Freescale BeeStack Consumer

Reference Manual

Document Number: BSCONRM
Rev. 2.0
06/2011

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2008, 2009, 2010, 2011. All rights reserved.

Contents

About This Book . . . iii
Audience . . . iii
Organization . . . iii
Revision History . . . iii
Definitions, Acronyms, and Abbreviations . . . iv
References . . . iv

Chapter 1
Introduction

1.1 What This Document Describes . . . 1-1
1.2 What This Document Does Not Describe . . . 1-1

Chapter 2
BeeStack Consumer Software Overview

2.1 Network Topology . . . 2-1
2.2 Node Functionality Enumeration . . . 2-3
2.3 System Overview . . . 2-3
2.4 BeeStack Consumer Available Libraries . . . 2-4

Chapter 3
BeeStack Consumer Network Layer Interface Description

3.1 Network Layer Interface . . . 3-1
3.2 System API . . . 3-2
3.2.1 NLME_StartRequest . . . 3-5
3.2.2 NLME_AutoDiscoveryRequest . . . 3-7
3.2.3 NLME_DiscoveryRequest . . . 3-8
3.2.4 NLME_DiscoveryResponse . . . 3-11
3.2.5 NLME_PairRequest . . . 3-13
3.2.6 NLME_PairResponse . . . 3-15
3.2.7 NLME_UnpairRequest . . . 3-17
3.2.8 NLME_UnpairResponse . . . 3-18
3.2.9 NLME_UpdateKeyRequest . . . 3-19
3.2.10 NLME_GetRequest . . . 3-20
3.2.11 NLME_SetRequest . . . 3-21
3.2.12 NLME_RxEnableRequest . . . 3-22
3.2.13 NLME_ResetRequest . . . 3-24
3.2.14 NLDE_DataRequest . . . 3-25
3.2.15 NWK_GetNodePanId . . . 3-27
3.2.16 NWK_ GetNodeShortAddress . . . 3-28
3.2.17 NWK_GenerateShortAddress . . . 3-28
3.2.18 NWK_GenerateSecurityKey . . . 3-29
3.2.19 NWK_AddNewPairTableEntry . . . 3-30
3.2.20 NWK_SavePersistentData . . . 3-31

3.2.21 NWK_ SaveFrameCounter . . . 3-32
3.2.22 NWK_SetMacAddress . . . 3-33
3.2.23 NWK_GetMacAddress . . . 3-34
3.2.24 NWK_GetLastPacketLQI . . . 3-34
3.2.25 NWK_GetAllowedLowPowerInterval . . . 3-35
3.2.26 NWK_IsIdle . . . 3-36
3.3 Message Data Types . . . 3-36
3.3.1 nwkNlmeStartCnf_t . . . 3-38
3.3.2 nwkNlmeAutoDiscoveryCnf_t . . . 3-39
3.3.3 nwkNlmeDiscoveryCnf_t . . . 3-41
3.3.4 nwkNlmeDiscoveryInd_t . . . 3-43
3.3.5 nwkNlmePairCnf_t . . . 3-45
3.3.6 nwkNlmePairInd_t . . . 3-47
3.3.7 nwkNlmeUnpairCnf_t . . . 3-49
3.3.8 nwkNlmeUnpairInd_t . . . 3-50
3.3.9 nwkNlmeCommStatusInd_t . . . 3-50
3.3.10 nwkNldeDataCnf_t . . . 3-52
3.3.11 nwkNldeDataInd_t . . . 3-53
3.4 BeeStack Consumer Data Types . . . 3-54
3.4.1 NodeData Database . . . 3-54
3.4.2 BeeStack Consumer NIBs . . . 3-56
3.4.3 Saving BeeStack Consumer Sensitive Information in FLASH . . . 3-61

Chapter 4
BeeStack Consumer/SynkroRF Hybrid Software Overview

4.1 Network Topology . . . 4-1
4.2 Node Functionality Enumeration . . . 4-1
4.3 Hybrid Available Libraries . . . 4-2

Chapter 5
BeeStack Consumer/SynkroRF Hybrid Network Layer Interface Description

5.1 General Hybrid Network Interface Information . . . 5-1
5.2 System API . . . 5-1
5.2.1 SynkroRF_PairRequest . . . 5-2
5.2.2 SynkroRF_PairResponse . . . 5-4
5.2.3 SynkroRF_ClearPairingInformation . . . 5-5
5.2.4 SynkroRF_SendCommand . . . 5-6
5.2.5 SynkroRF_AddNewPairTableEntry . . . 5-7
5.2.6 SynkroRF_GetPairedDeviceInfo . . . 5-9
5.2.7 Nwk_GetNodeType . . . 5-10
5.3 Message Data Types . . . 5-11
5.3.1 nwkSynkroRFPairCnf_t . . . 5-13
5.3.2 nwkSynkroRFPairInd_t . . . 5-14
5.3.3 nwkSynkroRFCommandCnf_t . . . 5-15

5.3.4 nwkSynkroRFCommandInd_t 5-16
5.4 Hybrid Data Types 5-17
5.4.1 NodeData Database. 5-17
5.4.2 Hybrid NIBs 5-17
5.4.3 Local SynkroRF Node Descriptors. 5-17
5.4.4 Saving Sensitive Information in FLASH 5-18

About This Book

This reference manual describes the functionality of the Freescale BeeStack Consumer network and provides detailed descriptions which will allow users to develop upper layer or application code for this product.

The Freescale BeeStack Consumer network software is designed for use with the following families of short range, low power, 2.4 GHz Industrial, Scientific, and Medical (ISM) band transceivers:

- Freescale MC1319x and MC1320x families, designed for use with the HC(S)08 GT/GB MCU.
- Freescale MC1321x family, that incorporate a low power 2.4 GHz radio frequency transceiver and an 8-bit microcontroller into a single LGA package.
- Freescale MC1322x Platform-In-Package, that combines a low power 2.4 GHz frequency transceiver and a 32-bit ARM7 microcontroller into a single LGA package.
- Freescale MC1323x low cost System-on-Chip (SoC) platform for the IEEE® 802.15.4 Standard that incorporates a complete, low power, 2.4 GHz radio frequency transceiver with Tx/Rx switch, an 8-bit HCS08 CPU, and a functional set of MCU peripherals into a 48-pin QFN package.

Audience

This reference manual is intended for application designers and users of the BeeStack Consumer network library.

Organization

This document contains the following chapters:

Chapter 1	Introduction - Describes this document.
Chapter 2	BeeStack Consumer Software Overview - Introduces the BeeStack Consumer software.
Chapter 3	BeeStack Consumer Network Layer Interface Description - Provides a description of the network interfaces.
Chapter 4	Provides a brief overview of the BeeStack Consumer/SynkroRF hybrid network software.
Chapter 5	Describes the BeeStack Consumer/SynkroRF hybrid network layer interface.

Revision History

The following table summarizes revisions to this manual since the previous release (Rev. 1.9).

Revision History

Date / Author	Description / Location of Changes
June 2011, Dev Team	Various updates throughout for MC1323x and CodeWarrior 10

Definitions, Acronyms, and Abbreviations

The following list defines the abbreviations used in this document.

API	Application Programming Interface
BeeStack Consumer	Freescale's implementation of the ZigBee RF4CE Standard
CE	Consumer Electronics
LQI	Link Quality Indication
NIB	Network Information Base
NLDE	Network Layer Data Entity
NLME	Network Layer Management Entity
NVM	Non volatile memory
NW Layer	Network Layer
RC	Remote Control
RF	Radio Frequency
SAP	Service Access Point
SynkroRF	An entertainment control network technology
PAN	Personal Area Network

References

The following sources were referenced to produce this book:

1. RF4CE Consumer Specification version 1.0.0, Document 080002r04
2. IEEE 802.15.4 Standard -2003, Part 14.5: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), The Institute of Electrical and Electronics Engineers, Inc. October 2003
3. *BeeStack Consumer Application User's Guide* (BSCONAUG)
4. *Freescale BeeKit Wireless Connectivity Toolkit User's Guide* (BKWCTKUG)

Chapter 1
Introduction

This manual describes the Freescale BeeStack Consumer protocol stack, its components and their functional roles in building Remote Control (RC) networks. The application programming interfaces (API) and messages included in this manual address every component required for communication in a RC network.

1.1 What This Document Describes

This manual provides BeeStack Consumer software designers and developers all of the function prototypes, macros and stack libraries required to develop applications for BeeStack Consumer wireless networks.

1.2 What This Document Does Not Describe

This manual does not describe how to install software, configure the hardware, or set up and use BeeStack Consumer applications.

See the following documents for help in setting up the Freescale hardware and using other Freescale software to configure devices.

- *BeeStack Consumer Application User's Guide*, (BSCONAUG)
- *Freescale BeeKit Wireless Connectivity Toolkit User's Guide*, (BKWCTKUG)

Chapter 2
BeeStack Consumer Software Overview

The BeeStack Consumer network is a software networking layer that sits on top of the IEEE 802.15.4 MAC and PHY layers. It is designed for Wireless Personal Area Networks (WPANs) and conveys information over short distances among the participants in the network. It enables small, power efficient, inexpensive solutions to be implemented for a wide range of applications. Some key characteristics of an BeeStack Consumer network are:

- An over the air data rate of 250 kbit/s in the 2.4 GHz band
- Three independent communication channels in the 2.4 GHz band
- Two network node types, controller node and respectively target node
- Channel Agility mechanism
- Provides robustness and ease of use
- Includes essential functionality to build and support a CE network

Figure 2-1 shows the software architecture of an application using the BeeStack Consumer network.

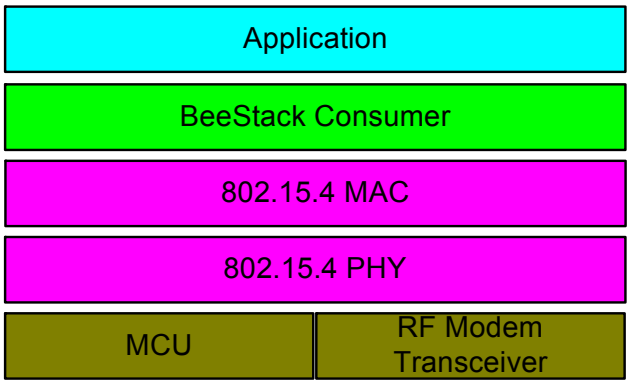

Figure 2-1. BeeStack Consumer Network Application Structure

2.1 Network Topology

An RC PAN is composed of the following two types of devices:

- Target Node
- Controller Node

A target node has full PAN coordinator capabilities and can start a network in its own right. Both types of node can join networks started by target nodes by pairing with that target. Multiple RC PANs form an RC network and nodes in the network can communicate between RC PANs.

To communicate with a target node, a controller node first switches to the channel and assumes the PAN identifier of the destination RC PAN. It then uses the network address, allocated through the pairing procedure, to identify itself on the RC PAN and thus communicate with the desired target node.

Figure 2-2 shows an example BeeStack Consumer topology which includes three target nodes:

- TV
- DVD Player
- CD Player

Each target node creates its own RC PAN. The TV, DVD and CD player also have dedicated RCs which are paired to each appropriate target node. A multi-function RC, capable of controlling all three target nodes itself, is added to the network by successively pairing to the desired target nodes. The DVD is also paired with the TV. For example the external input corresponding to the DVD can be selected on the TV when a DVD is inserted and played.

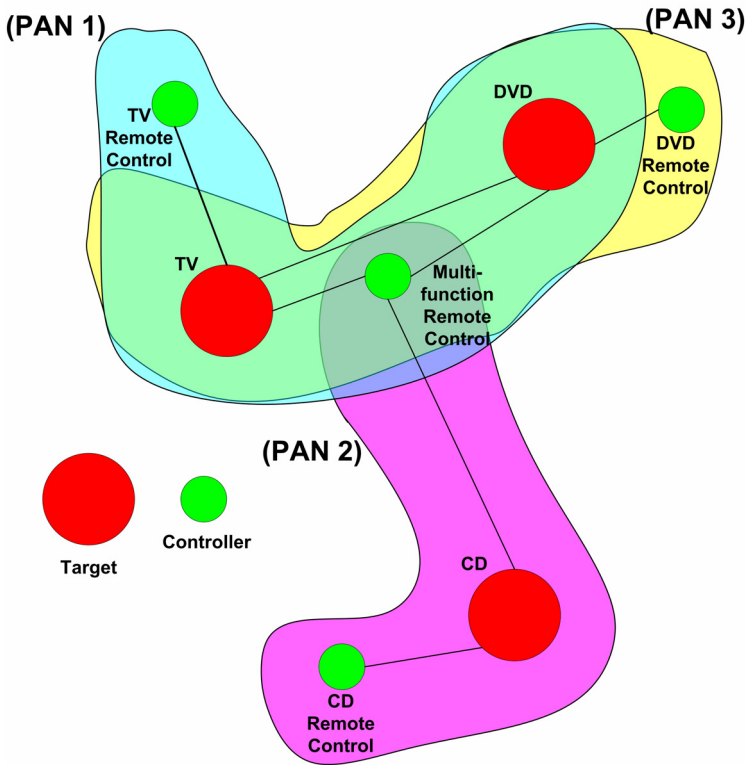

Figure 2-2. Example BeeStack Consumer Network Topology

As a consequence, this RC network consists of three separate RC PANs:

- Pan 1, managed by the TV that contains the TV RC, the multi-function RC and the DVD.
- Pan 2, managed by the CD player that contains the CD RC and the multi function RC.
- Pan 3, managed by the DVD that contains the DVD RC, multi-function RC and the TV.

2.2 Node Functionality Enumeration

The basic node types for the Freescale BeeStack Consumer network software are as follows:

- **Controller node -** This device contains a subset of the BeeStack Consumer features. The controller node type is used in remote control applications. This node does not start a PAN. During the pairing process, it receives a Pan Id and Short Address from the target node it has paired with. These values are used in future communications with the paired target node.
- **Target node -** This device contains a subset of the BeeStack Consumer features. The target node type is used in Consumer Electronics device applications (e.g. TV's, DVD's, etc.). It is intended to be used in backbone powered devices. Starting a target node always starts a new PAN.

2.3 System Overview

The BeeStack Consumer layer implements an interface between the application and the IEEE.802.15.4 MAC layer. The BeeStack Consumer layer conceptually includes a management entity that provides the service interfaces through which layer management functions may be invoked. This management entity is also responsible for maintaining a database of managed objects pertaining to the BeeStack Consumer layer.

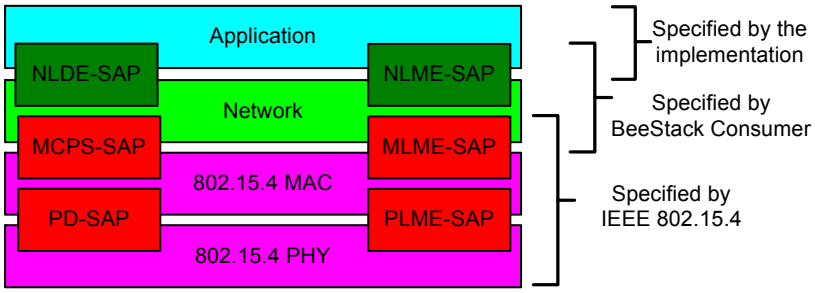

Figure 2-3. BeeStack Consumer Layer Interfaces

The BeeStack Consumer layer provides a number of services, accessed through the BeeStack Consumer API. The BeeStack Consumer services are listed in Table 3-1, Table 3-2 and Table 3-3 and they have the following characteristics:

- Are started using BeeStack Consumer API function calls
- Communicate to application the execution status using confirm messages
- Inform the application layer about asynchronous network information arrival using indication messages

2.4 BeeStack Consumer Available Libraries

This section describes the libraries available for the BeeStack Consumer software stack. There are two available libraries, with different code sizes due to functionality reduction.

Table 2-1. BeeStack Consumer Libraries

Library	Description	Usage
BeeStackConsumer.lib	Contains all BeeStack Consumer features	Should be used by applications exercising all network features. Includes API parameters validation.
BeeStackConsumer_NV.lib	Same as BeeStackConsumer.lib but no code for the verification of the parameters of any of the network services requested by the application through the network API.	Should be used by applications which are confident that the values of the API parameters are within the valid ranges specified by the RF4CE specification
BeeStackConsumer_NS.lib	Same as BeeStackConsumer.lib but no code for implementing the security support in the network.	Should be used by applications which are not exchanging encrypted data.
BeeStackConsumer_NA.lib	Same as BeeStackConsumer.lib but no code for the implementing the AutoDiscovery network feature.	Should be used by applications that do not need the AutoDiscovery network service to implement the PushButtonPairing procedure of ZRC profile.
BeeStackConsumer_NVNSNA.lib	Same as BeeStackConsumer.lib but no code for the API parameter verification, security support and AutoDiscovery service.	Should be used by applications that do not need security and do not require ZRC profile support features. API parameters are not verified, to achieve the smallest code size possible.
BeeStackConsumer_ZTC.lib	Contains all BeeStackConsumer features and also supports monitoring the interface between the MAC and the network from the ZTC application	Should be used in the development and validation phase. The added support for monitoring the stack interfaces increases the code size requirements.
BeeStackConsumer_ZTC_NV.lib	Same as BeeStackConsumer_ZTC but no code for the verification of network API parameters.	Should be used in the development and validation phase. The added support for monitoring the stack interfaces increases the code size requirements. Does not include API parameters validation.

Chapter 3
BeeStack Consumer Network Layer Interface Description

3.1 Network Layer Interface

The interface between the Application Layer and the Network Layer Management Entity (NLME) as well as the Network Layer Data Entity (NLDE) is based on primitives passed from one layer to the other through APIs and messages, as illustrated in Figure 3-1 All the requests and responses in the BeeStack Consumer standard have a distinct API while all the confirms and the indications have a distinct message.

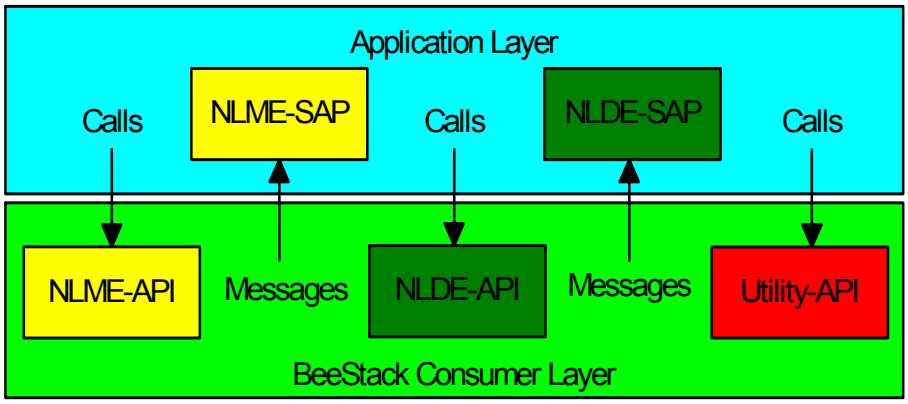

Figure 3-1. BeeStack Consumer Layer Interfaces

3.2 System API

This section describes the structures and functions available in the System API. To use the API, the `NwkInterface.h` header file must be included in the relevant source code files.

