on the device whether it is rejoining the network or not (TRUE-Rejoin, FALSE-does not join)
ScanChannels	UINT32	4	Each bit is set or cleared depending on the channel to be scanned
ScanDuration	UINT8	1	The length of time to spend scanning each channel
PowerSource	UINT8	1	This is the parameter to a part of MAC layer association request
RxOnWhenIdle	UINT8	1	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
MACSecurity	UINT8	1	This parameter becomes a part of the Capability Information parameter passed to the MLME-ASSOCIATE.request primitive

Semantics:

```
struct stNLMEJOINREQUEST

{
```

```
    UINT32      ScanChannels;
    UINT16      PANId;
    UINT8       ScanDuration;
    UINT8       PowerSource;
    UINT8       RxOnWhenIdle;
    UINT8       MACSecurity;
    BOOL        JoinAsRouter;
    BOOL        RejoinNetwork;
}
```

PANId – The 16-bit PAN identifier of the discovered network. The 2 highest-order bits of this parameter are reserved and shall be set as 0.

Scan Duration – valid values between 0x00 to 0x0e. This value is 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.

Scan Channel - The five most significant bits (b27 ... b31) are reserved. The 16 least significant bits (b11 ... b26) indicate which channels are to be scanned in preparation for starting a network (1=scan, 0=do not scan) for each of the 16 valid channels

Beacon Order – defines how often the beacon order is transmitted; if Beacon Order = 15, the beacon order is not transmitted

Superframe Order – Super frame order defines the length of the active portion of the beacon

Logical Channel – The current logical channel occupied by the network.

CapabilityInformation parameter passed to the MLME-ASSOCIATE.request primitive that is generated as the result of a successful executing of a NWK join. The values are:

—0x01 = Mains-powered device.—0x00 = other power source.

RxOnWhenIdle - This parameter indicates whether the device can be expected to receive packets over the air during idle portions of the active portion of the CAPa. The values are: 0x01 = the receiver is enabled when the device is idle. 0x00 = the receiver may be disabled when the device is idle. RxOnWhenIdle shall have a value of 0x01 for ZigBee coordinators and ZigBee routers operating in a non-beacon-oriented network.

RejoinNetwork - The parameter is TRUE if the device is joining directly or rejoining the network using the orphaning procedure. The parameter is FALSE if the device is requesting to join a network through association.

MACSecurity - This parameter becomes a part of the CapabilityInformation parameter passed to the MLME-ASSOCI-ATE.request primitive that is generated as the result of a successful executing of a NWK join. The values are: 0x01 = MAC security enabled. 0x00 = MAC security disabled.

Example

Refer to the API V V_ABZB_NWK_NLMEJOINRequest(NLME_Join_Request);, which is called in the

function V_ABZB_APP_NetworkJoinRequest in the file ABZB_APP_TASK.c

```
File Name – ABZB_APP_TASK.c
Function Name - V_ABZB_APP_NetworkJoinRequest0
void V_ABZB_APP_NetworkJoinRequest0()
{
// Create an instant to the tMACMsgDetails
tMACMsgDetails    Msg ={0};
//Create an instance to the stNLMEJOINRequest
stNLMEJOINRequest *NLME_Join_Request
……
……
// Allocate the memory to an instance of the stNLMEJOINRequest
NLME_Join_Request = (stNLMEJOINRequest *)malloc(sizeof(stNLMEJOINRequest ));

// checking whether the pointer is NULL, before use it.
if(NLME_Join_Request != NULL_PTR)
{
    // Stuff the values in the member of the instance
     // Stuff the ScanChannels
NLME_Join_Request->ScanChannels   = M_ABZB_SCAN_CHANNELS;
     // Stuff the PANId
     NLME_Join_Request->PANId         = M_ABZB_PANID;
     // Stuff the ScanDuration
     NLME_Join_Request->ScanDuration   = M_ABZB_SCAN_DURATION
     // Stuff the PowerSource
     NLME_Join_Request->PowerSource    = ZERO;
     // Stuff the PowerSource
     NLME_Join_Request->RxOnWhenIdle   = TRUE;
     // Stuff the MACSecurity
     NLME_Join_Request->MACSecurity    = ZERO;
```

```
    // Stuff the JoinAsRouter
    NLME_Join_Request->JoinAsRouter   = TRUE;
    // Stuff the RejoinNetwork
    NLME_Join_Request->RejoinNetwork  = FALSE;
    //Sending the NLME Join Request
V_ABZB_NWK_NLMEJOINRequest(NLME_Join_Request);
    //waiting till the NLME_JOIN_CONFIRM is received
    while(Msg.MACMsg.MsgId != NLME_JOIN_CONFIRM)
    //Getting the message from the APP mailbox
    GET_MESSAGE(APP_MBX, &(Msg.Msg));
    //free the message pointer sent by the NWK layer
    free(Msg.MACMsg.Message.MsgPtr);
}
……
……
}}
```

6.2.7 V_ABZB_NWK_NLMESTARTROUTERRequest

Function Prototype

V_ABZB_NWK_NLMESTARTROUTERRequest(stNLMESTARTROUTERRequest * ABZB_Pst_StartRouterReq);

When generated:

This API is called by the next higher layer to request the network layer to request for the initialization of itself as a router. If this API is not called, the device will function as an End device or an RFD.

On Receipt:

This API forms the start router request and sends it to the MAC layer. Further if the device role is neither that of a coordinator nor a router, error is sent back to the layer which calls this API.

Parameters:

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_StartRouterReq	Pointer to stNLMESTARTROUTERRequest		Void	

stNLMESTARTROUTERRequest

Element Name	Data Type	Data Size	Purpose/Remarks
BeaconOrder	UINT8	1	The beacon order of the network that the higher layers wish to form
SuperFrameOrder	UINT8	1	The superframe order of the network that the higher layers wish to form
BatteryLifeExtension	BOOL	1	Value of BatteryLifeExtension mode of the ZigBee PAN Coordinator (TRUE/FALSE)

Semantics:

```
struct stNLMESTARTROUTERREQUEST
{
        UINT8 BeaconOrder;
        UINT8 SuperFrameOrder;
        BOOL  BatteryLifeExtension;
}
```

Beacon Order – defines how often the beacon order is transmitted; if Beacon Order = 15, the beacon order is not transmitted

Superframe Order – Superframe order defines the length of the active portion of the beacon

Example:

Refer to the API V_ABZB_NWK_NLMESTARTROUTERRequest(NLME_Start_Router_Request);, which is called in the function V_ABZB_APP_Network_StartRouterRequest0in the file ABZB_APP_TASK.c

