ZigBee RF4CE Stack User Guide

JN-UG-3074
Revision 1.1
6 December 2012

Contents

About this Manual

This manual provides a reference point for information relating to the ZigBee RF4CE network stack which can be implemented on the NXP JN516x device. The manual provides both conceptual and practical information concerning the NXP ZigBee RF4CE stack software. Guidance is provided on use of the Application Programming Interface (API) for ZigBee RF4CE, and the API functions and associated resources (enumerations, structures and constants) are described. The manual should be used as a reference resource throughout ZigBee RF4CE application development.

Pre-requisites

The reader is expected to be familiar with:

- C programming
- JN516x Software Developer's Kit (SDK)

Note: ZigBee RF4CE is built on the IEEE 802.15.4 wireless network standard. Knowledge of IEEE 802.15.4 may therefore be beneficial when developing ZigBee RF4CE applications using the supplied API. The essential background concepts are covered in the *IEEE 802.15.4 Wireless Networks User Guide (JN-UG-3024)*, available from the Support section of the NXP web site (www.nxp.com/jennic/support).

Organisation

This manual is divided into two parts:

- Part I: Concept and Operational Information comprises two chapters:
  - Chapter 1 introduces ZigBee RF4CE networks.
  - Chapter 2 describes the essential principles of the ZigBee RF4CE stack implementation and use of the ZigBee RF4CE API functions.
- Part II: Reference Information comprises two chapters:
  - Chapter 3 provides detailed descriptions of the ZigBee RF4CE API functions.
  - Chapter 4 lists the enumerations, structures and constants used in the ZigBee RF4CE API.
- Part III: Appendices describes the enumerations, structures and constants used in the ZigBee RF4CE API. These resources are defined in the header file **RF4CE_API.h**.

Conventions

Files, folders, functions and parameter types are represented in **bold** type.

Function parameters are represented in *italics* type.

Code fragments are represented in the `Courier New` typeface.

This is a **Tip**. It indicates useful or practical information.

This is a **Note**. It highlights important additional information.

*This is a **Caution**. It warns of situations that may result in equipment malfunction or damage.*

Acronyms and Abbreviations

API	Application Programming Interface
CE	Consumer Electronics
CERC	Consumer Electronics Remote Control
MAC	Medium Access Controller
NIB	Network Information Base
NLDE	NWK Layer Data Entity
NLME	NWK Layer Management Entity
NWK	Network (layer)
PAN	Personal Area Network
PHY	Physical (layer)
RC	Remote Control
RF4CE	Radio Frequency for Consumer Electronics
ZRC	ZigBee Remote Control
ZID	ZigBee Input Device

Related Documents

JN-UG-3024 IEEE 802.15.4 Wireless Networks User Guide

JN-UG-3064 SDK Installation and User Guide

JN-UG-3087 JN516x Integrated Peripherals API User Guide

094945r00ZB ZigBee RF4CE Specification [ZigBee Alliance document]

094950r00ZB ZigBee RF4CE Device Type List [ZigBee Alliance document]

094951r00ZB ZigBee RF4CE Profile ID List [ZigBee Alliance document]

Trademarks

Part I:
Concept and Operational Information

1. Introduction to ZigBee RF4CE

ZigBee RF4CE is a wireless network standard designed specifically for Remote Control (RC) products in the Consumer Electronics (CE) domain. The standard was jointly devised by the ZigBee Alliance and the "Radio Frequency for Consumer Electronics" (RF4CE) consortium. The aim was to establish a simple, robust and low-cost radio communication standard for remote control in consumer products. The standard is built on the well-established IEEE 802.15.4 wireless network protocol.

In a ZigBee RF4CE network, one or more remote control units may be wirelessly networked to control one or more devices. For example, the standard can be used to achieve a comprehensive and flexible remote control solution for an audio-visual system that may include one or more of the following: TV, HDD recorder, Blu-ray player, DVD player, CD player, amplifier.