Table 3-1. BeeStack Consumer API in the Application to BeeStack Consumer Direction

	BeeStack Consumer API Function Name	Description	Over the air Activity	Synchronous Call	Available on Controller	Available on Target	Section
NLME	NLME_StartRequest	NLME-START.request	1)		X	X	3.3.1
	NLME_AutoDiscoveryRequest	NLME-AUTO-DISCOVERY.request	X		X	X	3.3.2
	NLME_DiscoveryRequest	NLME-DISCOVERY.request	X		X	X	3.3.3
	NLME_DiscoveryResponse	NLME-DISCOVERY.response	X		X	X	3.3.4
	NLME_PairRequest	NLME-PAIR.request	X		X	X	3.3.5
	NLME_PairResponse	NLME-PAIR.response	X		X	X	3.3.6
	NLME_UnpairRequest	NLME-UNPAIR.request	X		X	X	3.3.7
	NLME_UnpairResponse	NLME-UNPAIR.response		X	X	X	3.3.8
	NLME_UpdateKeyRequest	NLME-Update-Key.request		X	X	X	3.3.9
	NLME_GetRequest	NLME-GET.request		X	X	X	3.3.10
	NLME_SetRequest	NLME-SET.request		X	X	X	3.3.11
	NLME_RxEnableRequest	NLME-RX-ENABLE.request		X	X	X	3.2.12
	NLME_ResetRequest	NLME-RESET.request		X	X	X	3.2.13
NLDE	NLDE_DataRequest	NLME-DATA.request	X		X	X	3.2.14

Table 3-1. BeeStack Consumer API in the Application to BeeStack Consumer Direction (continued)

Freescale Implementation Specific Utility Services	NWK_GetNodePanId	This macro allows the application to		X	X	X	3.2.15
	NWK_GetNodeShortAddress	This macro allows the application to obtain the short address of the local node.		X	X	X	3.2.16
	NWK_GenerateShortAddress	This function allows the application to request the generation of a short address, that does not match any of the short addresses of the devices in the pair table.		X	X	X	3.2.17
	NWK_GenerateSecurityKey	This function allows the application to request the generation of a security key, that does not match any of the security keys of the devices in the pair table.		X	X	X	3.2.18
	NWK_AddNewPairTableEntry	This function allows the application to insert ‘offline’ the information of a new entry in the pair table, without initiating a pair process.		X	X	X	3.2.19
	NWK_SavePersistentData	This function allows the application to request the immediate saving of the BeeStack Consumer sensitive information into FLASH.		X	X	X	3.2.20
	NWK_SaveFrameCounter	This function allows the application to request the immediate saving of the nwkFrameCounter NIB attribute’s current value into FLASH.		X	X	X	3.2.21
	NWK_SetMacAddress	This function allows the application to request the setting of the IEEE 802.15.4 MAC address of the local node.		X	X	X	3.2.22
	NWK_GetMacAddress	This function allows the application to obtain the IEEE 802.15.4 MAC address of the local node.		X	X	X	3.2.23
	NWK_GetLastPacketLQI	This function allows the application to obtain the LQI of the last packet received by the network.		X	X	X	3.2.24
	NWK_GetAllowedLowPowerInterval	This function allows the application to determine if BeeStack Consumer layer can safely enter a low power mode and for how much time.		X	X	X	3.2.25
	NWK_IsIdle	This function allows the application to be		X	X	X	3.2.26

The call has activity over the air only for target nodes.

The asynchronous API calls start a network internal process. The value returned by the call only informs the calling entity if the request is accepted to be processed or not, and in the second case it offers information about the reason why the request was denied. If the return value indicates success, the BeeStack Consumer layer has accepted and already initiated the process. When the process will be completed, the application layer will be notified by a confirm message which will be sent by BeeStack Consumer layer through one of the BeeStack Consumer NLME/NLDE SAPs.

The BeeStack Consumer network layer is able to handle only one process at a time. There are two types of processes the network can execute:

- Non-interruptible processes: These are processes that cannot be interrupted by other requests from the application that will trigger the start of a new process. In this case, the second request will fail with the gNWDenied_c status and the network will continue to run the process started by the first request.
  - Non interruptible processes are: NLME_StartRequest, NLME_DiscoveryResponse, NLME_PairRequest, NLME_PairResponse, NLME_UnpairRequest.
- Interruptible processes: These are processes that are aborted if other requests from the application that will trigger the start of a new process are received while they are being executed. In this case, a confirm message with the status set to gNWAborted_c will be sent to the application, informing it about the aborting of the initial request. The second request will begin to be processed.
  - Interruptible processes: NLME_DiscoveryRequest, NLME_AutoDiscovery, NLDE_DataRequest

The synchronous API calls do not start a network internal process. When the application layer receives the return value of the API call, the primitive is completely executed. There will be no other confirm message sent by BeeStack Consumer layer trough the BeeStack Consumer NLME/NLDE SAP.

All BeeStack Consumer API calls return a code to report on their operation. The codes are defined as follows:

```
/* Return codes from the network */
#define gNWSuccess_c             0x00    /* Requested action was completed successfully */
#define gNWNoOrigCapacity_c       0xb0    /* No room in the originator pair table to add a new
entry */
#define gNWNoRecipCapacity_c      0xb1    /* No room in the recipient pair table to add a new
entry */
#define gNWDeviceIdNotPaired_c    0xb2    /* No pair information at the location specified by
deviceId */
#define gNWNoResponse_c           0xb3    /* No response was received by the originator from
the recipient */
#define gNWNotPermitted_c         0xb4    /* Operation not permitted */
#define gNWDuplicatePairing_c     0xb5    /* A duplicate pairing table entry was detected in a
pair request command */
#define gNWFrameCounterExpired_c  0xb6    /* The frame counter has reached its maximum value */
#define gNWDiscoveryError_c       0xb7    /* Too many unique discovery request or response
command frames were received than requested */
#define gNWDiscoveryTimeout_c     0xb8    /* No discovery request or response command frames
were received during discovery */
#define gNWSecurityTimeout_c     0xb9   /* The security link key exchange or recovery procedure
did not complete within the required time */
#define gNWSecurityFailure_c     0xba   /* A security link key was not successfully established
between both ends of a pairing link */
#define gNWInvalidParam_c        0xe8   /* One of the parameters of the API function is invalid
*/
#define gNWUnsupportedAttribute_c 0xf4    /* NIB Attribute not supported */
#define gNWInvalidIndex_c         0xf9    /* Table index out of range */

/* Freescale specific return codes */
```

```
#define gNWDenied_c              0x80    /* Request denied by the BeeStack Consumer layer, as
a result of the fact that another request that has started an non-interruptible process is
already executing */
#define gNWNoMemory_c            0x81    /* No memory available for the BeeStack Consumer to
perform requested action */
#define gNWNodeAlreadyStarted_c  0x82    /* Received when trying to start an already started
BeeStack Consumer node */
#define gNWNodeNotStarted_c      0x83    /* BeeStack Consumer node should be started before
executing requested action */
#define gNWNoTimers_c            0x84    /* No free timers available to perform the requested
action */
#define gNWAborted_c             0x85    /* The previously started interruptible process
(NLME_DiscoveryRequest, NLME_AutoDiscoveryRequest or NLDE_DataRequest) was aborted as a result
of the received request to start another process */

/* Mac return codes */
#define gSuccess_c              0x00
#define gPanAtCapacity_c        0x01
#define gPanAccessDenied_c      0x02
#define gBeaconLoss_c           0xE0
#define gChannelAccessFailure_c 0xE1
#define gDenied_c               0xE2
#define gDisableTrxFailure_c    0xE3
#define gFailedSecurityCheck_c  0xE4
#define gFrameTooLong_c         0xE5
#define gInvalidGts_c           0xE6
#define gInvalidHandle_c        0xE7
#define gInvalidParameter_c     0xE8
#define gNoAck_c                0xE9
#define gNoBeacon_c             0xEA
#define gNoData_c               0xEB
#define gNoShortAddress_c       0xEC
#define gOutOfCap_c             0xED
#define gPanIdConflict_c        0xEE
#define gRealignment_c          0xEF
#define gTransactionExpired_c   0xF0
#define gTransactionOverflow_c  0xF1
#define gTxActive_c             0xF2
#define gUnavailableKey_c       0xF3
#define gUnsupportedAttribute_c 0xF4
```

Each BeeStack Consumer API function is covered in more detail in the following sections.

3.2.1 NLME_StartRequest

Prototype

```
uint8_t NLME_StartRequest (void)
```

Arguments

The NLME_StartRequest primitive has no parameters.

NOTE

The behavior of the primitive depends on the value of the gNwkNib_StartWithNetworkInfoFromFlash_c NIB entry.

If the StartWithNetworkInfoFromFlash_c NIB entry is TRUE, the network will be started after loading the relevant data from FLASH.

If the StartWithNetworkInfoFromFlash_c NIB entry is FALSE, the network will not load the data from FLASH before executing the start procedure.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request has been accepted for processing by the network layer
- `gNWNodeAlreadyStarted_c` — The request is rejected as the node is already started
- `gNWDenied_c` — The request is rejected as the network layer is already processing another non-interruptible request
- `gNWNoMemory_c` — The request is rejected as the network needs to allocate messages from the common message pool which is empty
- `gNWNoTimers_c` — The request is rejected as no timers can be allocated to handle the request
- `gNWFrameCounterExpired_c` — Returned only when the start is made using persistent information from FLASH and when by adding the nwkcFrameCounterWindow value to the value of the frameCounter NIB from FLASH, the frameCounter NIB maximum range would be reached.

Functional Description

The NLME_StartRequest is an asynchronous API function available both for controller and target nodes. It makes a request for a BeeStack Consumer node to start the network layer.

This function call requests the starting of a BeeStack Consumer Start process. If the return value is `gNWSuccess_c`, the BeeStack Consumer layer has accepted and already started to process the Start request. When the Start process will be completed, the application layer will be notified by a Start Confirm message which will be sent by BeeStack Consumer layer trough the BeeStack Consumer NLME SAP. If the return value is not `gNWSuccess_c`, the BeeStack Consumer Start process will not be started. At this point, the request is considered to be completed and therefore the application should not wait for any Start Confirm message to be received later.

3.2.2 NLME_AutoDiscoveryRequest

Prototype

```
uint8_t NLME_AutoDiscoveryRequest
(
        appCapabilities_t recipAppCapabilities,
        uint8_t*          recipDeviceTypeList,
        uint8_t*          recipProfileIdList,
        uint32_t          autoDiscDuration
);
```

Arguments

Table 3-2 specifies the parameters for the NLME_AutoDiscoveryRequest primitive.

Table 3-2. NLME_AutoDiscoveryRequest Parameters

Name	Type	Valid range	Description
recipAppCapabilities	appCapabilities_t	-	The application capabilities of this node.
recipDeviceTypeList	uint8_t*	Each integer: 0x00 – 0xfe	The list of device types supported by this node.
recipProfileIdList	uint8_t*	Each integer: 0x00 – 0xff	The list of profile identifiers supported by this node.
autoDiscDuration	uint32_t	0x000000 – 0xffffff	The maximum number of MAC symbols NLME will be in auto discovery response mode.

```
appCapabilities_t is defined as:
typedef struct appCapabilities_tag
{
    uint8_t                 bUserStringSpecified   :1;
    uint8_t                 nrSupportedDeviceTypes :2;
    uint8_t                 reserved1              :1;
    uint8_t                 nrSupportedProfiles    :3;
    uint8_t                 reserved2              :1;
}appCapabilities_t;
```

Where bUserStringSpecified indicates whether the requesting device has a defined user string which will be transmitted over the air, while nrSupportedDeviceTypes and nrSupportedProfiles indicate the sizes of recipDeviceTypeList and recipProfileIdList.

The recipDeviceTypeList and recipProfileIdList must contain at least one element each.

Returns

Possible return values and their significance:

- `gNWSuccess_c`: the request has been accepted for processing by the network layer
- `gNWNodeNotStarted_c`: the request is rejected as the node is not started yet.
- `gNWDenied_c`: the request is rejected as the network layer is already processing another non-interruptible request

- gNWNoMemory_c: : the request is rejected as the network needs to allocate messages from the common message pool which is empty
- gNWInvalidParam_c:
  - The request is rejected as autoDiscDuration parameter is bigger than 0xFFFFFF
  - The request is rejected as recipAppCapabilities.nrSupportedDeviceTypes is equal to 0
  - The request is rejected as receipAppCapabilities.nrSupportedProfiles is equal to 0
- gNWNotPermitted_c — The request is rejected as the node is in power save mode (NIB attribute nwkInPowerSave is set to TRUE)
- gNWFrameCounterExpired_c — The request is rejected as the frame counter has reached its maximum value

Functional Description

The NLME_AutoDiscoveryRequest is an asynchronous API function available both for controller and target nodes. It makes a request for a BeeStack Consumer node to enter the auto discovery response mode.

This function call requests the starting of a BeeStack Consumer Auto Discovery process. If the return value is gNWSuccess_c, the BeeStack Consumer layer has accepted and already started to process the Auto Discovery request. When the Auto Discovery process will be completed, the application layer will be notified by an Auto Discovery Confirm message which will be sent by BeeStack Consumer layer trough the BeeStack Consumer NLME SAP.

If the return value is not gNWSuccess_c, the BeeStack Consumer Auto Discovery process will not be started. At this point, the request is considered to be completed and therefore the application should not wait for any Auto Discovery Confirm message to be received later.

3.2.3 NLME_DiscoveryRequest

Prototype

```
uint8_t NLME_DiscoveryRequest
(
        uint8_t*          recipPanId,
        uint8_t*          recipShortAddress,
        uint8_t           recipDeviceType,
        appCapabilities_t origAppCapabilities,
        uint8_t*          origDeviceTypeList,
        uint8_t*          origProfileIdList,
        uint8_t           discProfileIdListSize,
        uint8_t*          discProfileIdList,
        uint32_t          discDuration
);
```

Arguments

Table 3-3 specifies the parameters for the NLME_DiscoveryRequest primitive.

Table 3-3. NLME_DiscoveryRequest Parameters

Name	Type	Valid range	Description
recipPanId	uint8_t*	A valid PAN Identifier (different of NULL)	The PAN identifier of the destination device for the discovery. This value can be set to 0xffff to indicate a wildcard.
recipShortAddress	uint8_t*	A valid short network address (different of NULL)	The address of the destination device for the discovery. This value can be set to 0xffff to indicate a wildcard.
recipDeviceType	uint8_t	0x00 – 0xff	The device type to discover. This value can be set to 0xff to indicate a wildcard.
origAppCapabilities	appCapabilities_t	-	The application capabilities of this node.
origDeviceTypeList	uint8_t*	Each integer: 0x00 – 0xfe	The list of device types supported by this node.
origProfileIdList	uint8_t*	Each integer: 0x00 – 0xff	The list of profile identifiers supported by this node.
discProfileIdListSize	uint8_t	0x00 – 0xff	The number of profile identifiers contained in the discProfileIdList parameter.
discProfileIdList	uint8_t*	Each integer: 0x00 – 0xff	The list of profile identifiers against which profile identifiers contained in received discovery response command frames will be matched for acceptance.
discDuration	uint32_t	0x000000 – 0xffffff	The maximum number of MAC symbols to wait for discovery responses to be sent back from potential target nodes on each channel.

```
appCapabilities_t is defined as:
typedef struct appCapabilities_tag
{
    uint8_t                 bUserStringSpecified   :1;
    uint8_t                 nrSupportedDeviceTypes :2;
    uint8_t                 reserved1              :1;
    uint8_t                 nrSupportedProfiles    :3;
    uint8_t                 reserved2              :1;
}appCapabilities_t;
```

where bUserStringSpecified indicates whether the requesting device has a defined user string which will be transmitted on the air, while nrSupportedDeviceTypes and nrSupportedProfiles indicate the sizes of origDeviceTypeList and origProfileIdList.

The origDeviceTypeList and origProfileIdList must contain at least one element each.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request has been accepted for processing by the network layer
- `gNWNodeNotStarted_c` — The request is rejected as the node is not started yet
- `gNWDenied_c` — The request is rejected as the network layer is already processing another non-interruptible request
- `gNWNoMemory_c` — The request is rejected as the network needs to allocate messages from the common message pool which is empty
- `gNWFrameCounterExpired_c` — The request is rejected as the frame counter has reached its maximum value
- `gNWInvalidParam_c`:
  - The request is rejected as discDuration parameter is bigger than 0xFFFFFF
  - The request is rejected as discDuration parameter is bigger than (0.33 * NIB attribute nwkDiscoveryRepetitionInterval)
  - The request is rejected as origAppCapabilities.nrSupportedDeviceTypes is equal to 0
  - The request is rejected as origAppCapabilities.nrSupportedProfiles is equal to 0

Functional Description

The NLME_DiscoveryRequest is an asynchronous API function available both for controller and target nodes. It makes a request for a BeeStack Consumer node to discover other devices of interest (controller or target nodes), operating in its Personal Operating Space (POS).

This function call requests the starting of a BeeStack Consumer DiscoveryRequest process. If the return value is `gNWSuccess_c`, the BeeStack Consumer layer has accepted and already started to process the Discovery request. When the DiscoveryRequest process will be completed, the application layer will be notified by a Discovery confirm message which will be sent by BeeStack Consumer layer trough the BeeStack Consumer NLME SAP.

If the return value is not `gNWSuccess_c`, the BeeStack Consumer Discovery process will not be started. At this point, the request is considered to be completed and therefore the application should not wait for any Discovery Confirm message to be received later.

3.2.4 NLME_DiscoveryResponse

Prototype

```
uint8_t NLME_DiscoveryResponse
(
        uint8_t           status,
        uint8_t*          recipMacAddress,
        appCapabilities_t origAppCapabilities,
        uint8_t*          origDeviceTypeList,
        uint8_t*          origProfileIdList,
        uint8_t           discoveryReqLQI
);
```

Arguments

Table 3-4 specifies the parameters for the NLME_DiscoveryResponse primitive.

Table 3-4. NLME_DiscoveryResponse Parameters

Name	Type	Valid range	Description
status	uint8_t	gNWSuccess or gNWNoRecipCapacity_c	The status of the associated discovery indication
recipMacAddress	uint8_t*	Valid IEEE address (different of NULL)	The IEEE address of the device requesting discovery
origAppCapabilities	appCapabilities_t	-	The application capabilities of this node
origDeviceTypeList	uint8_t*	Each integer: 0x00 – 0xfe	The list of device types supported by this node
origProfileIdList	uint8_t*	Each integer: 0x00 – 0xff	The list of profile IDs supported by this node
discoveryReqLQI	uint8_t	0x0 – 0xff	The LQI of the associated discovery indication

```
appCapabilities_t is defined as:
typedef struct appCapabilities_tag
{
    uint8_t                 bUserStringSpecified   :1;
    uint8_t                 nrSupportedDeviceTypes :2;
    uint8_t                 reserved1              :1;
    uint8_t                 nrSupportedProfiles    :3;
    uint8_t                 reserved2              :1;
}appCapabilities_t;
```

where bUserStringSpecified indicates whether the requesting device has a defined user string which will be transmitted on the air, while nrSupportedDeviceTypes and nrSupportedProfiles indicate the sizes of origDeviceTypeList and origProfileIdList.