File Name – ABZB_APP_TASK.c

Function Name - V_ABZB_APP_Network_StartRouterRequest0

```
void V_ABZB_APP_Network_StartRouterRequest0(void)
{
// Create an instant to the tMACMsgDetails
tMACMsgDetails    Msg= {0};
//Create an instance to the stNLMESTARTROUTERRequest
stNLMESTARTROUTERRequest *NLME_Start_Router_Request;
……
……
```

```
// Allocate the memory to an instance of the stNLMESTARTROUTERRequest

NLME_Start_Router_Request =
(stNLMESTARTROUTERRequest*)malloc(sizeof(stNLMESTARTROUTERRequest));

// checking whether the pointer is NULL, before use it

if(NLME_Start_Router_Request != NULL_PTR)

{

        // Stuff the values in the member of the instance

// Stuff the BeaconOrder

NLME_Start_Router_Request->BeaconOrder         = M_ABZB_BEACONORDER;

// Stuff the SuperFrameOrder

NLME_Start_Router_Request->SuperFrameOrder     = M_ABZB_SUPERFAMEORDER;

// Stuff the BatteryLifeExtension

NLME_Start_Router_Request->BatteryLifeExtension = FALSE;

//Sending the Start router request
V_ABZB_NWK_NLMESTARTROUTERRequest(NLME_Start_Router_Request);

//waiting till the START ROUTER CONFIRM IS RECEIVED

while(Msg.MACMsg.MsgId != NLME_START_ROUTER_CONFIRM)

//getting the message from the APP mailbox

GET_MESSAGE(APP_MBX, &(Msg.Msg));

//free the message pointer sent by the NWK layer

free(Msg.MACMsg.Message.MsgPtr);

}

……}
```

6.2.8 V_ABZB_NWK_NLDEDATARequest

Function Prototype

V_ABZB_NWK_NLDEDATARequest(stNLDEDATARequest * ABZB_Pst_DataReq)

When Generated

This API is called whenever the upper layer wants to send data packet to its peer device.

On Receipt:

The NSDU packet is formed by this API and sent to the MAC layer to be sent further over the air. If the destination address is 0xffff, the packet is broadcast. Otherwise the routing is initiated to get the

next hop address where the packet needs to be sent.

Parameters:

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_DataReq	Pointer to stNLDEDATARequest		void	

stNLDEDATARequest

Element Name	Data Type	Data Size	Purpose/Remarks
DstAddr	Network Address	2	The network address of the entity or entities to which the NSDU is being transferred
NsduLength	Integer	Variable <= nwkcMaxPayloadSize	The number of octets comprising the NSDU to be transferred
Nsdu	SetofOctets	Variable	The set of octets comprising the NSDU to be transferred
NsduHandle	UINT8	1	The handle associated with the NSDU to be transmitted by the NWK layer entity
BroadcastRadius	UINT8	1	The distance, in hops, that a broadcast frame will be allowed to travel through the network.
DiscoverRoute	UINT8	1	TRUE = enable route discovery FALSE = disable route discovery
SecurityEnable	UINT8	1	The SecurityEnable parameter may be used to enable NWK layer security processing for the current frame

Semantics:

```
struct stNLDEDATAREQUEST
{
        UINT16          DstAddr;
        UINT8 NsduLength;
        UINT8 *Nsdu;
        UINT8 NsduHandle;
        UINT8 BroadcastRadius;
        UINT8 DiscoverRoute;
        BOOL  SecurityEnable;
}
```

Dst Address - The network address of the entity or entities to which the NSDU is being transferred.

NSDU Length - The number of octets comprising the NSDU to be transferred.

NSDU Handle - The handle associated with the NSDU to be transmitted by the NWK layer entity.

Broadcast Radius - The distance, in hops, that a frame will be allowed to travel through the network.

Discover Route - The Discover Route parameter may be used to control route discovery operations for the transit of this frame.

0x00 = suppress route discovery.

0x01 = enable route discovery.

0x02 = force route discovery.

Security Enable - The Security Enable parameter may be used to enable NWK layer security processing for the current frame. If the security level specified in the NIB is 0, meaning no security, then this parameter will be ignored. Otherwise, a value of TRUE denotes that the security processing specified by the security level will be applied and a value of FALSE denotes that no security processing will be applied.

Example:

Refer to the API V_ABZB_NWK_NLDEDATARequest(ABZB_Pst_DataReq, which is called in the function V_ABZB_APP_Send_NLDEDataRequest in the file ABZB_APP_TASK.c

```
File Name – ABZB_APP_TASK.c

Function Name - V_ABZB_APP_Send_NLDEDataRequest

void V_ABZB_APP_Send_NLDEDataRequest()

{

//Create an instance to the stNLDEDATARequest

stNLDEDATARequest *ABZB_Pst_DataReq;

……

// Allocate the memory to an instance of the stNLDEDATARequest

ABZB_Pst_DataReq = (stNLDEDATARequest *) malloc(sizeof(stNLDEDATARequest ));

// checking whether the pointer is NULL, before use it

if(ABZB_Pst_DataReq != NULL_PTR)

{

// Stuff the values in the member of the instance


// Stuff the DstAddr

ABZB_Pst_DataReq-> DstAddr        = Ui_ABZB_ZADKV2_GetDstAddress();
```

```
 // Stuff the NsduLength
ABZB_Pst_DataReq-> NsduLength      = M_ABZB_APP_DATA_NSDULENGTH;
// Allocate the memory for NSDU
ABZB_Pst_DataReq-> Nsdu            =(unsigned char *)malloc(sizeof(UINT8));
// checking whether the pointer is NULL, before use it
if(ABZB_Pst_DataReq-> Nsdu != NULL_PTR)
{
          // Stuff the Nsdu
*(ABZB_Pst_DataReq-> Nsdu)         = M_ABZB_PROTOCOL_ID_APP_DATA;
// Stuff the NsduHandle
ABZB_Pst_DataReq-> NsduHandle      = M_ABZB_APP_DATA_NSDUHANDLE;
// Stuff the BroadcastRadius
ABZB_Pst_DataReq-> BroadcastRadius = ZERO;
// Stuff the DiscoverRoute
ABZB_Pst_DataReq-> DiscoverRoute   = ZERO;
// Stuff the SecurityEnable
ABZB_Pst_DataReq-> SecurityEnable  = FALSE;
//sending the NLDE DATA REQUEST after framing the packet
V_ABZB_NWK_NLDEDATARequest(ABZB_Pst_DataReq);
}
}
……});
```

6.2.9 V_ABZB_NWK_NLMERESETConfirm

Function Prototype

V_ABZB_NWK_NLMERESETConfirm(stNLMERESETConfirm *ABZB_Pst_ResetConfirm)

When Generated

This is generated in response to a prior request to the network by the next higher layer (APS).