Note: A more detailed account of ZigBee RF4CE can be found in the *ZigBee RF4CE Specification (094945r00ZB)*, available from the ZigBee Alliance web site. An introduction to IEEE 802.15.4 can be found in the *IEEE 802.15.4 Wireless Networks User Guide (JN-UG-3024)*, available from the Support section of the Jennic web site.

1.1 Features

The main features of the ZigBee RF4CE standard are:

- Operates in one of 3 channels of the 2.4-GHz radio band
- Frequency agility over the 3 channels
- Multiple Star topology with inter-PAN communication
- Flexible transmission options
- Service discovery mechanism
- Device pairing mechanism
- Power saving mechanism
- Key-based security mechanism utilising industry-standard AES-128 scheme
- Simple RC profile for CE products, with option to add further standard or vendor-specific profiles

1.2 Node Types and Network Topologies

In the ZigBee RF4CE standard, two or more devices are organised into an RC Personal Area Network (PAN) with a Star topology. Multiple RC PANs can then form an RC network, allowing communication between PANs (as well as inside PANs).

An RC PAN consists of two node types:

- **Target node:** This node type is incorporated in a device to be controlled, e.g. a TV. The node acts as a PAN Co-ordinator and therefore creates a PAN. There must be only one Target node per RC PAN.
- **Controller node:** This node type sends or passes on control messages. It is incorporated in remote control units and in devices that relay control messages (e.g. a TV that passes control messages to a DVD player). An RC PAN can have multiple Controller nodes.

A simple RC PAN is illustrated in the figure below, consisting of a TV (Target node and PAN Co-ordinator) and a TV RC (Controller node).

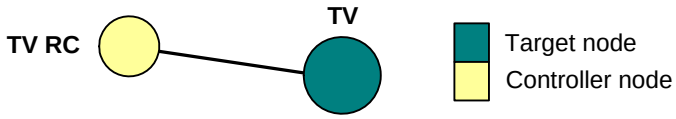

Figure 1: Example RC PAN

Extending the example, this RC PAN (PAN 1) is combined with another RC PAN (PAN 2) consisting of a DVD player (Target Node and PAN Co-ordinator) and a DVD RC (Controller node), as illustrated in the figure below.

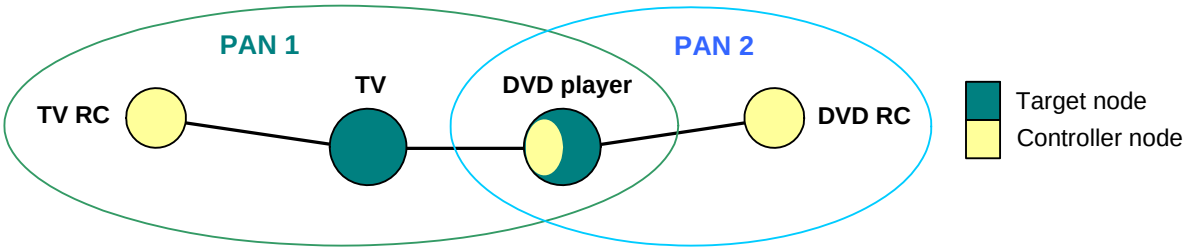

Figure 2: Example RC Network Formed from PAN 1 and PAN 2

In this RC network, the DVD player from PAN 2 also joins PAN 1 as a Controller node - this allows the DVD player to relay control messages from the DVD RC to the TV (for example, to use the DVD RC to adjust the volume level of the TV). Thus, the DVD player acts as a Target node (and PAN Co-ordinator) in PAN2 and as a Controller node in PAN 1.

Extending the example further, a third RC PAN (PAN 3) is added to the RC network, where PAN 3 consists of a Hi-Fi system (Target node and PAN Co-ordinator) and an Hi-Fi RC (Controller node), as well as a multi-function RC (Controller node). The multi-function RC also joins PAN 1 and PAN 2, allowing it to control all three Target nodes. The resulting RC network is illustrated in the figure below.