The origDeviceTypeList and origProfileIdList must contain at least one element each.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request has been accepted for processing by the network layer
- `gNWNodeNotStarted_c` — The request is rejected as the node is not started yet
- `gNWDenied_c` — The request is rejected as the network layer is already processing another non-interruptible request
- `gNWNoMemory_c` — The request is rejected as the network needs to allocate messages from the common message pool which is empty
- `gNWFrameCounterExpired_c` — The request is rejected as the frame counter has reached its maximum value
- `gNWInvalidParam_c`:
  - The request is rejected as origAppCapabilities.nrSupportedDeviceTypes is equal to 0
  - The request is rejected as origAppCapabilities.nrSupportedProfiles is equal to 0

Functional Description

The NLME_DiscoveryResponse is an asynchronous API function available both for controller and target nodes. It makes a request for a BeeStack Consumer node to respond after it has received a discovery request command.

This function call requests the starting of a BeeStack Consumer CommStatus process. The CommStatus process includes the Discovery response and Pair response subprocesses.

If the return value is `gNWSuccess_c`, the BeeStack Consumer layer has accepted and already started to process the CommStatus (Discovery response). When the CommStatus (Discovery response) process will be completed, the application layer will be notified by a CommStatus indication message which will be sent by BeeStack Consumer layer trough the BeeStack Consumer NLME SAP.

If the return value is not `gNWSuccess_c`, the BeeStack Consumer CommStatus process will not be started. At this point, the request is considered to be completed and therefore the application should not wait for any CommStatus indication message to be received later.

3.2.5 NLME_PairRequest

Prototype

```
uint8_t NLME_PairRequest
(
        uint8_t           recipChannel,
        uint8_t*          recipPanId,
        uint8_t*          recipMacAddress,
        appCapabilities_t origAppCapabilities,
        uint8_t*          origDeviceTypeList,
        uint8_t*          origProfileIdList,
        uint8_t           keyExTransferCount
);
```

Arguments

Table 3-5 specifies the parameters for the NLME_PairRequest primitive.

Table 3-5. NLME_PairRequest Parameters

Name	Type	Valid range	Description
recipChannel	uint8_t	0x0f, 0x14 or 0x19	The logical channel of the device with which to pair.
recipPanId	uint8_t*	A valid PAN identifier (different of NULL)	The PAN identifier of the device with which to pair.
recipMacAddress	uint8_t*	A valid IEEE address (different of NULL)	The IEEE address of the device with which to pair.
origAppCapabilities	appCapabilities_t		The application capabilities of this node
origDeviceTypeList	uint8_t*	Each integer: 0x00 – 0xfe	The list of device types supported by this node
origProfileIdList	uint8_t*	Each integer: 0x00 – 0xff	The list of profile IDs supported by this node
keyExTransferCount	uint8_t	0x00 – 0xff	The number of transfers the target should use to exchange the link key with the pairing originator.

```
appCapabilities_t is defined as:
typedef struct appCapabilities_tag
{
    uint8_t                 bUserStringSpecified   :1;
    uint8_t                 nrSupportedDeviceTypes :2;
    uint8_t                 reserved1              :1;
    uint8_t                 nrSupportedProfiles    :3;
    uint8_t                 reserved2              :1;
}appCapabilities_t;
```

where bUserStringSpecified indicates whether the requesting device has a defined user string which will be transmitted on the air, while nrSupportedDeviceTypes and nrSupportedProfiles indicate the sizes of origDeviceTypeList and origProfileIdList.

The origDeviceTypeList and origProfileIdList must contain at least one element each.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request has been accepted for processing by the network layer
- `gNWNodeNotStarted_c` — The request is rejected as the node is not started yet
- `gNWDenied_c` — The request is rejected as the network layer is already processing another non-interruptible request
- `gNWNoMemory_c` — The request is rejected as the network needs to allocate messages from the common message pool which is empty
- `gNWNoOrigCapacity_c` — The request is rejected as there is no room in the originator pair table to add a new entry
- `gNWFrameCounterExpired_c` — The request is rejected as the frame counter has reached its maximum value
- `gNWInvalidParam_c`:
  - The request is rejected as recipChannel parameter is not in the valid range
  - The request is rejected as origAppCapabilities.nrSupportedDeviceTypes is equal to 0
  - The request is rejected as origAppCapabilities.nrSupportedProfiles is equal to 0

Functional Description

The NLME_PairRequest is an asynchronous API function available both for controller and target nodes. It makes a request for a BeeStack Consumer node to establish a connection link with another BeeStack Consumer node that has known network identifiers (panId and MAC address). The information exchanged during the PairRequest process can be defined as application level information, which is delivered from originator's application layer to the recipient's application layer, like capabilities, device type list or profile ID list.

This function call requests the starting of a BeeStack Consumer PairRequest process. If the return value is `gNWSuccess_c`, the BeeStack Consumer layer has accepted and already started to process the Pair request. When the PairRequest process will be completed, the application layer will be notified by a Pair confirm message which will be sent by BeeStack Consumer layer trough the BeeStack Consumer NLME SAP.

If the return value is not `gNWSuccess_c`, the BeeStack Consumer Pair Request process will not be started. At this point, the request is considered to be completed and therefore the application should not wait for any Pair Confirm message to be received later.

3.2.6 NLME_PairResponse

Prototype

```
uint8_t NLME_PairResponse
(
        uint8_t           status,
        uint8_t*          recipPanId,
        uint8_t*          recipMacAddress,
        appCapabilities_t origAppCapabilities,
        uint8_t*          origDeviceTypeList,
        uint8_t*          origProfileIdList,
        uint8_t           deviceId
);
```

Arguments

Table 3-6 specifies the parameters for the NLME_PairResponse primitive.

Table 3-6. NLME_PairResponse Parameters

Name	Type	Valid range	Description
status	uint8_t	gNWSuccess_c, gNWNoRecipCapacity_c, gNWNotPermitted_c	The status of the pairing request
recipPanId	uint8_t*	A valid PAN identifier (different of NULL)	The PAN identifier of the device requesting the pair
recipMacAddress	uint8_t*	A valid IEEE address (different of NULL)	The IEEE address of the device requesting the pair
origAppCapabilities	appCapabilities_t	-	The application capabilities of this node
origDeviceTypeList	uint8_t*	Each integer: 0x00 – 0xfe	The list of device types supported by this node
origProfileIdList	uint8_t*	Each integer: 0x00 – 0xff	The list of profile IDs supported by this node
deviceId	uint8_t	0x00 – 0xfe	The reference to the provisional pairing entry

```
appCapabilities_t is defined as:
typedef struct appCapabilities_tag
{
    uint8_t                 bUserStringSpecified   :1;
    uint8_t                 nrSupportedDeviceTypes :2;
    uint8_t                 reserved1              :1;
    uint8_t                 nrSupportedProfiles    :3;
    uint8_t                 reserved2              :1;
}appCapabilities_t;
```

where bUserStringSpecified indicates whether the requesting device has a defined user string which will be transmitted on the air, while nrSupportedDeviceTypes and nrSupportedProfiles indicate the sizes of origDeviceTypeList and origProfileIdList.

The origDeviceTypeList and origProfileIdList must contain at least one element each.

Returns

Possible return values and their significance:

- gNWSuccess_c — The request has been accepted for processing by the network layer
- gNWNodeNotStarted_c — The request is rejected as the node is not started yet
- gNWDenied_c — The request is rejected as the network layer is already processing another non-interruptible request
- gNWNoMemory_c — The request is rejected as the network needs to allocate messages from the common message pool which is empty
- gNWFrameCounterExpired_c — The request is rejected as the frame counter has reached its maximum value
- gNWInvalidParam_c:
  - The request is rejected as origAppCapabilities.nrSupportedDeviceTypes is equal to 0
  - The request is rejected as origAppCapabilities.nrSupportedProfiles is equal to 0
  - The request is rejected as status parameter is out of the valid range
  - The request is rejected as deviceId parameter is bigger than the maximum number of entries supported in the pairing table and is not equal to 0xFF
  - The request is rejected as deviceId parameter is equal to 0xFF and the status parameter is gNWSuccess_c
  - The request is rejected as deviceId parameter is not equal to 0xFF and recipMacAddress parameter is not the one received in the Pair indication message
  - The request is rejected as deviceId parameter is not equal to 0xFF and recipPanId parameter is not the one received in the Pair indication message

Functional Description

The NLME_PairResponse is an asynchronous API function available both for controller and target nodes. It makes a request for a BeeStack Consumer node to exchange information with an BeeStack Consumer node that previously initiated a pair request command. The information exchanged during the PairResponse process can be defined as application level information, which is delivered from the pair request recipient application layer to the pair request originator application layer, like capabilities, device type list or profile ID list.

This function call requests the starting of a BeeStack Consumer CommStatus process. The CommStatus process includes the Discovery response and Pair response subprocesses.

If the return value is gNWSuccess_c, the BeeStack Consumer layer has accepted and already started to process the CommStatus (Pair response). When the CommStatus (Pair response) process will be completed, the application layer will be notified by a CommStatus indication message which will be sent by BeeStack Consumer layer trough the BeeStack Consumer NLME SAP.

If the return value is not gNWSuccess_c, the BeeStack Consumer CommStatus process will not be started. At this point, the request is considered to be completed and therefore the application should not wait for any CommStatus indication message to be received later.

3.2.7 NLME_UnpairRequest

Prototype

```
uint8_t NLME_UnpairRequest
(
        uint8_t deviceId
);
```

Arguments

Table 3-7 specifies the parameters for the NLME_UnpairRequest primitive.

Table 3-7. NLME_UnpairRequest Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1)	The reference into the local pairing table that is to be removed.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request has been accepted for processing by the network layer
- `gNWNodeNotStarted_c` — The request is rejected as the node is not started yet
- `gNWDenied_c` — The request is rejected as the network layer is already processing another non-interruptible request
- `gNWNoMemory_c` — The request is rejected as the network needs to allocate messages from the common message pool which is empty
- `gNWDeviceIdNotPaired_c` — The request is rejected as no pair information exists at the location specified by deviceId
- `gNWFrameCounterExpired_c` — The request is rejected as the frame counter has reached its maximum value
- `gNWInvalidParam_c` — The request is rejected as deviceId parameter is not in the valid range

Functional Description

The NLME_UnpairRequest is an asynchronous API function available both for controller and target nodes. It makes a request for a BeeStack Consumer node to remove a pairing link with another device from its pairing table.

This function call requests the starting of a BeeStack Consumer UnpairRequest process. When the UnpairRequest process will be completed, the application layer will be noticed by an Unpair confirm message which will be sent by BeeStack Consumer layer trough the BeeStack Consumer NLME SAP.

If the return value is not `gNWSuccess_c`, the BeeStack Consumer Unpair Request process will not be started. At this point, the request is considered to be completed and therefore the application should not wait for any Unpair Confirm message to be received later.

3.2.8 NLME_UnpairResponse

Prototype

```
uint8_t NLME_UnpairResponse
(
        uint8_t deviceId
);
```

Arguments

Table 3-8 specifies the parameters for the NLME_UnpairResponse primitive.

Table 3-8. NLME_UnpairResponse Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries - 1)	The reference into the local pairing table of the entry that is to be removed.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The local pair table entry was successfully removed
- `gNWNodeNotStarted_c` — The request is rejected as the node is not started yet
- `gNWDeviceIdNotPaired_c` — The request is rejected as no pair information exists at the location specified by deviceId
- `gNWInvalidParam_c` — The request is rejected as deviceId parameter is out of the valid range

Functional Description

The NLME_UnpairResponse API function is available both for controller and target nodes.

It makes a request for a BeeStack Consumer node to remove the pairing link indicated by the deviceId parameter from the pairing table.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NLME_UnpairResponse is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.9 NLME_UpdateKeyRequest

Prototype

```
uint8_t NLME_UpdateKeyRequest
(
        uint8_t           deviceId,
        uint8_t*          newLinkKey
);
```

Arguments

Table 3-8 specifies the parameters for the NLME_UpdateKeyRequest primitive.

Table 3-9. NLME_UpdateKeyRequest Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries - 1)	The reference into the local pairing table of the entry to update the security key to.
newLinkKey	uint8_t*	Any value	Pointer to a 16 bytes memory location that contains the security key to be written in the pair table entry pointed by deviceId parameter

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The security key of the entry pointed by deviceId was successfully updated
- `gNWNotPermitted_c` — The request is rejected as either the local node or the node in the pair table do not support security
- `gNWDeviceIdNotPaired_c` — The request is rejected as no pair information exists at the location specified by deviceId
- `gNWInvalidParam_c` — The request is rejected as deviceId parameter is out of the pair table size range

Functional Description

The NLME_UpdateKeyRequest API function is available both for controller and target nodes.

It makes a request for a BeeStack Consumer node to update the security key of one of the nodes that is paired with.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NLME_UpdateKeyRequest is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.10 NLME_GetRequest

Prototype

```
uint8_t NLME_GetRequest
(
        uint8_t          nibAttribute,
        uint8_t          nibAttributeIndex,
        uint8_t*         nibAttributeValue
);
```

Arguments

Table 3-10 specifies the parameters for the NLME_GetRequest primitive.

Table 3-10. NLME_GetRequest Parameters

Name	Type	Valid range	Description
nibAttribute	uint8_t	0x60 – 0x71	The identifier of the NIB attribute to read
nibAttributeIndex	uint8_t	Attribute specific	Used only for gNwkNib_PairingTable_c attribute
nibAttributeValue	uint8_t*	Pointer (different of NULL)	A pointer to a location where the request will store the attribute value

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The requested NIB attribute is successfully retrieved
- `gNWInvalidIndex_c` — The request is rejected as nibAttributeIndex is out of range (used only for gNwkNib_PairingTable_c)
- `gNWUnsupportedAttribute_c` — The request is rejected as nibAttribute parameter is out of valid range

Functional Description

The NLME_GetRequest API function is available both for controller and target nodes.

It makes a request for a BeeStack Consumer node to obtain information about a desired NIB attribute. If no corresponding NIB attribute is found, the NLME returns `gNWUnsupportedAttribute_c`.

This function call does not request starting any BeeStack Consumer process and therefore it is synchronous. When the application layer receives the return value of the API call, the NLME_GetRequest is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.11 NLME_SetRequest

Prototype

```
uint8_t NLME_SetRequest
(
        uint8_t           nibAttribute,
        uint8_t           nibAttributeIndex,
        uint8_t*          nibAttributeValue
);
```

Arguments

Table 3-11 specifies the parameters for the NLME_SetRequest primitive.

Table 3-11. NLME_SetRequest Parameters

Name	Type	Valid range	Description
nibAttribute	uint8_t	0x60 – 0x71	The identifier of the NIB attribute to change
nibAttributeIndex	uint8_t	Attribute specific	Used only for gNwkNib_PairingTable_c attribute.
nibAttributeValue	uint8_t*	Pointer (different of NULL)	A pointer to a memory location where the new value for the NIB attribute is to be read from

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The requested NIB attribute is successfully written
- `gNWInvalidIndex_c` — The request is rejected as nibAttributeIndex is out of range (used only for gNwkNib_PairingTable_c)
- `gNWUnsupportedAttribute_c` — The request is rejected as nibAttribute parameter is out of valid range
- `gNWInvalidParam_c` — The request is rejected as the value indicated by the nibAttributeValue pointer is out of the valid range of the nibAttibut

Functional Description

The NLME_SetRequest API function is available both for controller and target nodes.

This function call does not request starting any BeeStack Consumer process and therefore it is synchronous. When the application layer receives the return value of the API call, the NLME_SetRequest is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

When any of the activePeriod or dutyCycle NIBs are set using NLME_SetRequest service, the network will make a call to NLME_RxEnableRequest having the activePeriod NIB as parameter.

3.2.12 NLME_RxEnableRequest

Prototype

```
uint8_t NLME_RxEnableRequest
(
        uint32_t          rxOnDuration
);
```

Arguments

Table 3-12 specifies the parameters for the NLME_RxEnableRequest primitive.

Table 3-12. NLME_RxEnableRequest Parameters

Name	Type	Valid range	Description
rxOnDuration	uint32_t	0x000000 – 0xffffff	The number of MAC symbols for which the receiver is to be enabled 0x000000 – disabled until further notice 0xffffff – enabled until further notice

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request to enable or disable the receiver was successful
- `gNWNodeNotStarted_c` — The request is rejected as the node is not started yet
- `gNWDenied_c` — The request is rejected as the network layer is already processing another request. As the NLME_RxEnable request will affect the state of the receiver, the previously started process can not be interrupted.
- `gNWInvalidParam_c` — The rxOnDuration parameter is outside the valid range.

Functional Description

The NLME_RxEnableRequest API function is available both for controller and target nodes.

It makes a request for a BeeStack Consumer node to disable or to enable (for a finite period or until further notice) the receiver. This function also allows alternating the periods when the receiver is open with the periods when the receiver is closed. Table 3-13 provides more details about the output of the NLME_RxEnableRequest() API call, depending on the value of the rxOnDuration parameter and on the value of the nwkDutyCycle NIB.

Table 3-13. NLME_RxEnableRequest Service Function Details

Input		Output
rxOnDuration parameter	**nwkDutyCycle NIB**	
0x00	0x00	Disable receiver until further notice. Deactivate power saving
0xFFFFFF	0x00	Enable receiver until further notice. Deactivate power saving
nwkActivePeriod NIB	0x00	Keep the receiver open for the value of rxOnDuration parameter. Deactivate power saving

Table 3-13. NLME_RxEnableRequest Service Function Details

Input		Output
rxOnDuration parameter	**nwkDutyCycle NIB**	
any other value	0x00	Keep the receiver open for the value of rxOnDuration parameter. Deactivate power saving
0x00	not 0x00	Disable receiver until further notice. Deactivate power saving
0xFFFFFF	not 0x00	Enable receiver until further notice. Deactivate power saving
nwkActivePeriod NIB	not 0x00	Activate power saving. Node starts to automatically open and close its receiver, based on the values of the nwkActivePeriod and nwkDutyCycle NIBs.
any other value	not 0x00	Freescale implementation specific behavior. Try to set the nwkActivePeriod NIB to the value of the rxOnDuration parameter. If the rxOnDuration parameter does not respect the validation conditions of the nwkActivePeriod NIB (larger or at least equal to and smaller or at most equal to nwkDutyCycle NIB), function returns gNWInvalidParam_c status. If nwkActivePeriod NIB is successfully set to the value of rxOnDuration parameter, power saving mode is activated. Node starts to automatically open and close its receiver, based on the values of the nwkActivePeriod and nwkDutyCycle NIBs.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NLME_RxEnableRequest is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.13 NLME_ResetRequest

Prototype

```
uint8_t NLME_ResetRequest
(
        bool_t            bSetDefaultNib
);
```

Arguments

Table 3-14 specifies the parameters for the NLME_ResetRequest primitive.