On Receipt

On receipt of this primitive, the next higher layer is notified of the results of its request to reset the NWK layer. If the request was successful, then the status would reflect a success in re-setting the network layer. Otherwise, a FAILURE will be sent to the next higher layer (APS).

Parameters

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_ResetConfirm	Pointer to stNLMERESETConfirm		Void	

stNLMERESETConfirm

Element Name	Data Type	Data Size	Purpose/Remarks
NetworkCount	UINT8	1	Gives the number of networks
NetworkDescriptor	StNetworkDescriptor	-	A list of descriptors, one for each of the networks discovered
Status	UINT8	1	Status of the operation

Semantics:

```
struct stNLMERESETCONFIRM
{
        UINT8 Status;
}
```

Status - The result of the reset operation

(Refer 6.2.10 for typical example)

6.2.10 V_ABZB_NWK_NLMENETWORKFORMATIONConfirm

Function Prototype

V_ABZB_NWK_NLMENETWORKFORMATIONConfirm (stNLMENWKFORMATIONConfirm * ABZB_Pst_NetFormationConfirm)

When Generated

This primitive is generated by the network layer and issued to its next higher layer (APS) after receiving the status of network formation request from the MAC layer.

On Receipt

This primitive will inform the next higher layer, the results of a prior network formation request from the next higher layer. If the formation is successful, the 'SUCCESS' status is sent to the APS layer.

Parameters

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_NetFormationConfirm	Pointer to stNLMENWKFORMATIONConfirm		void	

stNLMENWKFORMATIONConfirm

Element Name	Data Type	Data Size	Purpose/Remarks
Status	UINT8	1	Status of the operation

Semantics:

```
struct stNLMENWKFORMATIONCONFIRM

{

        UINT8 Status;

}
```

Status - The result of the attempt to initialize a ZigBee coordinator or request a change to the super frame configuration

Example

Refer to the API V_ABZB_NWK_NLMENETWORKFORMATIONRequest(NLME_NWK_Formation_Request, which is called in the function V_ABZB_APP_NetworkFormationRequest0 in the file ABZB_APP_TASK.c

File Name – ABZB_APP_TASK.c

Function Name - V_ABZB_APP_NetworkFormationRequest0

```
void V_ABZB_APP_NetworkFormationRequest0()

{

tMACMsgDetails    Msg = {0};

        ……

        ……

        ……

//sending the NLME Formation Request
V_ABZB_NWK_NLMENETWORKFORMATIONRequest(NLME_NWK_Formation_Request);

        ……
```

```
        ……….

        ……….


        //This is the typical usage for confirmation.


//waiting till the NLME_NETWORK_FORMATION_CONFIRM is received  while(Msg.MACMsg.MsgId != NLME_NETWORK_FORMATION_CONFIRM)

//getting message from the APP mailbox

GET_MESSAGE(APP_MBX, &(Msg.Msg));

……….

        ……….

        ……….

}
```

6.2.11 V_ABZB_NWK_NLMEPERMITJOININGConfirm

Function Prototype

V_ABZB_NWK_NLMEPERMITJOININGConfirm (stNLMEPERMITJOININGConfirm * ABZB_Pst_PermitJoinConfirm)

When Generated

This is generated in response to the Permit joining request issued by a router or a ZigBee coordinator. This is generated after receiving the confirmation from the MAC layer.

On Receipt

When this primitive is received, the network layer notifies the next higher layer (APS) of the results of the permit joining request.

Parameters

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_PermitJoinConfirm	Pointer to stNLMEPERMITJOININGConfirm		void	

stNLMEPERMITJOININGConfirm

Element Name	Data Type	Data Size	Purpose/Remarks
Status	UINT8	1	Status of the operation

Semantics:

struct stNLMEPERMITJOININGCONFIRM

{

UINT8 Status;

}

Status - The status of the corresponding request

(Refer 6.2.10 for typical example)

6.2.12 V_ABZB_NWK_NLMENETWORKDISCOVERYConfirm

Function Prototype

V_ABZB_NWK_NLMENETWORKDISCOVERYConfirm(stNLMENWKDISCOVERYConfirm * ABZB_Pst_DiscoveryConfirm)

When Generated

This API is called whenever the network discovery task initiated by the network layer is completed.

On Receipt

This API shall update the next higher layer with the results of the network discovery. This API sends the number of networks discovered and descriptors for each of the networks discovered to the next higher layer (APS).

Parameters

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_DiscoveryConfirm	Pointer to stNLMENWKDISCOVERYConfirm		void	

stNLMENWKDISCOVERYConfirm

Element Name	Data Type	Data Size	Purpose/Remarks
NetworkCount	UINT8	1	Gives the number of networks
NetworkDescriptor	StNetworkDescriptor	-	A list of descriptors, one for each of the networks discovered
Status	UINT8	1	Status of the operation

Semantics:

```
struct stNLMENWKDISCOVERYCONFIRM
{
        stNetworkDescriptorList NetworkDescriptorList;
        UINT8                   NetworkCount;
        UINT8                   Status;
}
```

Network Descriptor List – A list of descriptors, one for each of the networks discovered. Table 103 gives a detailed account of the contents of each item.

Network Count – Gives the number of networks discovered by the search.

Status - Status value returned with the MLME-SCAN.confirm primitive

(Refer 6.2.10 for typical example)

6.2.13 V_ABZB_NWK_NLMEJOINConfirm

Function Prototype

V_ABZB_NWK_NLMEJOINConfirm(stNLMEJOINConfirm * ABZB_Pst_JoinConfirm)

When Generated

This primitive is generated by the Network layer when a response is received from the MAC layer for a prior join request issued by the network layer. If the prior request is successful, the same is sent to the next higher layer.

On Receipt

On receipt of this, the network layer extracts the status given by the MAC layer and the same is sent to the next higher layer (APS) on the initiating device.

Parameters

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_JoinConfirm	Pointer to stNLMEJOINConfirm		void	

stNLMEJOINConfirm

Element Name	Data Type	Data Size	Purpose/Remarks
PANId	UINT16	2	The PAN identifier from the NLME-JOIN.request to which this is a confirmation
Status	UINT8	1	Status of the operation

Semantics:

struct stNLMEJOINCONFIRM

{

UINT16 PANId;

UINT8 Status;

}

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.

Status - The status of the corresponding request

(Refer 6.2.10 for typical example)

6.2.14 V_ABZB_NWK_NLMEJOINIndication

Function Prototype

V_ABZB_NWK_NLMEJOINIndication(stNLMEJOINIndication * ABZB_Pst_JoinIndication)

When Generated

This primitive is issued by the network layer to the next higher layer after successfully adding a device to the network.

On Receipt

On receipt of this primitive, network layer informs the next higher layer (APS) that a device has joined the network by association.

Parameters

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_JoinIndication	Pointer to stNLMEJOINIndication		Void	

stNLMEJOINIndication

Element Name	Data Type	Data Size	Purpose/Remarks
ShortAddress	UINT16	2	The network address of an entity that has been added to the network
ExtendedAddress	UINT64	8	The 64-bit IEEE address of an entity that has been added to the network
CapabilityInformation	UINT8	1	Specifies the operational capabilities of the joining device

Semantics:

```
struct stNLMEJOININDICATION
{
        UINT64          ExtendedAddress;
        UINT16          ShortAddress;
        UINT8           CapabilityInformation;
BOOL            SecureJoin;
}
```

Extended Address – The 64-bit IEEE address of an entity that has been added to the network

Short Address – The network address of an entity that has been added to the network.

Capability Information – Specifies the operational capabilities of the joining device.

Secure Join - This parameter will be TRUE if the underlying MAC association was performed in a secure manner and FALSE otherwise

6.2.15 V_ABZB_NWK_NLMESTARTROUTERConfirm

Function Prototype

V_ABZB_NWK_NLMESTARTROUTERConfirm(stNLMESTARTROUTERConfirm * ABZB_Pst_StartRouterConfirm)

When Generated

This primitive is issued when the network layer receives a response to an earlier START ROUTER request by the network layer.

On Receipt

When the network layer receives this confirm primitive, it extracts the status from the MAC start confirm primitive (that is issued by the MAC to the network layer) and sends that status to the higher layer (APS).

Parameters

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_StartRouterConfirm	Pointer to stNLMESTARTROUTERConfirm		Void	

stNLMESTARTROUTERConfirm

Element Name	Data Type	Data Size	Purpose/Remarks
Status	UINT8	1	Status of the operation

Semantics:

Struct stNLMESTARTROUTERCONFIRM

{

UINT8 Status;

}

Status - The result of the attempt to initialize a ZigBee router

6.2.16 V_ABZB_NWK_NLDEDATAConfirm

Function Prototype

V_ABZB_NWK_NLDEDATAConfirm(stNLDEDATAConfirm *ABZB_Pst_DataConfirm)

When Generated

This API is invoked by the local network layer and sent to the next upper layer after the NLDE Data request is processed.

On Receipt

On receipt of this primitive the status of the data transmission is conveyed to the next upper layer. If the data has been successfully sent then the SUCCESS would be returned to the next higher layer else FAILURE is sent to the next upper layer.

Parameters

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_DataConfirm	Pointer to stNLDEDATAConfirm		void	

stNLDEDATAConfirm

Element Name	Data Type	Data Size	Purpose/Remarks
NsduHandle	Integer	1	The handle associated with the NSDU being confirmed
Status	Status	1	The status of the corresponding request

Semantics:

struct stNLDEDATACONFIRM

{

UINT8 NsduHandle;

```
    UINT8 Status;
}
```

NSDU Handle The handle associated with the NSDU being confirmed

Status - The status of the corresponding request

6.2.17 V_ABZB_NWK_NLDEDATAIndication

Function Prototype

V_ABZB_NWK_NLDEDATAIndication(stNLDEDATAIndication *ABZB_Pst_DataIndication)

When Generated

This API is called by the Network layer to inform the next higher layer (APS) layer that data for this device has been received. Any data that is not meant for this device is rejected.

On Receipt

The upper layer is informed about the data received at this device along with the sender's address.

Parameters

Element Name	Data Type	Data Size	Return Type	Purpose/Remarks
ABZB_Pst_DataIndication	Pointer to stNLDEDATAIndication		void	

stNLDEDATAIndication

Element Name	Data Type	Data Size	Purpose/Remarks
SrcAddress	UINT16	2	The individual device address from which the NSDU originated
NsduLength	UINT16	1	The number of octets comprising the NSDU being indicated
Nsdu	Set of octets	Variable	The set of octets comprising the NSDU being indicated
LinkQuality	UINT8	1	The link quality indication delivered by the MAC on receipt of this frame as a parameter of the MCPSDATA

Semantics:

```
struct stNLDEDATAINDICATION
{
    UINT16      SrcAddress;
    UINT8       NsduLength;
    UINT8       *Nsdu;
```

```
        UINT8           LinkQuality;
}
```

Src Address – The individual device address from which the NSDU originated

NsduLength – The number of octets comprising the NSDU being indicated.

Nsdu – The set of octets comprising the NSDU being indicated.

Link Quality - The link quality indication delivered by the MAC on receipt of this frame as a parameter of the MCPS-DATA.Indication primtitive

Example

Refer to the API V_ABZB_NWK_NLDEDATARequest(ABZB_Pst_DataReq, which is called in the function V_ABZB_APP_Forever_Loop in the file ABZB_APP_TASK.c

File Name – ABZB_APP_TASK.c

Function Name - V_ABZB_APP_Forever_Loop

```
void V_ABZB_APP_Forever_Loop()
{
//structure variable which will hold the MSG id and pointer from the kernel
tMACMsgDetails    Msg={0};

//This is the typical usage for Indication.

while(TRUE)
        {
                //getting message from the Application mailbox
GET_MESSAGE(APP_MBX, &(Msg.Msg));
         ……….
        ……….
                //switches to a case depending on the message ID
        switch(Msg.MACMsg.MsgId)
                {
          ……….
                 ……….
        ……….
```

```
                    //Indicates that there is data indication from the NWK layer
                    case NLDE_DATA_INDICATION:
                    ……….
          ……….
          ……….
}
}
}
```

6.2.18 Software Timer

Function Prototype

V_ABZB_Timer_Soft_set_timer3(UINT8 ABZB_Uch_TimerNo, UINT32 ABZB_Ul_Time, APP_MBX)

When Generated

This API is called by the Application to post a message to the Application Mail Box, either on occurrence of an event or at periodic intervals.

On Receipt

Application will post a message to the Application Mail Box (APP_MBX) after the lapse of a preset time (ABZB_Ul_Time). The message will carry the timer number (ABZB_Uch_TimerNo).