Table 3-14. NLME_ResetRequest Parameters

Name	Type	Valid range	Description
bSetDefaultNib	bool_t	TRUE, FALSE	If TRUE, the NWK layer is resetted and all NIB attributes are set to their default values. If FALSE, the NWK layer is resetted but all NIB attributes retain their values prior to the generation of the NLMERESET.request primitive.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request to reset the NWK layer is successful

Functional Description

The NLME_ResetRequest API function is available both for controller and target nodes.

It makes a request for a BeeStack Consumer node to reset the network layer (setting the NIB attributes to their default values or retaining their values prior the reset request).

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NLME_ResetRequest is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

NOTE

Issuing this primitive with the bSetDefaultNIB parameter set to TRUE removes all entries from the pairing table.Issuing this primitive with the bSetDefaultNIB parameter set to FALSE fills the NIBs (including the pair table) with their last saved values in the non volatile memory.

3.2.14 NLDE_DataRequest

Prototype

```
uint8_t NLDE_DataRequest
(
        uint8_t          deviceId,
        uint8_t          profileId,
        uint8_t*         vendorId,
        uint8_t          nsduLength,
        uint8_t*         nsdu,
        uint8_t          txOptions
);
```

Arguments

Table 3-15 specifies the parameters for the NLDE_DataRequest primitive.

Table 3-15. NLDE_DataRequest Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries - 1)	The pairing reference of the destination device. Ignored for broadcast transmissions.
profileId	uint8_t	0x00 – 0xff	The ID of the profile describing the data format
vendorId	uint8_t*	0x0000 or a valid vendor identifier (pointer different of NULL)	If the TxOptions parameter specifies that the data is vendor specific, this parameter specifies the vendor identifier. If this parameter is equal to 0x0000, the vendor identifier shall be set to nwkcVendorIdentifier. If the TxOptions parameter specifies that the data is not vendor specific this parameter is ignored.
nsduLength	uint8_t	0 – 90	The length of the payload (in bytes) to be transmitted.

Table 3-15. NLDE_DataRequest Parameters

nsdu	uint8_t*	-	A pointer to the payload to be transmitted.
txOptions	uint8_t	-	Transmission options for this NSDU. For b0 (transmission mode): 1 = broadcast transmission 0 = unicast transmission For b1 (destination addressing mode): 1 = use destination IEEE address 0 = use destination network address For b2 (acknowledgement mode): 1 = acknowledged transmission 0 = unacknowledged transmission For b3 (security mode): 1 = transmit with security 0 = transmit without security For b4 (channel agility mode): 1 = use single channel operation 0 = use multiple channel operation For b5 (channel normalization mode): 1 = specify channel designator 0 = do not specify channel designator For b6 (payload mode): 1 = data is vendor specific 0 = data is not vendor specific

The transmission options are defined in the `NwkInterface.h` header file, as follows:

```
/* Transmission options */
#define maskTxOptions_Broadcast_c                (1<<0)
#define maskTxOptions_UseRecipLongAddress_c      (1<<1)
#define maskTxOptions_UseAck_c                   (1<<2)
#define maskTxOptions_UseSecurity_c              (1<<3)
#define maskTxOptions_UseOneChannelOnly_c        (1<<4)
#define maskTxOptions_UseChannelDesignator_c     (1<<5)
#define maskTxOptions_VendorSpecificData_c       (1<<6)
```

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request has been accepted for processing by the network layer
- `gNWNodeNotStarted_c` — The request is rejected as the node is not started yet
- `gNWDenied_c` — The request is rejected as the network layer is already processing another non-interruptible request
- `gNWNoMemory_c` — The request is rejected as the network needs to allocate messages from the common message pool which is empty
- `gNWDeviceIdNotPaired_c` — The request is rejected as a unicast transmission is requested and no pair information exists at the location specified by deviceId
- `gNWFrameCounterExpired_c` — The request is rejected as the frame counter has reached its maximum value
- `gNWInvalidParam_c`:

— The request is rejected as a unicast transmission is used and the deviceId parameter is out of the valid range
— The request is rejected as profileId parameter is out of the valid range
— The request is rejected as nsduLength parameter is out of the valid range
— The request is rejected as there is no data payload to be transmitted but the nsduLength parameter is bigger than 0
— The request is rejected as the transmission is secured but the capabilities of the node indicate that the node doesn't support security

Functional Description

The NLDE_DataRequest is an asynchronous API function available both for controller and target nodes. It makes a request for a BeeStack Consumer node to transfer an application data unit to one of the nodes it is already paired with.

This function call requests the starting of a BeeStack Consumer DataRequest process. If the return value is `gNWSuccess_c`, the BeeStack Consumer layer has accepted and already started to process the DataRequest request. When the DataRequest process will be completed, the application layer will be notified by a Data confirm message which will be sent by BeeStack Consumer layer trough the BeeStack Consumer NLDE SAP.

If the return value is not `gNWSuccess_c`, the BeeStack Consumer Data Request process will not be started. At this point, the request is considered to be completed and therefore the application should not wait for any Data Confirm message to be received later.

3.2.15 NWK_GetNodePanId

Prototype

```
#define NWK_GetNodePanId()        nodeData.localPanId
```

Arguments

The NWK_GetNodePanId macro has no parameters.

Returns

Possible return values and their significance:

- Will return a pointer to a two bytes array containing the IEEE 802.15.4 PAN identifier of the local node

Functional Description

This macro is available on both controller and target nodes.

On receipt of NWK_GetNodePanId macro, the BeeStack Consumer layer returns a pointer to a 2 bytes location containing the IEEE 802.15.4 PAN identifier of the local node.

3.2.16 NWK_ GetNodeShortAddress

Prototype

```
#define NWK_GetNodeShortAddress() nodeData.localShortAddress
```

Arguments

The NWK_GetNodeShortAddress macro has no parameters.

Return value

Possible return values and their significance:

- Will return a pointer to a two bytes array containing the IEEE 802.15.4 short address of the local node

Functional Description

This macro is available on both controller and target nodes.

On receipt of NWK_GetNodeShortAddress macro, the BeeStack Consumer layer returns a pointer to a 2 bytes location containing the IEEE 802.15.4 short address of the local node.

3.2.17 NWK_GenerateShortAddress

Prototype

```
void NWK_GenerateShortAddress
(
        uint8_t*          shortAddress
);
```

Arguments

Table 3-16 specifies the parameters for the NWK_GenerateShortAddress primitive.

Table 3-16. NWK_GenerateShortAddress Parameters

Name	Type	Valid range	Description
shortAddress	uint8_t*	-	This is an output parameter for a 2 byte memory location (provided by the application) that will contain the new generated network address.

Returns

- This function has no return value.

Functional Description

The NWK_GenerateShortAddress API function is available both for controller and target nodes. It makes a request for a BeeStack Consumer node to generate an IEEE 802.15.4 short address that does not match any of the short addresses of the entries already in the pair table.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NWK_GenerateShortAddress request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.18 NWK_GenerateSecurityKey

Prototype

```
void NWK_GenerateSecurityKey
(
        uint8_t*          securityKey
);
```

Arguments

Table 3-17 specifies the parameters for the NWK_GenerateSecurityKey primitive.

Table 3-17. NWK_GenerateSecurityKey Parameters

Name	Type	Valid range	Description
securityKey	uint8_t*	-	This is an output parameter for a 16 byte memory location (provided by the application) containing the new generated security key.

Returns

- This function has no return value.

Functional Description

The NWK_GenerateSecurityKey API function is available both for controller and target nodes. It makes a request for a BeeStack Consumer node to generate a security key that does not match any of the security keys of the existing entries in the pair table.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NWK_GenerateSecurityKey request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.19 NWK_AddNewPairTableEntry

Prototype

```
uint8_t NWK_AddNewPairTableEntry
(
        uint8_t*            localShortAddress,
        uint8_t             recipChannel,
        uint8_t*            recipMacAddress,
        uint8_t*            recipPanId,
        uint8_t*            recipShortAddress,
        uint8_t             recipCapabilities,
        uint8_t*            securityKey,
        uint8_t*            recipUserString
);
```

Arguments

Table 3-18 specifies the parameters for the NWK_ AddNewPairTableEntry primitive.

Table 3-18. NWK_ AddNewPairTableEntry Parameters

Name	Type	Valid range	Description
localShortAddress	uint8_t*	A valid short address (pointer different of NULL)	The network address to be assumed by the local device. This is an input parameter.
recipChannel	uint8_t	15, 20, 25	The expected channel of the new paired device.
recipMacAddress	uint8_t*	A valid 802.15.04 IEEE address (pointer different of NULL)	The IEEE address of the new paired device. This is an input parameter.
recipPanId	uint8_t*	A valid PAN identifier (pointer different of NULL)	The PAN identifier of the new paired device. This is an input parameter.
recipShortAddress	uint8_t*	A valid short address (pointer different of NULL)	The network address of the new paired device. This is an input parameter.
recipCapabilities	uint8_t	-	The node capabilities of the new paired device.
securityKey	uint8_t*	A valid security key (pointer different of NULL)	The link key to be used to secure the paired link. This is an input parameter.
recipUserString	uint8_t*	A valid user string (pointer different of NULL)	The user string of the new paired device. This is an input parameter.

Returns

Possible return values and their significance:

- the index in the pair table corresponding to the new added entry
- `gNWNoOrigCapacity_c`: the request is rejected as there is no room in the pair table to add a new entry
- `gNWInvalidParam_c`: the request is rejected as recipChannel parameter is out of the valid range

Functional Description

The NWK_AddNewPairTableEntry API function is available both for controller and target nodes.

It makes a request for a BeeStack Consumer node to add a new paired link in its pair table, without starting a pair process. The pairing information must be provided by the application.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NWK_AddNewPairTableEntry request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.20 NWK_SavePersistentData

Prototype

```
void NWK_SavePersistentData
(
        void
);
```

Arguments

- This function has no arguments.

Returns

- This function has no return value.

Functional Description

The NWK_SavePersistentData API function is available both for controller and target nodes.

It makes a request for a BeeStack Consumer node to save the BeeStack Consumer sensitive information into the non volatile memory. This sensitive information includes the pair table (nodeData) and the NIB table (gNwkNib). This information will not be written in FLASH during the function call. The function call will only mark this information as being pending to be written in FLASH. The actual write in FLASH operation will be executed in the Idle Task of the application, the first time it is run.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NWK_SavePersistentData request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.21 NWK_ SaveFrameCounter

Prototype

```
uint8_t NWK_SaveFrameCounter
(
        void
);
```

Arguments

- This function has no arguments.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request for saving of the nwkFrameCounter NIB attribute into FLASH was successfully marked as pending
- `gNWNodeNotStarted_c` — The request is rejected as the node is not started yet

Functional Description

The NWK_SaveFrameCounter API function is available both for controller and target nodes, but only after the node is started.

It makes a request for a BeeStack Consumer node to save the nwkFrameCounter NIB attribute into Flash. The pair table and the rest of the NIB table will not be updated in Flash. This information will not be written in FLASH during the function call. The function call will only mark this information as being pending to be written in FLASH. The actual write in FLASH operation will be executed in the Idle Task of the application, the first time it is run.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NWK_SaveFrameCounter request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.22 NWK_SetMacAddress

Prototype

```
uint8_t NWK_SetMacAddress(uint8_t* pMacAddress);
```

Arguments

Table 3-19 specifies the parameters for the NWK_SetMacAddress primitive.

Table 3-19. NWK_SetMacAddress Parameters

Name	Type	Valid range	Description
pMacAddress	uint8_t*	Valid IEEE MAC address	The IEEE address to be used by the local device. This is an input parameter.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The request to set the MAC address of the local node to a specific value was successful
- `gNWInvalidParam_c` — The request is rejected as pMacAddress parameter is NULL

Functional Description

The NWK_SetMacAddress API function is available both for controller and target nodes and should always be called after the resetting the node, but before starting it.

It makes a request for a BeeStack Consumer node to set the MAC address of the local node to a specific value. The IEEE MAC address must be provided by the application.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NWK_SetMacAddress request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.23 NWK_GetMacAddress

Prototype

```
uint8_t NWK_GetMacAddress(uint8_t* pMacAddress);
```

Arguments

Table 3-20 specifies the parameters for the NWK_GetMacAddress primitive.

Table 3-20. NWK_GetMacAddress Parameters

Name	Type	Valid range	Description
pMacAddress	uint8_t*	-	This is an output parameter for a 8 byte memory location (provided by the application) containing the IEEE MAC address of the local node.

Returns

Possible return values and their significance:

- `gNWSuccess_c` — The IEEE MAC address of the local node was successfully obtained
- `gNWInvalidParam_c` — The request is rejected as pMacAddress parameter is NULL

Functional Description

The NWK_GetMacAddress API function is available both for controller and target nodes.

It makes a request to retrieve the local node's IEEE 802.15.4 extended address into a specific location.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NWK_GetMacAddress request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.24 NWK_GetLastPacketLQI

Prototype

```
uint8_t NWK_GetLastPacketLQI(void);
```

Arguments

- This function has no arguments.

Returns

Possible return values and their significance:

- LQI of the last received network packet

Functional Description

The NWK_GetLastPacketLQI API function is available both for controller and target nodes.

It makes a request for a BeeStack Consumer node to get the LQI of the last received packet. This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NWK_GetLastPacketLQI request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.25 NWK_GetAllowedLowPowerInterval

Prototype

```
uint32_t NWK_GetAllowedLowPowerInterval(void);
```

Arguments

- This function has no arguments.

Returns

Possible return values and their significance:

- The amount of time (in MAC symbols) the network can safely can enter low power mode from the moment this call is made:
- `0x00000000`: the BeeStack Consumer network can not enter low power mode at the moment this request was made. This can happen due to the following reasons:
- The network is not idle (it is executing a process)
- The network is idle, but the receiver is set to be always ON. The network will not allow the platform to enter low power mode while the receiver is open
- The receiver is set to work in intermittent mode but the function call is made during the active period of the nwkDutyCycle, when the receiver is open, therefore the network do not allow the entering of a low power mode
- `0x00FFFFFF`: the network is idle and the receiver is set to be always OFF. In this case the network allows entering low power mode for 0x00FFFFFF MAC symbols (approximately 268.4 seconds), which is the maximum interval the LPM (low power mode) platform component can be configured to enter low power
- The amount of time (in MAC symbols) remaining from the moment the call is made until the next activePeriod is to begin: the network is idle, the receiver is set to work in intermittent mode and the function call is made during the inactive period of the nwkDutyCycle

Functional Description

The NWK_GetAllowedLowPowerInterval API function is available both for controller and target nodes.

It makes a request for the BeeStack Consumer layer to inform the calling entity about the availability time interval of the network to enter low power mode.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the NWK_GetAllowedLowPowerInterval request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.2.26 NWK_IsIdle

Prototype

```
bool_t  NWK_IsIdle(void);
```

Arguments

- This function has no arguments.

Returns

Possible return values and their significance:

- `TRUE` — The BeeStack Consumer network is in idle state
- `FALSE` — The BeeStack Consumer network is not in idle state, but is busy executing a process

Functional Description

The NWK_IsIdle API function is available both for controller and target nodes.

This function is used to determine if the BeeStack Consumer layer is in the idle state or not.

This function call does not request the starting of any BeeStack Consumer process and for this reason its call is synchronous. When the application layer returns from the API call, the NWK_IsIdle request is completely executed. There will be no other confirm message sent by BeeStack Consumer trough the BeeStack Consumer NLME or NLDE SAPs.

3.3 Message Data Types

The interface between the Network Layer Management Entity (NLME) as well as the Network Layer Data Entity (NLDE) and the Application Layer is based on service primitives passed from the network to the application through a Service Access Point (SAP). Two SAPs must be implemented as functions in the application:

void NWK_NLME_SapHandler(nwkNlmeToAppMsg_t* nwkNlmeToAppMsg)

void NWK_NLDE_SapHandler(nwkNldeToAppMsg_t* nwkNldeToAppMsg)

A message is the entity passed from the network layer to the application layer as a SAP parameter. The application typically stores the received messages in message queues. A message queue decouples the execution context which ensures that the call stack does not build up between modules when communicating. The decoupling also ensures that timing critical modules can queue a message to less timing critical modules and move on, which ensures that the receiving module does process the message immediately.

Both NLME and NLDE service primitives use the same type of messages as defined in the `NwkInterface.h` file. Because the NLME and NLDE interfaces are based on messages being passed to the SAPs, each message needs to have its own identifier. These identifiers are shown in the following tables.

Table 3-21. Primitives in the NLME to Application Direction

Message identifier	BeeStack Consumer NLME to Application Primitives	Available on Controller	Available on Target	Section
gNwkNlmeStartCnf_c	NLME-START.Confirm	X	X	3.3.1
gNwkNlmeAutoDiscoveryCnf_c	NLME-AUTO-DISCOVERY.Confirm	X	X	3.3.2
gNwkNlmeDiscoveryCnf_c	NLME-DISCOVERY.Confirm	X	X	3.3.3
gNwkNlmeDiscoveryInd_c	NLME- DISCOVERY.Indication	X	X	3.3.4
gNwkNlmePairCnf_c	NLME-PAIR.Confirm	X	X	3.3.5
gNwkNlmePairInd_c	NLME- PAIR.Indication	X	X	3.3.6
gNwkNlmeUnpairCnf_c	NLME- UNPAIR.Confirm	X	X	3.3.7
gNwkNlmeUnpairInd_c	NLME- UNPAIR.Indication	X	X	3.3.8
gNwkNlmeCommStatusInd_c	NLME-COMM-STATUS.Indication	X	X	3.3.9

Table 3-22. Primitives in the NLDE to Application Direction

Message identifier	BeeStack Consumer NLDE to Application Primitives	Available on Controller	Available on Target	Sub-clause
gNwkNldeDataCnf_c	NLDE-DATA.Confirm	X	X	3.3.10
gNwkNldeDataInd_c	NLDE-DATA.Indication	X	X	3.3.11

This section describes the main C-structures and data types used by the NLME/NLDE interface.

The structures used to describe the BeeStack Consumer confirm and indications have been collected in single message type as unions, plus a message type that corresponds to the enumerations of the primitives. These are the structures which transport messages through the SAPs.