Element Name	Data Type	Data Size	Purpose/Remarks
User defined	UINT8	1	The application posts a message at the expiry of preset time with the timer identity as 0xE0 or 0xE1 or 0xE2
User defined	UINT32	1	Preset time
User defined	S_Mail_Box		Posting to APP_ MBX

Example

Refer to the API V_ABZB_NWK_NLDEDATARequest(ABZB_Pst_DataReq, which is called in the function V_ABZB_APP_Send_NLDEDataRequest in the file ABZB_APP_TASK.c

```
File Name – ABZB_ASP_TI_SRCc

Function Name - PORT1_SRC


#pragma vector=PORT1_VECTOR
__interrupt void PORT1_SRC (void)
{
```

```
……….

        ……….

        ……….
// LOADING THE TIMER FOR DEBOUNCE
V_ABZB_Timer_Soft_set_timer3( M_DEBOUNCE_EVENT,    M_ABZB_DSP_KEYDEBOUNCE,
APP_MBX );

……….

        ……….

        ……….
}
```

Handle the Message when Timer expires

File Name – ABZB_APP_TASK.c

Function Name - V_ABZB_APP_Forever_Loop

```
void V_ABZB_APP_Forever_Loop()
{
//structure variable which will hold the MSG id and pointer from the kernel
tMACMsgDetailsMsg={0};
//This is the typical usage for Handle Timer Expiration.
while(TRUE)
        {
                //getting message from the Application mailbox
GET_MESSAGE(APP_MBX, &(Msg.Msg));
……….
        //switches to a case depending on the message ID
        switch(Msg.MACMsg.MsgId)
                {
……….
```

```
        ……….

        ……….

//the message is posted on mailbox when a keypress is occurred hence //construct a data packet and send it TI NWK

case M_DEBOUNCE_EVENT:
        V_ABZB_APP_Send_NLDEDataRequest();

                        break;

……….

        ……….

        ……….

        }

}}
```

7 Summary of AIRBEE-ZNS Lite NETWORK API

7.1 Stack APIs

S.No	Primitive Name	Parameter	Type	Return type	Applicable for	
					Coordinator	Router
1	V_ABZB_NWK_Init	DeviceRole StackProfile Security	UINT8 UINT8 BOOL	Void	Y	Y
2	V_ABZB_NWK_NLMERESETRequest	NA	NA	Void	Y	Y
3	V_ABZB_NWK_NLMENETWORKFORMATIONRequest	ABZB_Pst_NetFormationReq	Pointer to stNLMENWKFORMATIONRequest	Void	Y	N
4	V_ABZB_NWK_NLMEPERMITJOININGRequest	ABZB_Pst_PermitJoinReq	Pointer to stNLMEPERMITJOININGRequest	Void	Y	Y
5	V_ABZB_NWK_NLMENETWORKDISCOVERYRequest	ABZB_Pst_DiscoveryReq	Pointer to stNLMENWKDISCOVERYRequest*	Void	N	Y
6	V_ABZB_NWK_NLMEJOINRequest	ABZB_Pst_JoinReq	Pointer of type stNLMEJOINRequest		N	Y
7	V_ABZB_NWK_NLMESTARTROUTERRequest	ABZB_Pst_StartRouterReq	Pointer to stNLMESTARTROUTERRequest	Void	N	Y
8	V_ABZB_NWK_NLDEDATARequest	ABZB_Pst_DataReq	Pointer tostNLDEDATARequest	Void	Y	Y
9	V_ABZB_Timer_Soft_set_timer3	ABZB_Uch_TimerNo, ABZB_UI_Time, APP_MBX	UINT8, UINT32, S_Mail_Box	Void	Y	Y

7.2 Call Back Functions

S.No	Primitive Name	Parameter	Type	Return type	Applicable for	
					Coordinator	Router
10	V_ABZB_NWK_NLMERESETConfirm	ABZB_Pst_ResetConfirm	Pointer to stNLMERESETConfirm	Void	Y	Y
11	V_ABZB_NWK_NLMENETWORKFORMATIONConfirm	ABZB_Pst_NetFormationConfirm	Pointer to stNLMENWKFORMATIONConfirm	Void	Y	N
12	V_ABZB_NWK_NLMEPERMITJOININGConfirm	ABZB_Pst_PermitJoinConfirm	Pointer to stNLMEPERMITJOININGConfirm	Void	Y	Y
13	V_ABZB_NWK_NLMENETWORKDISCOVERYConfirm	ABZB_Pst_DiscoveryConfirm	Pointer to stNLMENWKDISCOVERYConfirm	Void	N	Y
14	V_ABZB_NWK_NLMEJOINConfirm	ABZB_Pst_JoinConfirm	Pointer to stNLMEJOINConfirm	Void	N	Y
15	V_ABZB_NWK_NLMEJOINIndication	ABZB_Pst_JoinIndication	Pointer to stNLMEJOINIndication	Void	Y	Y
16	V_ABZB_NWK_NLMESTARTROUTERConfirm	ABZB_Pst_StartRouterConfirm	Pointer to stNLMESTARTROUTERConfirm	Void	N	Y
17	V_ABZB_NWK_NLDEDATAConfirm	ABZB_Pst_DataConfirm	Pointer of type stNLDEDATAConfirm		Y	Y
18	V_ABZB_NWK_NLDEDATAIndication	ABZB_Pst_DataIndication	Pointer of type stNLDEDATAIndication		Y	Y

8 Application Development

Custom applications are developed using the Airbee-ZNS Lite libraries. The library is developed using IAR's 'C' compiler. Hence IAR-IDE 'C' compiler is a pre-requisite for custom application development using Airbee ZNS Lite libraries.

The Stack/Heap size in the IAR-IDE workspace should be 500/600 bytes and is set under the "General" option. C/C++ optimization size should be set to High (maximum optimization).

Application development involves the following process:

- Create / modify source file in 'C'.
- Compile.
- Flash the code on to the device.

To create/modify custom application using the Airbee-ZNS Lite firmware delivered:

- Download the Airbee firmware deliverable
- Double-click 'ZADKV2\Project' folder.
- Double-click 'Project 'folder.
- Double-click the 'IAR-IDE' file named ABZB_ZADKV2.eww to open in IAR-IDE work bench.

Note: Do not change the IAR-IDE workspace environment settings.

- Click the 'Application folder' at the left pane project window.
- Select the project ABZB_ZADKV2-Debug and under 'Options' select the Pre-processor settings.
- Double click the required source file (e.g. ABZB_APP_TASK.c). The file opens in the editor window on the right side pane.

The source file can now be modified / edited. In order to develop a new source file select File>New.

- Compile the file with the source code written / modified by clicking 'Compile' icon.
- Write / modify / create source codes as needed and compile each file as in 10.
- Click on 'Make' to generate .hex file.

The hex files will be generated and added to your folder.

8.1 To Configure a PAN Coordinator:

II Open the ABZB_FOUR_NODE_APP.h and uncomment the macro "COORDINATOR".

III Ensure Routers are commented

IIII Right click ABZB_NWK_COORDINATOR.Lib, select 'Options' and uncheck the exclude check box.

IVI Under Options select 'Linker' at Category list

VI Click Output tab to enter the file name of the Coordinator

VII Select the project ABZB_ZADKV2-Debug to 'Clean' and 'Rebuild All'

8.2 To configure Routers:

VIII Select the project ABZB_ZADKV2-Debug and under 'Options' select the Pre-processor settings

VIIII Open the ABZB_FOUR_NODE_APP.h and uncomment the macro ROUTER.

IXI Ensure Coordinator is commented

XI Open the ABZB_FOUR_NODE_APP.h and uncomment the ROUTER macro

XII Right click ABZB_NWK_ROUTER.Lib, select 'Options' and uncheck the exclude check box.

XIII Under Options select 'Linker' at Category list

XIIII Click Output tab to enter the file name of the Router

XIVI Select the project ABZB_ZADKV2-Debug to 'Clean' and 'Rebuild All'

- Flash the hex files generated to the devices by clicking 'Debug' icon and mark the devices as PAN Coordinator / Router1 / Router2 / Router3.

The application workspace for custom development is provided for TI MSP430F1612 platform. The procedure described above to generate the executables is for MSP430F1612.

To build the executables for TI MSP430F1611, in the project 'Options', under 'Target' tab the 'Device' should be selected as MSP430F1611. Rest of the procedure remain as described above.

For the 4 Node Demo Application, the sample demo Hex files are generated as below:

Open …:…..\IAR Systems\Embedded Workbench 4.0\430\config

Select 'lnk430F1611. xcl' and open the file

Under RAM memory, set the Heap size to 1100 – 2500

Rest of the procedure to generate the executables same as described above.

In the Airbee-ZNS Lite stack library, memory occupied by the Coordinator and router libraries as in the table below:

Build	Flash (kilo bytes)	RAM (kilo bytes)
Coordinator	46	4.4
Router	46	4.4

The Flash and RAM sizes are similar for both MSP430F1611 and 1612.

The memory available for the application programmers' use as below:

Device	Flash (kilo bytes)	RAM (kilo bytes)
MSP430F1611	2	5.6 *
MSP430F1612	9	0.6 *

* By default the stack memory RAM is set to 500 bytes, assuming that the user may require this size of stack memory in most applications. The user has the option to use the unused memory space for his application development by reducing the stack depth, which will increase the available RAM to that extent.

For instance, the 4-node application uses approximately 70 bytes leaving 430 bytes of the stack (RAM) unused.

NOTE:

Only channels 11 to 26 can be used as per IEEE802.15.4 specifications.

Application task name shall not be changed.

Stack profile value 0x05 shall not be changed.

Sequence of the header file inclusion shall not be altered.

Sequence of the APIs called in the application task file ABZB_APP_TASK.c shall not be altered.

9 Application Notes

9.1 PAN Coordinator - Startup

PAN Coordinator is the principal controller of an 802.15.4-2003 based wireless network. The Pan Coordinator is responsible for network formation.

The sequence for network formation is as follows:

Network Reset – Network reset can be invoked through NLME-RESET primitive to reset the network (Refer API V_ABZB_NWK_NLMERESETRequest)

Network Formation Request – Network formation can be invoked through NLME-NETWORK-FORMATION primitive to initialize the device as a ZigBee Ready coordinator of a new network (Refer: V_ABZB_NWK_NLMENETWORKFORMATIONRequest).

Network Permit Joining – Network permit joining can be invoked through NLME-PERMIT-JOINING primitive to allow a ZigBee Ready Coordinator to accept devices onto its network. (Refer: V_ABZB_NWK_NLMEPERMITJOININGRequest). Network formation is complete when Permit Joining is successful.

9.2 Router - Startup

Router is an 802.15.4-2003 device responsible for associating & disassociating devices into its PAN and is capable of routing messages between devices and supporting associations. The sequence for Router startup is as follows:

Network Reset – Network reset can be invoked through NLME-RESET primitive to reset the network (Refer: API V_ABZB_NWK_NLMERESETRequest)

Network Discovery Request – Network discovery can be invoked through NLME-NETWORK-DISCOVERY to discover operating Personal Area Networks in its operating space. (Refer: Network Join Request)

Network Join Request – Network Join request is invoked to enable the router join a network. (Refer: V_ABZB_NWK_NLMEJOINRequest()). The same API can be used fort the rejoin request.

Network Permit Joining – Network permit joining can be invoked through NLME-PERMIT-JOINING primitive to allow the router to accept devices onto its network. (Refer: V_ABZB_NWK_NLMEPERMITJOININGRequest). Network formation is complete when Permit Joining is successful.

9.3 Permit Join

This functionality enables a ZigBee ready PAN Coordinator or Router to set its permit duration. A Router or PAN Coordinator will allow devices to join its network only when the permit duration is active. If the value is set to 0Xff, it is always active.

This feature is exploited by the application to form the target mesh topology designed to demonstrate the Airbee-ZNS Lite feature.

9.4 Application Binding

Application binding is done after network formation. Binding causes logical links to be created

between two or more co-operating devices to accomplish control or monitor functionality between the devices. The source and the destination address of two co-operating devices are paired to form the binding table.

9.5 Mail Box Messaging Concept

The communication (request, response, indication, and confirmation) in the Airbee stack is done by messages passed between the layers through a mail box concept.