For messages from NLME to application the following structure/union is used:

```
/* General structure of a message received by the application over NLME SAP */
typedef struct nwkNlmeToAppMsg_tag
{
  nwkNlmeToAppMsgType_t           msgType;
  union {
    nwkNlmeStartCnf_t       nwkNlmeStartCnf;
    nwkNlmeAutoDiscoveryCnf_t     nwkNlmeAutoDiscoveryCnf;
    nwkNlmeDiscoveryCnf_t         nwkNlmeDiscoveryCnf;
    nwkNlmeDiscoveryInd_t         nwkNlmeDiscoveryInd;
    nwkNlmePairCnf_t              nwkNlmePairCnf;
    nwkNlmePairInd_t              nwkNlmePairInd;
    nwkNlmeUnpairCnf_t            nwkNlmeUnpairCnf;
```

```
    nwkNlmeUnpairInd_t            nwkNlmeUnpairInd;
    nwkNlmeCommStatusInd_t        nwkNlmeCommStatusInd;
  } msgData;
}nwkNlmeToAppMsg_t;
```

Where `nwkNlmeToAppMsgType_t` is:

```
/* Messages used for informing the application about confirms or indications from BeeStack
Consumer arrived trough the NLME SAP */
typedef enum {
  gNwkNlmeStartCnf_c = 0,
  gNwkNlmeAutoDiscoveryCnf_c,
  gNwkNlmeDiscoveryCnf_c,
  gNwkNlmeDiscoveryInd_c,
  gNwkNlmePairCnf_c,
  gNwkNlmePairInd_c,
  gNwkNlmeUnpairCnf_c,
  gNwkNlmeUnpairInd_c,
  gNwkNlmeCommStatusInd_c,
  gNwkNlmeMax_c
}nwkNlmeToAppMsgType_t;
```

For messages from NLDE to application the following structure/union is used:

```
/* General structure of a message received by the application over NLDE SAP */
typedef struct nwkNldeToAppMsg_tag
{
  nwkNldeToAppMsgType_t    msgType;
  union
  {
    nwkNldeDataCnf_t       nwkNldeDataCnf;
    nwkNldeDataInd_t       nwkNldeDataInd;
  } msgData;
}nwkNldeToAppMsg_t;
```

Where `nwkNldeToAppMsgType_t` is:

```
/* Messages used for informing the application about confirms or indications from BeeStack
Consumer arrived trough the NLDE SAP */
typedef enum {
  gNwkNldeDataCnf_c       = 0,
  gNwkNldeDataInd_c,
  gNwkNldeMax_c
}nwkNldeToAppMsgType_t;
```

A detailed description of each BeeStack Consumer message is presented in the following sections.

3.3.1 nwkNlmeStartCnf_t

Message Structure

The semantics of the nwkNlmeStartCnf_t message is as follows:

```
typedef struct nwkNlmeStartCnf_tag
{
        uint8_t           status;
}nwkNlmeStartCnf_t;
```

Structure Members

Table 3-23 specifies the fields available in the nwkNlmeStartCnf_t message structure.

Table 3-23. nwkNlmeStartCnf_t message structure

Name	Type	Valid range	Description
Status	uint8_t	gNWSuccess_c, gNWNoMemory_c or any other MAC status returned by the MLME-SCAN.confirm, MLME-START.confirm or MLME-SET.confirm primitives	The status of the BeeStack Consumer Start process.

Possible values of the status:

- `gNWSuccess_c` — The Start process has completed successfully
- `gNWNoMemory_c` — The Start process has failed to complete as the network needs to allocate messages from the common message pool which is empty

A detailed description for the MAC status possible values can be found in R[2].

Functional Description

The Start Confirm message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that a BeeStack Consumer Start process has completed and also offers valuable information about the way this request has been accomplished.

For details on how the BeeStack Consumer Start process is initiated, see Section 3.2.1, "NLME_StartRequest".

3.3.2 nwkNlmeAutoDiscoveryCnf_t

Message Structure

The semantics of the nwkNlmeAutoDiscoveryCnf_t message are as follows:

```
typedef struct nwkNlmeAutoDiscoveryCnf_tag
{
        uint8_t          status;
        uint8_t          origMacAddress[8];
        uint8_t          origPanId[2];
}nwkNlmeAutoDiscoveryCnf_t;
```

Structure Members

Table 3-24 specifies the fields available in the nwkNlmeAutoDiscoveryCnf_t message structure.

Table 3-24. nwkNlmeAutoDiscoveryCnf_t message structure

Name	Type	Valid range	Description
Status	uint8_t	gNWSuccess_c, gNWDiscoveryTimeout_c, gNWAborted_c , gNWNoMemory_c , gNWDiscoveryError_c or any other MAC status returned by the MCPS-Data.confirm primitive	The status of the BeeStack Consumer Auto Discovery process.
origMacAddress	uint8_t array with 8 elements	a valid MAC address	The IEEE address to which the discovery response was sent
origPanId	uint8_t array with 2 elements	a valid PAN Id	The PAN Id to which the discovery response was sent

Possible values of the status:

- `gNWSuccess_c` — The Auto Discovery process has completed successfully
- `gNWNoMemory_c` — The Auto Discovery process is ended as the network needs to allocate messages from the common message pool which is empty
- `gNWDiscoveryTimeout_c` — The Auto Discovery process is ended because a Discovery request with parameters matching the auto discovery parameters was not received during the autoDiscDuration time interval
- `gNWAborted_c` — The Auto Discovery process is ended because another asynchronous request was initiated. Even if the new request was ended returning an error status, the Auto Discovery process is still ended.
- `gNWDiscoveryError_c` — The Auto Discovery process is ended because two discovery requests from two different nodes were received during the autoDiscDuration time

A detailed description for the MAC status possible values can be found in R[2]

Functional Description

The Auto Discovery Confirm message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that a BeeStack Consumer Auto Discovery process has completed and also offers valuable information about the way this request has been accomplished.

For details on how the BeeStack Consumer Auto Discovery process is initiated, see Section 3.2.2, “NLME_AutoDiscoveryRequest”.

3.3.3 nwkNlmeDiscoveryCnf_t

Message Structure

The semantics of the nwkNlmeDiscoveryCnf_t message are as follows:

```
typedef struct nwkNlmeDiscoveryCnf_tag
{
        uint8_t                  status;
        uint8_t                  nrDiscoveredNodes;
        nodeDescriptorBlock_t*   pNodeDescriptorBlocks;

}nwkNlmeDiscoveryCnf_t;

#define aNodeDescriptorsPerBlock                          2

struct nodeDescriptorBlock_tag {
  nodeDescriptor_t              nodeDescriptorList[aNodeDescriptorsPerBlock];
  uint8_t                       nodeDescriptorCount;
  struct nodeDescriptorBlock_tag* pNext;
};

typedef struct nodeDescriptor_tag
{
    uint8_t                 status;
    uint8_t                 recipChannel;
    uint8_t                 recipPanId[2];
    uint8_t                 recipMacAddress[8];
    uint8_t                 recipCapabilities;
    uint8_t                 recipVendorId[2];
    uint8_t                 recipVendorString[gSizeOfVendorString_c];
    appCapabilities_t       recipAppCapabilities;
    uint8_t                 recipUserString[gSizeOfUserString_c];
    uint8_t                 recipDeviceTypeList[gMaxNrOfNodeDeviceTypes_c];
    uint8_t                 recipProfilesList[gMaxNrOfNodeProfiles_c];
    uint8_t                 requestLQI;
}nodeDescriptor_t;

typedef struct nodeDescriptorBlock_tag nodeDescriptorBlock_t;
```

Structure Members

Table 3-25 specifies the fields available in the nwkNlmeDiscoveryCnf_t message structure.

Table 3-25. nwkNlmeDiscoveryCnf_t message structure

Name	Type	Valid range	Description
Status	uint8_t	gNWSuccess_c, gNWAborted_c , gNWNoMemory_c , gNWDiscoveryTimeout_c, gNWDiscoveryError_c or any other MAC status returned by the MCPS-Data.confirm primitive	The status of the BeeStack Consumer DiscoveryRequest process.

Table 3-25. nwkNlmeDiscoveryCnf_t message structure

nrDiscoveredNodes	uint8_t	0x00 – 0x04 (0x00 - nwkcMaxNodeDescListSize)	The number of node descriptors discovered
pNodeDescriptorBlocks	nodeDescriptorBlock_t	-	Pointer to a list of node descriptors discovered.

Possible values of the status:

- `gNWSuccess_c` — The DiscoveryRequest process has completed successfully
- `gNWAborted_c` — The DiscoveryRequest process is ended because another asynchronous request was initiated. Even if the new request completed, returning an error status, the DiscoveryRequest process is still ended
- `gNWNoMemory_c` — The DiscoveryRequest process is ended as the network needs to allocate messages from the common message pool which is empty
- `gNWDiscoveryTimeout_c` — The DiscoveryRequest process is ended because no Discovery responses were received during the discDuration time interval
- `gNWDiscoveryError_c` — The DiscoveryRequest process ended because more than nwkcMaxNodeDescListSize (4) distinct discovery responses were received during the discDuration time

A detailed description for the MAC status possible values can be found in R[2]. For details on `nodeDescriptor_t` see R[1].

Functional Description

The Discovery Confirm message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that a BeeStack Consumer DiscoveryRequest process has completed and also offers valuable information about the way this request has been accomplished.

For details on how the BeeStack Consumer DiscoveryRequest process is initiated, see Section 3.2.3, “NLME_DiscoveryRequest”.

3.3.4 nwkNlmeDiscoveryInd_t

Message Structure

The semantics of the nwkNlmeDiscoveryInd_t message are as follows:

```
typedef struct nwkNlmeDiscoveryInd_tag
{
        uint8_t                 status;
        uint8_t                 origMacAddress[8];
        uint8_t                 origCapabilities;
        uint8_t                 origVendorId[2];
        uint8_t*                origVendorString;
        appCapabilities_t       origAppCapabilities;
        uint8_t*                origUserString;
        uint8_t*                origDeviceTypeList;
        uint8_t*                origProfilesList;
        uint8_t                 requestedDeviceType;
        uint8_t                 rxLinkQuality;
}nwkNlmeDiscoveryInd_t;
```

Structure Members

Table 3-26 specifies the fields available in the nwkNlmeDiscoveryInd_t message structure.

Table 3-26. nwkNlmeDiscoveryInd_t message structure

Name	Type	Valid range	Description
Status	uint8_t	gNWSuccess_c, gNWNoRecipCapacity_c	The status of the pairing table
origMacAddress	uint8_t array with 8 elements	A valid IEEE address	The IEEE address of the device requesting the discovery
origCapabilities	uint8_t	-	The node capabilities of the originator of the discovery request
origVendorId	uint8_t array with 2 elements	A valid Vendor ID	The vendor identifier of the originator of the discovery request
origVendorString	uint8_t*	7 octets	The vendor string of the originator of the discovery request.
origAppCapabilities	appCapabilities_t	-	The application capabilities of the originator of the discovery request
origUserString	uint8_t*	NULL or 15 characters	The user string of the originator of the discovery request
origDeviceTypeList	uint8_t*	Each integer: 0x00 – 0xfe	The list of device types supported by the originator of the discovery request
origProfileIdList	uint8_t*	Each integer: 0x00 – 0xff	The list of profile IDs supported by the originator of the discovery request
requestedDeviceType	uint8_t	0x00 – 0xff	The device type being discovered. 0xff indicates any device
rxLinkQuality	uint8_t	0x00 – 0xff	The link quality of the discovery request, as passed by the MAC

appCapabilities_t is defined as:

```
typedef struct appCapabilities_tag
{
        uint8_t                  bUserStringSpecified   :1;
        uint8_t                  nrSupportedDeviceTypes :2;
        int8_t                  reserved1              :1;
        uint8_t                  nrSupportedProfiles    :3;
        int8_t                  reserved2              :1;
}appCapabilities_t;
```

where bUserStringSpecified indicates whether the discovery requesting device has a defined user string, while nrSupportedDeviceTypes and nrSupportedProfiles indicate the sizes of origDeviceTypeList and origProfileIdList.

Possible values of the status:

- `gNWSuccess_c` — A free entry exists in the pair table
- `gNWNoRecipCapacity_c` — There is no free entry in the pair table

Functional Description

The NLME_Discovery Indication message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that a BeeStack Consumer DiscoveryRequest command was received.

On receipt of the NLME_Discovery Indication message, the application decides whether to respond (using a NLME_DiscoveryResponse) based on the information contained in the primitive. If the application decides not to respond, no primitive is issued.

3.3.5 nwkNlmePairCnf_t

Message Structure

The semantics of the nwkNlmePairCnf_t message are as follows:

```
typedef struct nwkNlmePairCnf_tag
{
        uint8_t                   status;
        uint8_t                   deviceId;
        uint8_t                 recipVendorId[2];
        uint8_t*                recipVendorString;
        appCapabilities_t       recipAppCapabilities;
        uint8_t*                recipUserString;
        uint8_t*                recipDeviceTypeList;
        uint8_t*                recipProfilesList;
}nwkNlmePairCnf_t;
```

Structure Members

Table 3-27 specifies the fields available in the nwkNlmePairCnf_t message structure.

Table 3-27. nwkNlmePairCnf_t message structure

Name	Type	Valid range	Description
Status	uint8_t	gNWSucces_c, gNWNoMemory_c, gNWNoResponse_c, gNWNoRecipCapacity_c, gNWNotPermitted_c, gNWSecurityTimeout_c, gNWSecurityFailure_c, or a status value from the MCPS-DATA.confirm primitive	The status of the BeeStack Consumer PairRequest process.
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1), 0xff	The pairing table reference for this pairing link (will be 0xff if status is not gNWSuccess_c)
recipVendorId	uint8_t array with 2 elements	A valid vendor identifier	The vendor ID of the pair response originator
recipVendorString	uint8_t*	A pointer to a 7 bytes array	The vendor string of the pair response originator
recipAppCapabilities	appCapabilities_t	-	The application capabilities of the pair response originator
recipUserString	uint8_t*	NULL or 15 characters	The user defined identification string of the pair response originator
recipDeviceTypeList	uint8_t*	Each integer: 0x00 – 0xfe	The list of device types supported by the pair response originator
recipProfileIdList	uint8_t*	Each integer: 0x00 – 0xfe	The list of profile IDs supported by the pair response originator

```
appCapabilities_t is defined as:
typedef struct appCapabilities_tag
{
```

```
        uint8_t                 bUserStringSpecified   :1;
        uint8_t                 nrSupportedDeviceTypes :2;
        uint8_t                 reserved1              :1;
        uint8_t                 nrSupportedProfiles    :3;
        uint8_t                 reserved2              :1;
}appCapabilities_t;
```

where bUserStringSpecified indicates whether the pair response originator has a defined user string, while nrSupportedDeviceTypes and nrSupportedProfiles indicate the sizes of recipDeviceTypeList and recipProfileIdList.

Possible values of the status:

- `gNWSuccess_c` — The PairRequest process has completed successfully
- `gNWNoResponse_c` — No pair response was received from the device the pair request was sent to
- `gNWNotPermitted_c` — The target device has denied the pair request
- `gNWNoRecipCapacity_c` — The target device has no room in the pair table to accept the pair request
- `gNWSecurityTimeout_c` — Process is ended as the security link key seeds transfer was not completed during ((n+1) * nwkcMaxKeySeedWaitTime) time interval
- `gNWSecurityFailure_c` — The process is ended as a seed with invalid sequence number was received
- `gNWNoMemory_c` — The PairRequest process is ended as the network needs to allocate messages from the common message pool which is empty

A detailed description for the MAC status possible values can be found in R[2].

Functional Description

The Pair Confirm message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that a BeeStack Consumer PairRequest process has completed and also offers valuable information about the way this request has been accomplished.

For details on how the BeeStack Consumer PairRequest process is initiated, see Section 3.2.5, “NLME_PairRequest”.

3.3.6 nwkNlmePairInd_t

Message Structure

The semantics of the nwkNlmePairInd_t message are as follows:

```
typedef struct nwkNlmePairInd_tag
{
        uint8_t                 status;
        uint8_t                 origPanId[2];
        uint8_t                 origMacAddress[8];
        uint8_t                 origCapabilities;
        uint8_t                 origVendorId[2];
        uint8_t*                origVendorString;
        appCapabilities_t       origAppCapabilities;
        uint8_t*                origUserString;
        uint8_t*                origDeviceTypeList;
        uint8_t*                origProfilesList;
        uint8_t                    keyExTransferCount;
        uint8_t                 deviceId;
}nwkNlmePairInd_t;
```

Structure Members

Table 3-28 specifies the fields available in the nwkNlmePairInd_t message structure.

Table 3-28. nwkNlmePairInd_t message structure

Name	Type	Valid range	Description
Status	uint8_t	gNWSuccess_c, gNWNoRecipCapacity_c, gNWDuplicatePairing_c	The status of the provisional pairing
origPanId	uint8_t array with 2 elements		The PAN identifier of the device requesting the pair
origMacAddress	uint8_t array with 8 elements	A valid IEEE address	The IEEE address of the device requesting the pair
origCapabilities	uint8_t	-	The node capabilities of the device requesting the pair
origVendorId	uint8_t array with 2 elements	A valid Vendor ID	The Vendor Id of the device requesting the pair
origVendorString	uint8_t*	7 octets	The Vendor string of the device requesting the pair
origAppCapabilities	appCapabilities_t	-	The application capabilities of the device requesting the pair
origUserString	uint8_t*	NULL or 15 characters	The user defined identification string of the device requesting the pair
origDeviceTypeList	uint8_t*	Each integer: 0x00 – 0xfe	The list of device types supported by the device requesting the pair
origProfileIdList	uint8_t*	Each integer: 0x00 – 0xff	The list of profile IDs supported by the device requesting the pair.

Table 3-28. nwkNlmePairInd_t message structure

keyExTransferCount	uint8_t	0x00 – 0xff	The number of transfers the originator of the pair requested this node to use to exchange the link key if it accepts the pair request.
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1), 0xff	Next free pairing reference that will be used if this pairing request is successful. If this value is equal to 0xff, the NWK layer has no free pairing table entries.

```
appCapabilities_t is defined as:
typedef struct appCapabilities_tag
{
        uint8_t                 bUserStringSpecified   :1;
        uint8_t                 nrSupportedDeviceTypes :2;
        uint8_t                 reserved1              :1;
        uint8_t                 nrSupportedProfiles    :3;
        uint8_t                 reserved2              :1;
}appCapabilities_t;
```

where bUserStringSpecified indicates whether the originating pair request device has a defined user string, while nrSupportedDeviceTypes and nrSupportedProfiles indicate the sizes of origDeviceTypeList and origProfileIdList.

Possible values of the status:

- `gNWSuccess_c` — A new provisional entry containing the information about the pair requesting device was created
- `gNWNoRecipCapacity_c` — No room in the pair table to create an entry containing the information about the pair requesting device
- `gNWDuplicatePairing_c` — The information about the pairing requesting device already exists in the pair table, in the entry pointed by the deviceId field

Functional Description

The Pair Indication message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that a BeeStack Consumer PairRequest command was received.

On receipt of the Pair Indication message, the application decides whether to respond (using a NLME_PairResponse) based on the information contained in the primitive. If the application decides not to respond, no primitive is issued.

3.3.7 nwkNlmeUnpairCnf_t

Message Structure

The semantics of the nwkNlmeUnpairCnf_t message are as follows:

```
typedef struct nwkNlmeUnpairCnf_tag
{
        uint8_t                 status;
        uint8_t                 deviceId;
}nwkNlmeUnpairCnf_t;
```

Structure Members

Table 3-29 specifies the fields available in the nwkNlmeUnpairCnf_t message structure.

Table 3-29. nwkNlmeUnpairCnf_t message structure

Name	Type	Valid range	Description
Status	uint8_t	gNWSucces_c, gNWNoMemory_c or any other status value from the MCPS-DATA.confirm primitive	The status of the BeeStack Consumer UnpairRequest process.
deviceId	uint8_t	0x00 – 0xff	The pairing table reference for this pairing link. In case the status is not gNWSuccess_c this parameter is not valid.