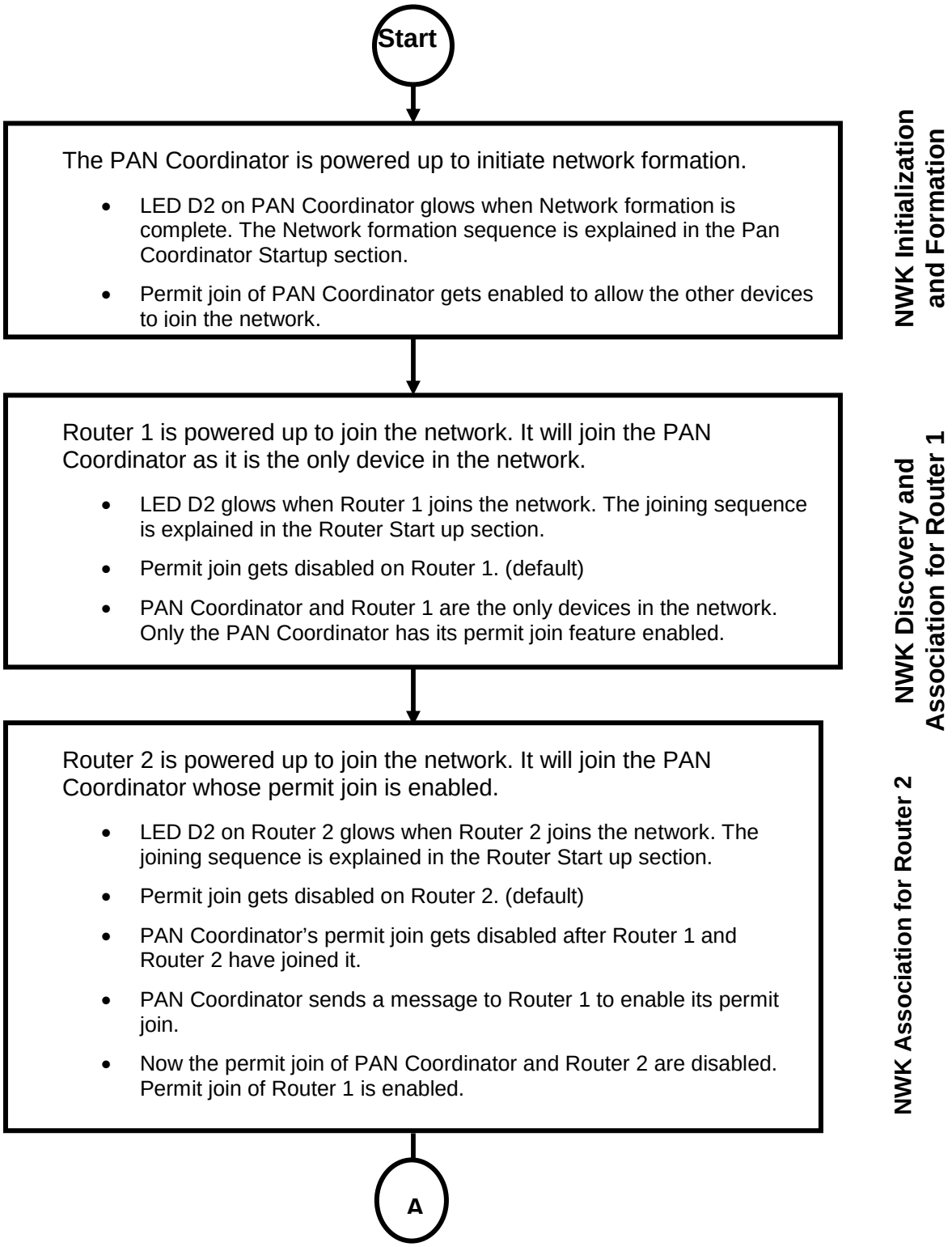
The Real Time Operating System (RTOS) governs the mail box.

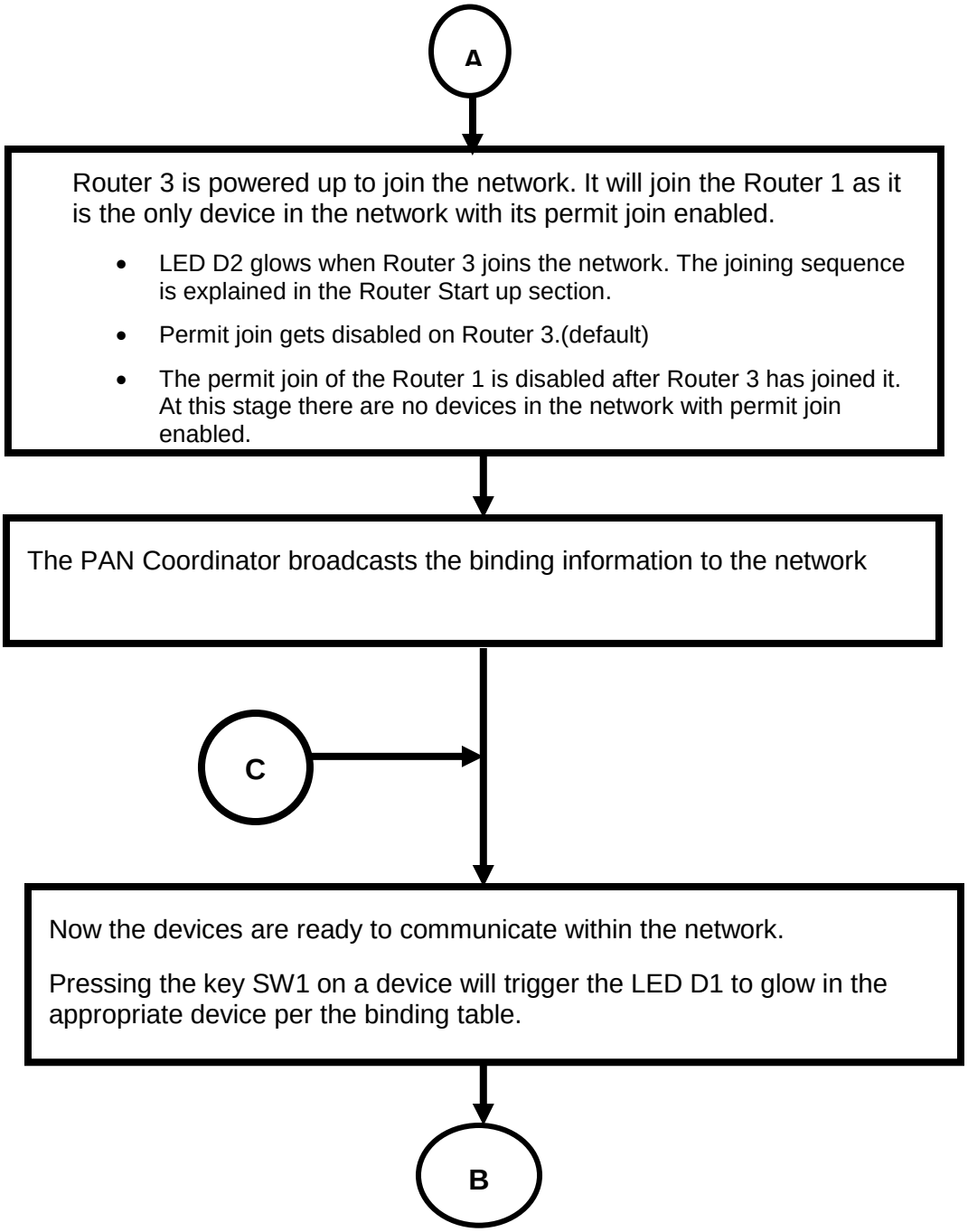
Data and command indications are communicated from the network layer to the application through messaging via the application mail box (APP_MBX). Application messages need not send any message, as the required functionality is expressed as library functions illustrated in the Stack-API section in the *Programmer's Reference Manual*. Fetching and processing the mail box messages can be referred in the ABZB_APP_TASK.C using the utility, GET_MESSAGE function.

User applications should be written into the body of the V_ABZB_APP_TASK_Main() function, example is provided in the ABZB_APP_TASK.c file. Application task is exposed to the user to write his application.

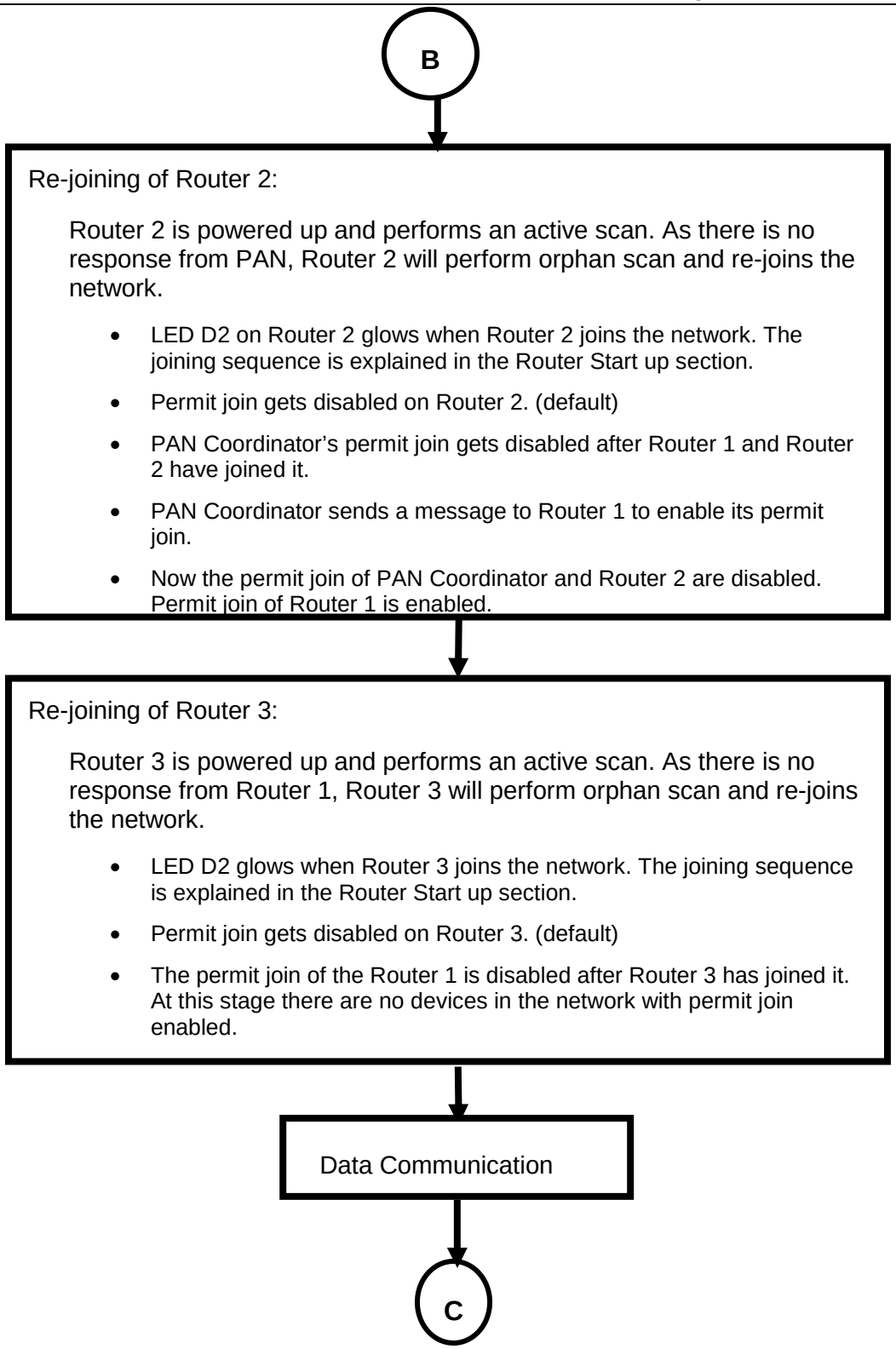
9.6 Behavioral characteristics of Airbee-ZNS Lite

The following flowchart describes the behavioral characteristics of Airbee-ZNS Lite in the 4-node network configuration.


Start
The PAN Coordinator is powered up to initiate network formation.
LED D2 on PAN Coordinator glows when Network formation is complete. The Network formation sequence is explained in the Pan Coordinator Startup section.
Permit join of PAN Coordinator gets enabled to allow the other devices to join the network.
NWK Initialization and Formation
Router 1 is powered up to join the network. It will join the PAN Coordinator as it is the only device in the network.
LED D2 glows when Router 1 joins the network. The joining sequence is explained in the Router Start up section.
Permit join gets disabled on Router 1. (default)
PAN Coordinator and Router 1 are the only devices in the network. Only the PAN Coordinator has its permit join feature enabled.
NWK Discovery and Association for Router 1
Router 2 is powered up to join the network. It will join the PAN Coordinator whose permit join is enabled.
LED D2 on Router 2 glows when Router 2 joins the network. The joining sequence is explained in the Router Start up section.
Permit join gets disabled on Router 2. (default)
PAN Coordinator's permit join gets disabled after Router 1 and Router 2 have joined it.
PAN Coordinator sends a message to Router 1 to enable its permit join.
Now the permit join of PAN Coordinator and Router 2 are disabled. Permit join of Router 1 is enabled.
NWK Association for Router 2
A



NWK Association for Router 3



The re-joining process need not occur in the given sequence in the flowchart. It may occur anywhere in the process using orphan scan.

The child device will not be able to join the network if its parent device is rejoined the network after the child device has joined.

10 Limitations

- Until the bind table update is completed in all the four devices, the devices shall be within the network range.
- All four devices must participate in the network.
- Bindings can be done only in one-to-one format, e.g. one key-press can be bound to one lamp only.
- This firmware supports only four device network and does not feature upper Application Layer. However, Airbee has simplified interfaces with Application Layer to enable simpler development of custom applications.
- The NSDUHANDLE in NLDEDataRequest should not be 0xEA or 0xEB or 0XEC. Also, the first byte of the NSDUData should be 0x05.
- The IEEE address cannot be modified.