Possible values of the status:

- `gNWSuccess_c` — The UnpairRequest process has completed successfully
- `gNWNoMemory_c` — The UnpairRequest process is ended as the network needs to allocate messages from the common message pool which is empty

A detailed description for the MAC status possible values can be found in R[2].

Functional Description

The Unpair Confirm message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that a BeeStack Consumer UnpairRequest process has completed and also offers valuable information about the way this request has been accomplished. Even if the status is not gNWSucces_c, due to the fact that the MCPS-DATA.confirm reported an error in the transmission of the UnpairRequest frame, the entry is still removed from the pairing table.

For details on how the BeeStack Consumer UnpairRequest process is initiated, see Section 3.2.7, “NLME_UnpairRequest”.

3.3.8 nwkNlmeUnpairInd_t

Message Structure

The semantics of the nwkNlmeUnpairInd_t message are as follows:

```
typedef struct nwkNlmeUnpairInd_tag
{
        uint8_t                 deviceId;
}nwkNlmeUnpairInd_t;
```

Structure Members

Table 3-30 specifies the fields available in the nwkNlmeUnpairInd_t message structure.

Table 3-30. nwkNlmeUnpairInd_t message structure

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1)	The pairing table reference that has been removed from the pairing table.

Functional Description

The Unpair indication message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that a BeeStack Consumer Unpair request command was received.

On receipt of the Unpair indication message, the application decides whether to respond (using a NLME_UnpairResponse) based on the information contained in the primitive. If the application decides not to respond, no primitive is issued.

3.3.9 nwkNlmeCommStatusInd_t

Message Structure

The semantics of the nwkNlmeCommStatusInd_t message are as follows:

```
typedef struct nwkNlmeCommStatusInd_tag
{
        uint8_t          deviceId;
        uint8_t          targetPanId[2];
        uint8_t          targetAddressMode;
        uint8_t          targetAddress[8];
        uint8_t          status;
}nwkNlmeCommStatusInd_t;
```

Structure Members

Table 3-31 specifies the fields available in the nwkNlmeCommStatusInd_t message structure.

Table 3-31. nwkNlmeCommStatusInd_t message structure

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1), 0xff	Reference into the pairing table indicating the recipient node. A value of 0xff indicates that a discovery response frame was sent.
targetAddress	uint8_t array with 8 elements	A valid IEEE/short address (depending of the targetAddressMode)	The address of the destination device. If the short address is used, only the first two bytes of the array are valid.
targetPanId	uint8_t array with 2 elements	Valid PAN identifier	The PAN ID of the destination device
targetAddressMode	uint8_t	0, 1	The type of the address: 1 – 64 bit IEEE address 0 – 16 bit short address
Status	uint8_t	gNWSuccess_c, gNWNoMemory_c, gNWSecurityFailure_c, gNWSecurityTimeout_c, or any other status value from the MCPS-DATA.confirm primitive	The status of the transmission.

Possible values of the status:

- `gNWSuccess_c` — The BeeStack Consumer Pair response or the BeeStack Consumer Discovery were successfully sent
- `gNWNoMemory_c` — The CommStatus process is ended as the network needs to allocate messages from the common message pool which is empty
- `gNWSecurityFailure_c` — The CommStatus process is ended as a BeeStack Consumer Pair response was sent, but no Ping request was received during nwkResponseWaitTime time period, or because the received Ping request was not valid
- `gNWSecurityTimeout_c` — The CommStatus process is ended as the security link key seeds transfer was not completed during ((n+1) * nwkcMaxKeySeedWaitTime) time interval

A detailed description for the MAC status possible values can be found in R[2]

Functional Description

The Comm Status indication message can be received by the application layer both on controller and target nodes.

This message notifies the application layer about the status of a BeeStack Consumer communication (the status of sending discovery response or pair response).

For details on how the BeeStack Consumer CommStatus process is initiated, see Section 3.2.6, “NLME_PairResponse” and Section 3.2.4, “NLME_DiscoveryResponse”.

3.3.10 nwkNldeDataCnf_t

Message Structure

The semantics of the nwkNldeDataCnf_t message are as follows:

```
typedef struct nwkNldeDataCnf_tag
{
        uint8_t           status;
        uint8_t           deviceId;
        uint8_t           profileId;
}nwkNldeDataCnf_t ;
```

Structure Members

Table 3-32 specifies the fields available in the nwkNldeDataCnf_t message structure.

Table 3-32. nwkNldeDataCnf_t message structure

Name	Type	Valid range	Description
Status	uint8_t	gNWSucces_c, gNWNoResponse_c, gNWNoMemory_c, gNWAborted_c or any other status value from the MCPS-DATA.confirm primitive	The status of the BeeStack Consumer DataRequest process.
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1)	The pairing table reference for the NSDU being confirmed
profileId	uint8_t	0x00 - 0xFE	The profileId that was provided in the NLDE Data Request for which the confirm is received

Possible values of the status:

- `gNWSuccess_c` — The DataRequest process has completed successfully
- `gNWNoResponse_c` — The DataRequest process has failed as no MAC ack was received from the recipient device, even though the data transmission was acknowledged
- `gNWNoMemory_c` — The DataRequest process has failed as the network needs to allocate messages from the common message pool which is empty
- `gNWAborted_c` — The DataRequest process is interrupted by another request that needs to start a process.

A detailed description for the MAC status possible values can be found in R[2].

Functional Description

The Data confirm message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that a BeeStack Consumer DataRequest process previously requested has completed and also offers valuable information about the way this request has been accomplished.

For details on how the BeeStack Consumer DataRequest process is initiated, see Section 3.2.14, “NLDE_DataRequest”.

3.3.11 nwkNldeDataInd_t

Message Structure

The semantics of the nwkNldeDataInd_t message are as follows:

```
typedef struct nwkNldeDataInd_tag
{
        uint8_t                 deviceId;
        uint8_t                 profileId;
        uint8_t                 vendorId[2];
        uint8_t                 LQI;
        uint8_t                 rxFlags;
        uint8_t                 dataLength;
        uint8_t*                pData;
}nwkNldeDataInd_t;
```

Structure Members

Table 3-33 specifies the fields available in the nwkNldeDataInd_t message structure.

Table 3-33. nwkNldeDataInd_t message structure

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTable Entries - 1), 0xff	Reference into the pairing table which matched the information contained in the received NSDU. A value of 0xff indicates that a broadcast frame was received that does not correspond to an entry in the pairing table.
profileId	uint8_t	0x00 – 0xff	The identifier of the profile indicating the format of the received data.
vendorId	uint8_t array with two elements	A valid vendor identifier	If the RxFlags parameter specifies that the data is vendor specific, this parameter specifies the vendor identifier. If the RxFlags parameter specifies that the data is not vendor specific this parameter is ignored.
LQI	uint8_t	0x00 – 0xff	LQI value measured during reception of the NSDU
rxFlags	uint8_t	-	Reception indication flags for this NSDU. For bB0B (reception mode): 1 = received as broadcast 0 = received as unicast For bB1B (security mode): 1 = received with security 0 = received without security For bB2B (payload mode): 1 = data is vendor specific 0 = data is not vendor specific
pData	uint8_t*	-	A pointer to the received payload data
dataLength	uint8_t	0 – 90	The length of the received payload

The reception indication flags are defined in the `NwkInterface.h` header file, as follows:

```
/* Reception options */
#define maskRxOptions_Broadcast_c                (1<<0)
#define maskRxOptions_UseSecurity_c              (1<<1)
#define maskRxOptions_VendorSpecificData_c       (1<<2)
```

Functional Description

The Data indication message can be received by the application layer both on controller and target nodes.

This message notifies the application layer that an application data has just been received and also offers valuable information related to the sender of the packet and the way this packet has been received.

3.4 BeeStack Consumer Data Types

3.4.1 NodeData Database

NodeData Database is a structure defined in RAM memory that keeps information about the current node (the PAN identifier and the short address of the current node) and the pair table.

NodeData Database is a structure defined in the RAM memory which keeps MAC related information about the current node (the PAN identifier and the short address) and also information about the nodes that are paired with it. The information in this table is modified during some of the BeeStack Consumer processes such as Start, Pair, Unpair, Set and Reset, and each time it is changed, the whole structure is saved in NVM. When a node is started, the application can choose if it wants to restore the NodeData Database structure from NVM, or if it wants a clean version of it, by setting the `gNwkNib_StartWithNetworkInfoFromFlash_c` NIB accordingly.

The NodeData Database type definition can be found in the `NwkInterface.h` header file.

```
/* NodeData Database structure */
typedef struct nodeData_tag
{
        uint8_t                     localPanId[2];
        uint8_t                     localShortAddress[2];
        pairTableEntry_t            pairTableEntry[gMaxPairingTableEntries_c];
} nodeData_t;

/* Structure of information kept inside an entry in the pair table of a BeeStack Consumer node */
typedef struct pairTableEntry_tag
{
        uint8_t                 localShortAddress[2];
        uint8_t                 recipChannel;
        uint8_t                 recipMacAddress[8];
        uint8_t                 recipPanId[2];
        uint8_t                 recipShortAddress[2];
        uint8_t                 recipCapabilities;
        uint32_t                recipFrameCounter;
        uint8_t                 securityKey[gSizeOfSecurityKey_c];
    /* Fields that do not appear in the spec but present in the FSL implementation of BeeStack
Consumer */
    uint8_t                 recipUserString[gSizeOfUserString_c];
}pairTableEntry_t;

/* Define the number of the entries in the node's pair table */
#ifndef gMaxPairingTableEntries_c
  #define gMaxPairingTableEntries_c                     5
#endif
A copy of the NodeData structure among with the NIB attributes table is also kept in FLASH, by
the NVM platform component.
```

```
/* This data set contains network layer variables to be preserved across resets */
NvDataItemDescription_t const gaNvNwkDataSet[] = {
  {(uint8_t*)&nodeData, sizeof(nodeData_t)},
  {(uint8_t*)&gNwkNib,  sizeof(nwkNib_t)},
  {NULL, 0}       /* Required end-of-table marker. */
};
```

Table 3-34 specifies the fields available in the nodeData_t message structure.

Table 3-34. nodeData_t message structure

Name	Type	Valid range	Description
localPanId	uint8_t array with two elements	Valid PAN identifier	The PAN identifier of the current node. In case the current node type is controller, the PAN identifier will be set to 0
localShortAddress	uint8_t array with two elements	Valid short address	The short address of the current node. In case the current node type is controller, the short address will be set to 0
pairTableEntry	pairTableEntry_t	-	The entries in the pair table. Freescale BeeStack Consumer implementation supports up to 16 entries in the pair table when running on MC1323x platform and up to 8 entries in the pair table when running on MC1320x, MC1321x and MC1322x platforms.

Table 3-35 specifies the fields available in the pairTableEntry_t message structure.

Table 3-35. pairTableEntry_t message structure

Name	Type	Valid range	Description
localShortAddress	uint8_t array with 2 elements	Valid short address	The short address of the current node in the PAN of the target device
recipChannel	uint8_t	15, 20, 25	The channel the target device was paired on
recipMacAddress	uint8_t array with 8 elements	Valid IEEE address	the IEEE address of the target device
recipPanId	uint8_t array with 2 elements	Valid PAN identifier	the PAN ID of the target device
recipShortAddress	uint8_t array with 2 elements	Valid short address	the short address of the target device
recipCapabilities	uint8_t	-	the node capabilities of the target device
recipFrameCounter	uint32_t	0x00000000 – 0xffffffff	the frameCounter of the latest network packet received from the target device
securityKey	uint8_t array with 16 elements	-	the security key for the paring link. In case no security is used, the securityKey will be set to 0.
recipUserString	uint8_t array with 15 elements	-	the user string of the target device

3.4.2 BeeStack Consumer NIBs

The NIB comprises attributes required to manage the NWK layer of a device. The attributes contained in the NIB are shown in the following table:

Table 3-36. NIB Attributes

Name	Type	Identifier	Description
nwkActivePeriod	uint32_t	gNwkNib_ActivePeriod_c	The active period of a device in MAC symbols
nwkBaseChannel	uint8_t	gNwkNib_BaseChannel_c	The logical channel that was chosen during device initialization
nwkDiscoveryLQIThreshold	uint8_t	gNwkNib_DiscoveryLQIThreshold_c	The LQI threshold below which discovery requests will be rejected
nwkDiscoveryRepetitionInterval	uint32_t	gNwkNib_DiscoveryRepetitionInterval_c	The interval, in MAC symbols, at which discovery attempts are made on all channels
nwkDutyCycle	uint16_t	gNwkNib_DutyCycle_c	The duty cycle of a device in MAC symbols. A value of 0x000000 indicates the device is not using power saving
nwkFrameCounter	uint32_t	gNwkNib_FrameCounter_c	The frame counter added to the transmitted NPDU
nwkIndicateDiscoveryRequests	bool_t	gNwkNib_IndicateDiscoveryRequests_c	Indicates whether the NLME indicates the reception of discovery request command frames to the application. TRUE indicates that the NLME notifies the application
nwkInPowerSave	bool_t	gNwkNib_InPowerSave_c	The power save mode of the node. TRUE indicates that the device is operating in power save mode
nwkPairingTable	uint8_t	gNwkNib_PairingTable_c	The pairing table managed by the device
nwkMaxDiscoveryRepetitions	uint8_t	gNwkNib_MaxDiscoveryRepetitions_c	The maximum number of discovery attempts made at the nwkDiscoveryRepetitionInterval rate

Table 3-36. NIB Attributes (continued)

nwkMaxFirstAttemptCSMABackoffs	uint8_t	gNwkNib_MaxFirstAttemptCSMABackoffs_c	The maximum number of backoffs the MAC CSMA-CA algorithm will attempt before declaring a channel access failure for the first transmission attempt
nwkMaxFirstAttemptFrameRetries	uint8_t	gNwkNib_MaxFirstAttemptFrameRetries_c	The maximum number of MAC retries allowed after a transmission failure for the first transmission attempt
nwkMaxReportedNodeDescriptors	uint8_t	gNwkNib_MaxReportedNodeDescriptors_c	The maximum number of node descriptors that can be obtained before reporting to the application
nwkResponseWaitTime	uint32_t	gNwkNib_ResponseWaitTime_c	The maximum time in MAC symbols, a device shall wait for a response command frame following a request command frame
nwkScanDuration	uint8_t	gNwkNib_ScanDuration_c	A measure of the duration of a scanning operation, according to [R2]
nwkUserString	uint8_t	gNwkNib_UserString_c	The user defined character string used to identify this node
nwkStartWithNetworkInfoFromFlash	bool_t	gNwkNib_StartWithNetworkInfoFromFlash_c	Freescale specific attribute. Indicates whether during the start process the network layer variables stored in NVM (nodeData and gNwkNib) will be restored from NVM or not. TRUE indicates that the variables will be restored with NVM data. FALSE indicates that the node will start with an empty pair table and with the NIB table filled with the default values specified by the standard.

Table 3-36. NIB Attributes (continued)

nwkAcceptDataInAutoDiscoveryMode	bool_t	gNwkNib_AcceptDataInAutoDiscoveryMode_c	Kept only for interface backwards compatibility with older BeeStack Consumer stacks. Setting it to any of the TRUE or FALSE values will have no effect on the network behavior. The NLDE Data.Indication packets are always forwarded by the network to the application no matter the process that is currently taking place (Ex: pair, discovery, AutoDiscovery, Idle)
nwkKeySeedPALevel	uint8_t	gNwkNib_KeySeedPALevel_c	Freescale specific attribute. Allows the upper layer to set the power level used for transmitting the key seeds during the pair process to either a smaller or a bigger value than the one specified by the ZigBee RF4CE standard: -15dBm. This NIB can be set to any uint8_t value in the range [0,10]. Default value is 5. More details about the relation between the values of the NIB and the values of the output power are presented in Table 3-37.

Table 3-36. NIB Attributes (continued)

nwkKeepPairInfoAtDuplicatePairFail	bool_t	gNwkNib_KeepPairInfoAtDuplicatePairFail_c	Freescale specific attribute. Allows the upper layer to request the network not to delete the pair information in case when the pair is done for an entry already in the pair table and the pair process fails. Available on both controller and target nodes. Default value is FALSE.
nwkFADisabled	bool_t	gNwkNib_FADisabled_c	Freescale specific attribute. Allows the upper layer to request the network to disable the Frequency Agility module and remain locked on one single channel. This channel is pointed by gNwkNib_BaseChannel_c NIB. If this NIB is set to TRUE then: 1. All the multi-channel transmissions will be performed on the channel indicated by the gNwkNib_BaseChannel_c NIB 2. The node will remain locked on the channel indicated by gNwkNib_BaseChannel_c NIB even if it will receive data frames with channelDesignator option requesting it to switch to another channel. 3. The node will remain locked on the channel indicated by gNwkNib_BaseChannel_c NIB even this channel is experiences heavy traffic or radio interference conditions. The recommended sequence for having a node work on one single channel is: 1. Set the gNwkNib_FADisabled_c NIB to TRUE. 2. Set the gNwkNib_BaseChannel_c NIB to the one the network should work on (one of the 15, 20 or 25 values)

```
/* Network NIB attribute identifiers */
typedef enum
{
```

```
        gNwkNib_FirstAttribute_c = 0x60,
        gNwkNib_ActivePeriod_c   = 0x60,
        gNwkNib_BaseChannel_c,
        gNwkNib_DiscoveryLQIThreshold_c,
        gNwkNib_DiscoveryRepetitionInterval_c,
        gNwkNib_DutyCycle_c,
        gNwkNib_FrameCounter_c,
        gNwkNib_IndicateDiscoveryRequests_c,
        gNwkNib_InPowerSave_c,
        gNwkNib_PairingTable_c,
        gNwkNib_MaxDiscoveryRepetitions_c,
        gNwkNib_MaxFirstAttemptCSMABackoffs_c,
        gNwkNib_MaxFirstAttemptFrameRetries_c,
        gNwkNib_MaxReportedNodeDescriptors_c,
        gNwkNib_ResponseWaitTime_c,
        gNwkNib_ScanDuration_c,
         gNwkNib_UserString_c,
  /* Freescale defined NIB attributes */
        gNwkNib_StartWithNetworkInfoFromFlash_c,
        gNwkNib_AcceptDataInAutoDiscoveryMode_c,
        gNwkNib_KeySeedPALevel_c,
        gNwkNib_KeepPairInfoAtDuplicatePairFail_c,
        gNwkNib_FADisabled_c,
        gNwkNib_LastAttribute_c
} nwkNibAttribute_t;

/* Network NIB table structure */
typedef struct nwkNib_tag
{
        uint32_t  nwkActivePeriod;
        uint8_t   nwkBaseChannel;
        uint8_t   nwkDiscoveryLQIThreshold;
        uint32_t  nwkDiscoveryRepetitionInterval;
        uint16_t  nwkDutyCycle;
        uint32_t  nwkFrameCounter;
        bool_t    nwkIndicateDiscoveryRequests;
        bool_t    nwkInPowerSave;
        uint8_t   nwkPairingTable;
        uint8_t   nwkMaxDiscoveryRepetitions;
        uint8_t   nwkMaxFirstAttemptCSMABackoffs;
        uint8_t   nwkMaxFirstAttemptFrameRetries;
        uint8_t   nwkMaxReportedNodeDescriptors;
        uint32_t  nwkResponseWaitTime;
        uint8_t   nwkScanDuration;
        uint8_t   nwkUserString[15];
  /* Freescale defined NIB attributes */
        bool_t    nwkStartWithNetworkInfoFromFlash;
        bool_t    nwkAcceptDataInAutoDiscoveryMode;
        uint8_t   nwkKeySeedPALevel;
        bool_t    nwkKeepPairInfoAtDuplicatePairFail;
        bool_t    nwkFADisabled;
} nwkNib_t;
```

Table 3-37. gNwkNib_KeySeedPALevel_c NIB Details

gNwkNib_KeySeedPALevel_c	BeeStack Consumer On S08/QE128 Platform	BeeStack Consumer On ARM7 Platform	BeeStack Consumer On MC1323x
0	-28.7 dBm	-28 dBm	-30.63 dBm
1	-22 dBm	-21 dBm	-28.52 dBm
2	-18.5 dBm	-19 dBm	-25.30 dBm
3	-16.2 dBm	-17 dBm	-22.77 dBm
4	-15.9 dBm	-16 dBm	-19.02 dBm
5	-15.3 dBm	-15 dBm	-15.88 dBm
6	-8.5 dBm	-10 dBm	- 9.37 dBm
7	-7.0 dBm	-4.5 dBm	- 7.07 dBm
8	-1.6 dBm	-1.5 dBm	- 4.26 dBm
9	-0.66 dBm	-1 dBm	0.20 dBm
10	1.42 dBm	1.7 dBm	2.30 dBm

Table 3-38. gNwkNib_KeySeedPALevel_c NIB Details

gNwkNib_KeySeedPALevel_c	BeeStack Consumer On S08/QE128 Platform	BeeStack Consumer On ARM7 Platform	BeeStack Consumer On MC1323x Platform
0	-28.7 dBm	-28 dBm	-24.9 dBm
1	-22 dBm	-21 dBm	-22.3 dBm
2	-18.5 dBm	-19 dBm	-21.0 dBm
3	-16.2 dBm	-17 dBm	-19.7 dBm
4	-15.9 dBm	-16 dBm	-17.1 dBm
5	-15.3 dBm	-15 dBm	-14.5 dBm
6	-8.5 dBm	-10 dBm	-10.6 dBm
7	-7.0 dBm	-4.5 dBm	- 9.3 dBm
8	-1.6 dBm	-1.5 dBm	- 8.0 dBm
9	-0.66 dBm	-1 dBm	0.0 dBm
10	1.42 dBm	1.7 dBm	3.0 dBm

3.4.3 Saving BeeStack Consumer Sensitive Information in FLASH

The management of saving information in FLASH is handled by the NVM platform component. This component handles 2 independent data sets, one for the application specific data and another one for the network specific data. Each data set can store and retrieve from FLASH up to 504 bytes of information.

The BeeStack Consumer network sensitive information that needs to be preserved between resets of the board consists of the nodeData structure and the NIB table. While the NIB table has a fixed size, the size

of the nodeData structure depends on the number of entries in the pair table. The maximum number of entries in the pair table is derived by the limitation that the size of the nodeData plus the size of the NIB table should not exceed the following:

- 504 bytes on the MC1320x, MC1321x and MC1322x platforms
- 1016 bytes on the MC1323x platform

The maximum number of entries in the pair table of a BeeStack Consumer node is as follows:

- When the network runs on a MC1320x, MC1321x or MC1322x platform this value is 8
- When the network runs on a MC1323x platform this value is 16

The NVM component can only handle the save in FLASH of a whole data set. This means that even if the application has changed only a NIB value, the whole NIB table plus nodeData structure will be saved in FLASH, and not only that particular NIB. However, there is one exception from this rule. Considering that during the life cycle of a BeeStack Consumer product the value that will probably be the most often written in FLASH is the frameCounter NIB, a special mechanism was designed to allow saving it without actually having to update the whole network data set in FLASH. The mechanism uses the remaining space in the network data set (504 bytes – size of nodeData – size of NIB table) to consecutively store in FLASH the values of the frameCounter NIB, each time they are updated. When there is no room left in the 504 bytes space of the network data set to retain a new update of the frameCounter NIB, the whole network dataSet is updated in FLASH. This mechanism minimizes the number of FLASH pages writes that a BeeStack Consumer application will trigger during its life time cycle.

The BeeStack Consumer network will request the saving of the whole network data set information in FLASH in the following situations:

- After calling the NLME_Start service if:
  - The start was issued without persistent information from FLASH. This is done to save the pair table in FLASH that was cleared during the start process
  - The start was issued for the first time after the board was programmed. This is done to create the network dataSet in FLASH
- After a call to NLME_PairResponse, if the pair process was completed successfully. This is done to save the updated pair table in FLASH
- After a call to NLME_UnpairRequest call. This is done to save the updated pair table in FLASH
- After a call to NLME_UnpairResponse call. This is done to save the updated pair table in FLASH.
- After a call to NLME_SetRequest, only if the NIB is not NwkNib_FrameCounter_c and also different from its previous version. Setting a NIB to the same value it had before does not trigger the save of the NIB table in FLASH.
- After a call to NWK_AddNewPairTableEntry call. This is done to save the updated pair table in FLASH.
- After a call to NWK_SavePersistentData call. This is done because it is explicitly requested by the application.
- After a call to NLME_RxEnableRequest, only if the value of any of the nwkActivePeriod or nwkInPowerSave NIBs has changed. This allows and automatic start of an intermittent Rx

(inPowerSave) operation after a reset with NIB table loaded from FLASH in the case where the node was previously in intermittent Rx (inPowerSave) mode.

The BeeStack Consumer network requests the saving of the frameCounter NIB in the following situations:

- After a call to NLME_SetRequest when NIB is equal to gNwkNib_FrameCounter_c.
- After a call to NLME_Start call using persistent information from FLASH, as the frameCounter is incremented with nwkcFrameCounterWindow and the new value is saved back in FLASH.
- When the frameCounter reaches the nwkcFrameCounterWindow value.
- After a call to NWK_SaveFrameCounter call.

Chapter 4
BeeStack Consumer/SynkroRF Hybrid Software Overview

This chapter provides a brief overview of the ZigBee RF4CE (BeeStack Consumer)/SynkroRF hybrid network software.

4.1 Network Topology

The BeeStack Consumer/SynkroRF hybrid node allows a Freescale BeeStack Consumer network and a SynkroRF network to coexist and communicate with each other. A hybrid node can be part of either type of network.

4.2 Node Functionality Enumeration

The basic node types for the Freescale BeeStack Consumer/SynkroRF hybrid network software are as follows:

- **Controller Node -** This device contains a subset of the BeeStack Consumer/SynkroRF hybrid features. The controller node type is used in remote control applications. This node does not start a PAN. During the pairing process, it receives a Pan Id and Short Address from the target node it has paired with, whether it is paired with an BeeStack Consumer node or a SynkroRF node. These values are used in future communications with the paired target node.
- **Target Node -** This device contains a subset the BeeStack Consumer/SynkroRF hybrid features. The target node type is used in Consumer Electronics device applications (e.g. TVs, DVDs, etc.). It is intended to be used in backbone powered devices. Starting a target node always starts a new PAN. A target node is able to accept pair requests from both BeeStack Consumer and SynkroRF nodes. The same PAN is used for both BeeStack Consumer and for SynkroRF communications.

NOTE

A hybrid device must operate as the same type of node in either type of network. For example, it is not possible to operate as an BeeStack Consumer controller and a SynkroRF target (controlled device), or an BeeStack Consumer target and a SynkroRF controller.

4.3 Hybrid Available Libraries

This section describes the libraries available for the BeeStack Consumer/SynkroRF hybrid software stack. Table 4-1 describes each of the two available libraries.

Table 4-1. BeeStack Consumer/SynkroRF Hybrid Network Libraries

Library Type	Description	Typical Usage
BeeStackConsumer_SynkroHybrid.lib	Contains all BeeStack Consumer/SynkroRF hybrid features.	Should be used by applications exercising all network features. Includes API parameters validation.
BeeStackConsumer_SynkroHybrid_ZTC.lib	Contains all BeeStack Consumer/SynkroRF hybrid features and also supports monitoring the interface between the MAC and the network from the ZTC application	Should be used in the development and validation phase. The added support for monitoring the stack interfaces increases the code size requirements.

Chapter 5
BeeStack Consumer/SynkroRF Hybrid Network Layer Interface Description

This chapter describes the BeeStack Consumer/SynkroRF hybrid network layer interface.

5.1 General Hybrid Network Interface Information

The BeeStack Consumer/SynkroRF hybrid network layer adds new primitives to the existing ZigBee RF4CE Network Layer Management Entities (NLME) and Network Layer Data Entities (NLDE). No new network layer entities were created, but several new utility functions were added.

The SynkroRF functionality is initialized at the same time as the ZigBee RF4CE functionality. When the application calls the ZigBee RF4CE NLME_Start service, the hybrid network layer starts the SynkroRF node as well as the ZigBee RF4CE node. A call to ZigBee RF4CE NLME_Reset service also resets the SynkroRF node.

5.2 System API

This section describes the additional structures and functions available in the System API of a BeeStack Consumer/SynkroRF hybrid node. To use the API, the `NwkInterface.h` header file must be included in the relevant source code files.

Table 5-1. SynkroRF Hybrid API (Application to Network Layer Direction)

Hybrid API Function Name		Description	Over the air Activity	Synchronous Call	Available on Controller	Available on Target	Section
NLME	SynkroRF_PairRequest	This function call initiates the controller side SynkroRF pairing process	X		X		5.2.1
	SynkroRF_PairResponse	This function call initiates the target side SynkroRF pairing process	X			X	5.2.2
	SynkroRF_ClearPairingInformation	This function removes a pairing entry with a SynkroRF device		X	X	X	5.2.3
NLDE	SynkroRF_SendCommand	This function call initiates the command transmission process.	X		X	X	5.2.4

Table 5-1. SynkroRF Hybrid API (Application to Network Layer Direction)

Hybrid API Function Name		Description	Over the air Activity	Synchronous Call	Available on Controller	Available on Target	Section
Freescale Implementation Specific Utility Services	SynkroRF_AddnewPairTableEntry	This function allows the application to insert 'offline' the information of a new SynkroRF entry in the pair table, without initiating a pair process.		X	X	X	5.2.5
	SynkroRF_GetPairedDeviceInfo	This function allows the application to obtain a copy of the pair table entry of a SynkroRF device.		X	X	X	5.2.6
	NWK_GetNodeType	This function allows the application to get the (BeeStack Consumer or SynkroRF) used for a particular entry in the pairing		X	X	X	5.2.7

All SynkroRF hybrid API calls return a code to report on their operation. No new return codes are defined for SynkroRF hybrid nodes. Instead, the same codes as defined by the ZigBee RF4CE specification are used.

The SynkroRF pairing processes started by SynkroRF_PairRequest and SynkroRF_PairResponse are uninterruptable. The command transmission process is interruptable.

The following sections provide a detailed description of each hybrid SynkroRF API function.

5.2.1 SynkroRF_PairRequest

Prototype

```
uint8_t SynkroRF_PairRequest(
                            uint8_t     deviceType,
                            uint8_t*    pPairingData,
                            uint8_t     length,
                            uint16_t    timeOut
                          );
```

Arguments

Table 5-2 specifies the parameters for the SynkroRF_PairRequest primitive.

Table 5-2. NLME_PairRequest Parameters

Name	Type	Valid range	Description
deviceType	uint8_t	gDeviceType_Max	The SynkroRF device type to search for during pairing.

Table 5-2. NLME_PairRequest Parameters

pPairingData	uint8_t*	-	Additional application defined data to be included in the pair request frame.
length	uint8_t	0 - 64	The length of the additional data to be included in the pair request frame.
timeOut	uint16_t	300 – 65536	The amount of time, in milliseconds, after which the pairing process is aborted in the absence of pair responses.

Return value

Possible return values and their significance:

`gNWSuccess_c` The request has been accepted for processing by the network layer

`gNWNodeNotStarted_c` The node has not yet been started

`gNWDenied_c` The network layer is already processing another non-interruptible request

`gNWNoMemory_c` The network could not allocate needed messages from the common message pool

`gNWNotPermitted` The function was called on a target node; the service is only available on controller nodes

`gNWNoOrigCapacity_c` There is no room in the pair table to add a new entry

`gNWInvalidParam_c`: TimeOut is smaller than 300 ms

Length is greater than 0 and pPairingData is NULL

Functional Description

SynkroRF_PairRequest is an asynchronous API function available on BeeStack Consumer/SynkroRF hybrid controller nodes. It makes a request for a hybrid controller node to establish a connection link with a SynkroRF controlled node which has the given device type. Included in the pair request frame is some application defined data, up to 64 bytes long.

This function call requests the starting of a SynkroRF controller pairing process. If the return value is `gNWSuccess_c`, the network layer has accepted the request for processing. When the pairing process is completed, the application layer will be noticed by a SynkroRF pair confirm message which will be sent by the network layer trough the NLME SAP.

SynkroRF pairing can proceed with either a legacy SynkroRF node or another BeeStack Consumer/SynkroRF hybrid node on the SynkroRF protocol.

Two hybrid nodes cannot be paired on both RF4CE and SynkroRF protocols in the same time. The second pairing will fail.

If the return value is not `gNWSuccess_c`, the SynkroRF controller pairing process has not been started. At this point, the request is considered to be complete, therefore the application should not expect a pair confirm message to arrive later.

5.2.2 SynkroRF_PairResponse

Prototype

```
uint8_t SynkroRF_PairResponse(
                              uint8_t          deviceId,
                              uint8_t          status,
                              uint8_t          length,
                              uint8_t*         pPairingData
                             );
```

Arguments

Table 5-3 specifies the parameters for the SynkroRF_PairResponse primitive.

Table 5-3. SynkroRF_PairResponse Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – 0xfe	The reference to the provisional pair table entry
status	uint8_t	gNWSuccess_c, gNWNotPermitted_c	Whether or not the application accepts pairing
length	uint8_t	0 - 64	The amount of data to be included in the pair response frame
pPairingData	uint8_t*	-	A pointer to the data to be included in the pair response frame

Return value

Possible return values and their significance:

`gNWSuccess_c` The request has been accepted for processing by the network layer

`gNWNodeNotStarted_c` The node has not yet been started

`gNWDenied_c` The network layer is already processing another non-interruptible request

`gNWNoMemory_c` The network could not allocate needed messages from the common message pool

`gNWInvalidParam_c` An incorrect deviceId was supplied

Length is greater than 0 and pPairingData is NULL

The supplied status is other than `gNWSuccess_c or gNWNotPermitted_c`

Functional Description

SynkroRF_PairResponse is an asynchronous API function available on BeeStack Consumer/SynkroRF hybrid target nodes. It makes a request for a hybrid node to exchange information with a SynkroRF controller node that has previously initiated pairing.

This function initiates the target side SynkroRF pairing process on a BeeStack Consumer/SynkroRF hybrid node.

If the return value is `gNWSuccess_c`, the network layer has accepted the request for processing. When the pairing process is complete, the application layer will be notified by a CommStatus indication message sent trough the NLME SAP.

If the return value is not `gNWSuccess_c`, the pairing process has not been started. At this point, the request is considered to be complete, therefore the application should not expect a CommStatus indication message to arrive later.

5.2.3 SynkroRF_ClearPairingInformation

Prototype

```
uint8_t Synkro_ClearPairingInformation (
                            uint8_t deviceId
                           );
```

Arguments

Table 5-4 specifies the parameters for the SynkroRF_ClearPairingInformation primitive.

Table 5-4. SynkroRF_ClearPairingInformation Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1)	The reference to the local pair table entry that is to be removed.

Return value

Possible return values and their significance:

`gNWSuccess_c` The relevant pair table entry has been deleted

`gNWNodeNotStarted_c` The node has not yet been started

`gNWDenied_c` The request is rejected as the network layer is already processing another non-interruptible request

`gNWDeviceIdNotPaired_c` The request is rejected as no pair information exists at the location specified by deviceId

`gNWInvalidParam_c` deviceId is outside the allowed range

The pair table entry pointed to by deviceId belongs to an RF4CE node

Functional Description

SynkroRF_ClearPairingInformation is a synchronous API function available both for hybrid controller and hybrid target nodes. It is used to request the deletion of the pair table entry of a paired SynkroRF node.

5.2.4 SynkroRF_SendCommand

Prototype

```
uint8_t SynkroRF_SendCommand(
                            uint8_t             deviceId,
                            uint16_t            cmdId,
                            uint8_t*            pData,
                            uint8_t             length,
                            bool_t              bAckRequired
                          );
```

Arguments

Table 5-5 specifies the parameters for the SynkroRF_SendCommand primitive.

Table 5-5. SynkroRF_SendCommand Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries - 1)	The pairing reference of the destination device.
cmdId	uint16_t	0x0000 – 0xFFFF	The ID of the SynkroRF command to be transmitted
pData	uint8_t*	-	A pointer to the command payload to be transmitted.
length	uint8_t	-	The length of the payload (in bytes) to be transmitted.
bAckRequired	bool_t	FALSE, TRUE	Whether to request a MAC layer acknowledgement to the frame

Return value

Possible return values and their significance:

`gNWSuccess_c` The request has been accepted for processing by the network layer

`gNWNodeNotStarted_c` The node has not yet been started

`gNWDenied_c` The network layer is already processing another non-interruptible request

`gNWNoMemory_c` The network could not allocate needed messages from the common message pool

`gNWDeviceIdNotPaired_c` The supplied deviceId points to an empty pair table entry

`gNWFrameCounterExpired_c` The frame counter has reached its maximum value

`gNWInvalidParam_c` The supplied deviceId is invalid (either out of range or belongs to an RF4CE node)
The payload is incorrect for the given command. See the *SynkroRF Network Reference Manual* for more details.
The requested command is not supported by the destination node as indicated by its node descriptor. See the *SynkroRF Network Reference Manual* for details.
Length is greater than 0 and pData is NULL.

Functional Description

SynkroRF_SendCommand is an asynchronous API function available both for controller and target nodes. It makes a request to the network layer to transmit a SynkroRF command to one of the nodes it is already paired with.

This function call requests the starting of a SynkroRF command transmission process.

If the return value is `gNWSuccess_c`, the network has accepted the request for processing. When the command transmission process is complete, the application layer will be noticed by a SynkroRF command confirm message sent through the NLDE SAP.

If the return value is not `gNWSuccess_c`, the SynkroRF command transmission process has not been started. At this point, the request is considered to be complete, therefore the application should not expect any SynkroRF command confirm message to arrive later.

5.2.5 SynkroRF_AddNewPairTableEntry

Prototype

```
uint8_t SynkroRF_AddNewPairTableEntry(
               uint8_t*                 localShortAddress,
               uint8_t                  recipChannel,
               uint8_t*                 recipMacAddress,
               uint8_t*                 recipPanId,
               uint8_t*                 recipShortAddress,
               SynkroRFNodeDescriptor_t* pRecipNodeDescriptor,
               uint8_t*                 recipUserString
               );
```

Arguments

Table 5-6 specifies the parameters for the SynkroRF_AddNewPairTableEntry primitive.

Table 5-6. SynkroRF_AddNewPairTableEntry Parameters

Name	Type	Valid range	Description
localShortAddress	uint8_t*	A valid short address (pointer different of NULL)	The network address to be assumed by the local device
recipChannel	uint8_t	15, 20, 25	The expected channel of the new paired device.
recipMacAddress	uint8_t*	A valid 802.15.4 IEEE address (non-NULL pointer)	The IEEE address of the new paired device.
recipPanId	uint8_t*	A valid PAN identifier (non-NULL pointer)	The PAN identifier of the new paired device.
recipShortAddress	uint8_t*	A valid short address (non-NULL pointer)	The network address of the new paired device.

Table 5-6. SynkroRF_AddNewPairTableEntry Parameters

pRecipNodeDescriptor	SynkroRFNodeDescriptor_t*	non-NULL pointer	The node capabilities of the new paired device.
recipUserString	uint8_t*	A valid user string (non-NULL pointer)	The user string of the new paired device.

SynkroRFNodeDescriptor_t is defined as:

```
typedef struct SynkroRFNodeDescriptor_tag
{
   uint8_t deviceType;                    /* device Type */
   uint8_t vendorId[2];                   /* vendor Id   */
   uint8_t productId[2];                  /* product Id  */
   uint8_t versionId;                     /* version Id  */
  uint8_t supportedConnections;          /* the number of connections supported on this node */
   uint8_t capabilities[gSynkroRFMaxDeviceCapabilities_c];
}SynkroRFNodeDescriptor_t;
```

Where: deviceType is the SynkroRF device type of the device, vendorId, productId and versionId contain the vendor ID, product ID and version ID of the device, supportedConnections is the size of the pair table and capabilities is a bitfield array (5 bytes long) indicating what SynkroRF command sets the device can handle.

Return value

The index in the pair table corresponding to the new added entry. Possible return values and their significance:

`gNWNoOrigCapacity_c` There is no room in the pair table to add a new entry

`gNWInvalidParam_c` The recipChannel parameter is invalid

Functional Description

SynkroRF_AddNewPairTableEntry is a synchronous API function available both on controller and target nodes.

It makes a request for a BeeStack Consumer/SynkroRF hybrid node to add a new SynkroRF entry in its pair table, without starting a pair process. The pairing information must be provided by the application.

This function call does not request the starting of any network layer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the SynkroRF_AddNewPairTableEntry request has been carried out. No confirm messages of any kind will arrive through the SAPs.

5.2.6 SynkroRF_GetPairedDeviceInfo

Prototype

```
uint8_t SynkroRF_GetPairedDeviceInfo(
               uint8_t                 deviceId,
               SynkroRFPairTableEntry_t* pSynkroRFPairTableEntry
               );
```

Arguments

Table 5-7 specifies the parameters for the SynkroRF_GetPairedDevice primitive.

Table 5-7. SynkroRF_GetPairedDeviceInfo Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries - 1)	The pair table index of the device about which information is requested
pSynkroRFPairTableEntry	SynkroRFPairTableEntry_t*	-	The memory area where the paired device info should be placed

SynkroRFPairTableEntry_t is defined as follows:

```
typedef struct SynkroRFPairTableEntry_tag
{
    uint8_t                 localShortAddress[2];
    uint8_t                 recipChannel;
    uint8_t                 recipMacAddress[8];
    uint8_t                 recipPanId[2];
    uint8_t                 recipShortAddress[2];
    uint32_t                recipFrameCounter;
    SynkroRFNodeDescriptor_t   recipSynkroRFNodeDescriptor;
    uint8_t                 recipUserString[gSizeOfUserString_c];
}SynkroRFPairTableEntry_t;
```

Return value

Possible return values and their significance:

`gNWSuccess_c` The paired device info has been copied to the specified address

`gNWDeviceIdNotPaired_c` The supplied deviceId points to an empty pair table entry

`gNWInvalidParam_c` The supplied deviceId is out of range or points to an RF4CE node

Functional Description

SynkroRF_GetPairedDeviceInfo is a synchronous API function available both on controller and target nodes.

It makes a request for the network layer to copy the pair table data of the indicated SynkroRF device to the given location.

This function call does not request the starting of any network layer process and for this reason its call is synchronous. When the application layer receives the return value of the API call, the SynkroRF_GetPairedDeviceInfo request has been carried out. No confirm messages of any kind will arrive through the SAPs.

5.2.7 Nwk_GetNodeType

Prototype

```
protocolType_t Nwk_GetNodeType(
                                  uint8_t deviceId
                               );
```

Arguments

Table 5-8 specifies the parameters for the Nwk_GetNodeType primitive.

Table 5-8. Nwk_GetNodeType Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries - 1)	The pair table index of the device about which information is requested

Return Value

The function returns the protocol type used by the paired node in the form of an enumeration which has the following possible self-explanatory values.

```
typedef enum protocolType_tag
{
  gProtocolTypeRF4CE_c,
  gProtocolTypeSynkroRF_c
}protocolType_t;
```

Functional Description

Nwk_GetNodeType is a synchronous API function available on both target and controller nodes. It informs the application of the protocol used for communication by the paired node.

The function does not perform any validation of the deviceId parameter, so care must be taken to have a valid deviceId before calling the function.

5.3 Message Data Types

The interface between the Network Layer Management Entity (NLME) as well as the Network Layer Data Entity (NLDE) and the Application Layer is based on service primitives passed from the network to the application through a Service Access Point (SAP). Two SAPs must be implemented as functions in the application:

- void NWK_NLME_SapHandler(nwkNlmeToAppMsg_t* nwkNlmeToAppMsg)
- void NWK_NLDE_SapHandler(nwkNldeToAppMsg_t* nwkNldeToAppMsg)

SynkroRF services also communicate with the application through these two SAPs.

The BeeStack Consumer/SynkroRF hybrid service primitives use the same type of messages as defined in the `NwkInterface.h` file. Because the hybrid interfaces are based on messages being passed to the SAPs, each message needs to have its own identifier. These identifiers are enumerated in the following tables.

Table 5-9. New Hybrid Primitives (NLME to Application Direction)

Message identifier	SynkroRF process	Available on Controller	Available on Target	Section
gNwkSynkroRFPairCnf_c	Controller pairing process	X		5.3.1
gNwkSynkroRFPairInd_c	Target pairing process		X	5.3.2

Table 5-10. New Hybrid Primitives in the NLDE to Application Direction

Message identifier	SynkroRF process	Available on Controller	Available on Target	Section
gNwkSynkroRFCommandCnf_c	Command transmission	X	X	5.3.3
gNwkSynkroRFCommandInd_c	Command reception	X	X	5.3.4

This section describes the main C-structures and data types used by the NLME/NLDE interface.

The structures used to describe the ZigBee RF4CE confirm and indication messages have been collected in a single message type as a union plus a message type that corresponds to the enumerations of the primitives. These are the structures which transport messages through the SAPs. For hybrid nodes, the structures have been enhanced with the hybrid functionality related messages.

For messages from NLME to the application, the following structure/union is used:

```
/* General structure of a message received by the application over NLME SAP */
typedef struct nwkNlmeToAppMsg_tag
{
  nwkNlmeToAppMsgType_t           msgType;
  union {
    nwkNlmeStartCnf_t             nwkNlmeStartCnf;
    nwkNlmeAutoDiscoveryCnf_t     nwkNlmeAutoDiscoveryCnf;
    nwkNlmeDiscoveryCnf_t         nwkNlmeDiscoveryCnf;
    nwkNlmeDiscoveryInd_t         nwkNlmeDiscoveryInd;
```

```
    nwkNlmePairCnf_t              nwkNlmePairCnf;
    nwkNlmePairInd_t              nwkNlmePairInd;
    nwkNlmeUnpairCnf_t            nwkNlmeUnpairCnf;
    nwkNlmeUnpairInd_t            nwkNlmeUnpairInd;
    nwkNlmeCommStatusInd_t        nwkNlmeCommStatusInd;
    nwkSynkroRFPairCnf_t            nwkSynkroRFPairCnf;
    nwkSynkroRFPairInd_t            nwkSynkroRFPairInd;
  } msgData;
}nwkNlmeToAppMsg_t;
```

Where: `nwkNlmeToAppMsgType_t` is as follows:

```
/* Messages used for informing the application about confirms or indications from RF4CE arrived
trough the NLME SAP */
typedef enum {
  gNwkNlmeStartCnf_c = 0,
  gNwkNlmeAutoDiscoveryCnf_c,
  gNwkNlmeDiscoveryCnf_c,
  gNwkNlmeDiscoveryInd_c,
  gNwkNlmePairCnf_c,
  gNwkNlmePairInd_c,
  gNwkNlmeUnpairCnf_c,
  gNwkNlmeUnpairInd_c,
  gNwkNlmeCommStatusInd_c,
  gNwkSynkroRFPairCnf_c = 0xF0,
  gNwkSynkroRFPairInd_c,
  gNwkNlmeMax_c
}nwkNlmeToAppMsgType_t
```

For messages from NLDE to application the following structure/union is used:

```
/* General structure of a message received by the application over NLDE SAP */
typedef struct nwkNldeToAppMsg_tag
{
  nwkNldeToAppMsgType_t    msgType;
  union
  {
    nwkNldeDataCnf_t            nwkNldeDataCnf;
    nwkNldeDataInd_t            nwkNldeDataInd;
    nwkSynkroRFCommandCnf_t       nwkSynkroRFCommandCnf;
    nwkSynkroRFCommandInd_t       nwkSynkroRFCommandInd;
  } msgData;
}nwkNldeToAppMsg_t;
```

Where: nwkNldeToAppMsgType_t is as follows:

```
/* Messages used for informing the application about confirms or indications from RF4CE arrived
trough the NLDE SAP */
typedef enum {
  gNwkNldeDataCnf_c       = 0,
  gNwkNldeDataInd_c,
  gNwkSynkroRFCommandCnf_c  = 0xF0,
  gNwkSynkroRFCommandInd_c,
  gNwkNldeMax_c
}nwkNldeToAppMsgType_t;
```

A detailed description of each hybrid message is presented in the following sections.

5.3.1 nwkSynkroRFPairCnf_t

Message structure

The nwkSynkroRFPairCnf_t message has the following structure:

```
typedef struct nwkSynkroRFPairCnf_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
  uint8_t                 length;
  uint8_t*                pPairingData;
}nwkSynkroRFPairCnf_t;
```

Structure members

Table 5-11 specifies the fields available in the nwkSynkroRFPairCnf_t message structure.

Table 5-11. nwkSynkroRFPairCnf_t message Structure

Name	Type	Possible values	Description
status	uint8_t	gNWSucces_c, gNWNoResponse_c, or a status value from the MCPS-DATA.confirm primitive	The status of the SynkroRF pairing process.
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1), 0xff	The pairing table reference for this pairing link (will be 0xff if status is not gNWSuccess_c)
length	uint8_t	-	The length of the additional pairing data present in the pair response frame
pData	uint8_t*	-	Additional pairing data present in the pair response frame

Possible values of the status:

`gNWSuccess_c` The SynkroRF pairing process has completed successfully

`gNWNoResponse_c` No pair response was received until the time-out if the MAC layer could not transmit the SynkroRF pair request frame over the air, the status field will contain the status returned by the MAC. A detailed description for the MAC status possible values can be found in R[2].

Functional Description

The SynkroRF Pair Confirm message can be received by the application layer on BeeStack Consumer/SynkroRF hybrid controller nodes.

This message notifies the application layer that a SynkroRF pairing process has completed and provides information about how the process has completed.

5.3.2 nwkSynkroRFPairInd_t

Message structure

The nwkSynkroRFPairInd_t message has the following structure:

```
typedef struct nwkSynkroRFPairInd_tag
{
  uint8_t                   status;
  uint8_t                   deviceId;
  uint8_t                   LQI;
  uint8_t                   length;
  uint8_t*                  pPairingData;
  SynkroRFNodeDescriptor_t*   pNodeDescriptor;
}nwkSynkroRFPairInd_t;
```

Structure members

Table 5-12 specifies the fields available in the nwkSynkroRFPairInd_t message structure.

Table 5-12. nwkSynkroRFPairInd_t message Structure

Name	Type	Possible values	Description
status	uint8_t	gNWSuccess_c, gNWNoRecipCapacity_c, gNWDuplicatePairing_c	The status of the provisional pairing
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1), 0xff	Next free pairing reference that will be used if this pairing request is successful. If this value is equal to 0xff, the NWK layer has no free pairing table entries.
length	uint8_t	0 - 64	The length of any additional pairing data present in the pair request frame
pPairingData	uint8_t*	-	A pointer to any additional data present in the pair request frame
pNodeDescriptor	SynkroRFNodeDescriptor_t*	-	The SynkroRF node descriptor of the device requesting pairing

SynkroRFNodeDescriptor_t is defined as:

```
typedef struct SynkroRFNodeDescriptor_tag
{
   uint8_t deviceType;                    /* device Type */
   uint8_t vendorId[2];                   /* vendor Id   */
   uint8_t productId[2];                  /* product Id  */
   uint8_t versionId;                     /* version Id  */
  uint8_t supportedConnections;         /* the number of connections supported on this node */
   uint8_t capabilities[gSynkroRFMaxDeviceCapabilities_c];
}SynkroRFNodeDescriptor_t;
```

Where: deviceType is the SynkroRF device type of the device, vendorId, productId and versionId contain the vendor ID, product ID and version ID of the device, supportedConnections is the size of the pair table and capabilities is a bitfield array (5 bytes long) indicating what SynkroRF commands the device can handle.

Possible values of the status are as follows:

`gNWSuccess_c` A new provisional entry containing the information about the pair requesting device was created

`gNWNoRecipCapacity_c` The pair table is full

`gNWDuplicatePairing_c` The device requesting pairing is already in the pair table

Functional Description

The SynkroRF pair indication message can be received by the application on hybrid target nodes.

This message notifies the application layer that a SynkroRF pair request command was received.

On receipt of the SynkroRF pair indication message, the application decides whether to accept pairing based on the information in the message. The application informs the network layer of the decision using the SynkroRF_PairResponse function call.

5.3.3 nwkSynkroRFCommandCnf_t

Message structure

The nwkSynkroRFCommandCnf_t message has the following structure:

```
typedef struct nwkSynkroRFCommandCnf_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
} nwkSynkroRFCommandCnf_t;
```

Structure members

Table 5-13 specifies the fields available in the nwkSynkroRFCommandCnf_t message structure.

Table 5-13. nwkSynkroRFCommandCnf_t message Structure

Name	Type	Valid range	Description
status	uint8_t	gNWSucces_c, gNWNoResponse_c, gNWNoMemory_c, gNWAborted_c or any other status value from the MCPS-DATA.confirm primitive	The status of the SynkroRF command transmission process.
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1)	The pairing table reference of the command destination.

Possible values of the status:

`gNWSuccess_c` The command has been successfully transmitted

`gNWNoResponse_c` A MAC layer acknowledgement has been requested and not received even after retransmission

`gNWNoMemory_c` No message buffers could be allocated to construct the frame

`gNWAborted_c` The transmission process has been aborted by another request if the MAC layer could not transmit the SynkroRF command frame over the air, the status field will contain the status returned by the MAC.

Functional Description

The SynkroRF command confirm message can be received by the application layer both on hybrid controller and hybrid target nodes.

This message notifies the application layer about the end result of a request to transmit a SynkroRF command.

5.3.4 nwkSynkroRFCommandInd_t

Message structure

The nwkSynkroRFCommandInd_t message has the following structure:

```
typedef struct nwkSynkroRFCommandInd_tag
{
  uint8_t                 deviceId;
  uint16_t                cmdId;
  uint8_t                 LQI;
  uint8_t                 dataLength;
  uint8_t*                pData;
}nwkSynkroRFCommandInd_t;
```

Structure members

Table 5-14 specifies the fields available in the nwkSynkroRFCommandInd_t message structure.

Table 5-14. nwkSynkroRFCommandInd_t message Structure

Name	Type	Possible values	Description
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries - 1), 0xff	Reference to the pair table entry of the command originator
cmdId	uint16_t	-	The ID of the received SynkroRF command.
LQI	uint8_t	0x00 – 0xff	LQI value measured during reception of the NSDU
dataLength	uint8_t	0 – 90	The length of the command payload
pData	uint8_t*	-	A pointer to the received payload data

Functional Description

The SynkroRF command indication message can be received by the application layer both on hybrid controller and hybrid target nodes.

It informs the application of the reception of a SynkroRF command from a paired device.

5.4 Hybrid Data Types

The following sections detail each to the hybrid data types.

5.4.1 NodeData Database

The overall structure of the NodeData Database of a hybrid node is unmodified from that of a pure ZigBee RF4CE node. Pair table entries belonging to SynkroRF nodes have bit 7 of the `recipCapabilities` field set to 1.

5.4.2 Hybrid NIBs

BeeStack Consumer/SynkroRF hybrid nodes do not have additional NIBs compared to pure ZigBee RF4CE nodes. The `discoveryLQIThreshold` NIB acts also as a SynkroRF pairing threshold.

5.4.3 Local SynkroRF Node Descriptors

Two SynkroRF devices exchange node descriptors during pairing. The node descriptor of a device is a constant structure that resides in flash and is completed at compile time.

The structure is of type SynkroRFNodeDescriptor_t, which is defined as:

```
typedef struct SynkroRFNodeDescriptor_tag
{
   uint8_t deviceType;                    /* device Type */
   uint8_t vendorId[2];                   /* vendor Id   */
   uint8_t productId[2];                  /* product Id  */
   uint8_t versionId;                     /* version Id  */
  uint8_t supportedConnections;          /* the number of connections supported on this node */
   uint8_t capabilities[gSynkroRFMaxDeviceCapabilities_c];
}SynkroRFNodeDescriptor_t;
```

Where: deviceType is the SynkroRF device type of the device, vendorId, productId and versionId contain the vendor ID, product ID and version ID of the device, supportedConnections is the size of the pair table and capabilities is a bitfield array (5 bytes long) indicating what SynkroRF command sets the device can handle.

5.4.4 Saving Sensitive Information in FLASH

The process of saving data to FLASH is the same on hybrid nodes as on pure ZigBee RF4CE nodes.

In addition to a pure node, the hybrid network also requests to save the information from the entire network data set in FLASH in the following situations:

- After a SynkroRF_PairResponse call, if the pair process was completed successfully, to save the updated pair table
- After the arrival of SynkroRF pair indication message, if the pair process was completed successfully, to save the updated pair table
- After a SynkroRF_ClearPairingInformation call, to save the updated pair table
- After a SynkroRF_AddNewPairTableEntry call, to save the updated pair table