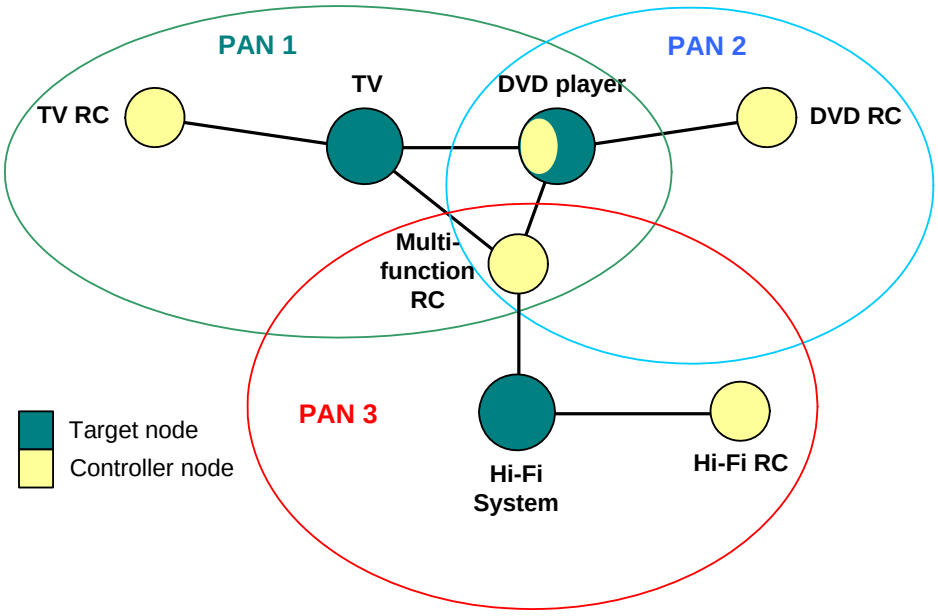

Figure 3: Example RC Network Formed from PAN 1, PAN 2 and PAN 3

1.3 Radio Channels and Frequency Agility

The ZigBee RF4CE standard employs the 2.4-GHz radio frequency band which is available in the IEEE 802.15.4 standard. However, only three of the sixteen channels in this band (numbered 11-26) are available in ZigBee RF4CE. The available channels are numbers 15, 20 and 25, which are centred on the frequencies 2425 MHz, 2450 MHz and 2475 MHz, respectively.

When a PAN Co-ordinator (Target node) forms a PAN, it will scan the three channels for activity and select the quietest channel for the PAN. The RC PANs within an RC network can operate in different channels but it is possible for two or more RC PANs to operate in the same channel. In the latter case, the PANs are distinguished by their PAN IDs, which is a unique 16-bit value for each PAN in the operating neighbourhood (see Section 1.4.1).

If a node of an RC network is a member of two (or more) PANs that operate in different channels, it will be necessary for the common node to change channels when relaying control messages from one PAN to another.

ZigBee RF4CE provides a frequency agility option which, when enabled, allows the Target node of an RC PAN to automatically change channel if the initial channel chosen becomes increasingly busy (with traffic from other PANs or radio sources) and therefore problematic. A Controller node of the RC PAN will then detect the channel change when its transmissions to the Target node are unacknowledged (frequency agility requires acknowledgements to be enabled). In this case, the Controller node will transmit in all three channels until it finds the Target node again.

1.4 RC PAN Formation

An RC PAN is created by a Target node, which acts as the PAN Co-ordinator. The created PAN goes through the following formation process:

1. Target node initialises itself as a Target node and PAN Co-ordinator, and then selects a radio channel and PAN ID (see Section 1.4.1).
2. Each Controller node initialises itself as a Controller node.
3. Target node or each Controller node performs a 'service discovery' to find nodes with which it can be paired (see Section 1.4.2).
4. Target node or each Controller node pairs itself with suitable node(s) which it has discovered (see Section 1.4.3).

1.4.1 Initialisation

When the Target node is started, it performs the following initialisation tasks:

1. The node initialises itself as a Target node and PAN Co-ordinator.
2. The node searches for a suitable radio channel for its PAN by performing an 'energy detection scan' in the three available radio channels (see Section 1.3). It selects the channel with the least detected activity.
3. The node then performs an 'active scan' in all three radio channels to detect any other IEEE 802.15.4-based PANs that are operating in these channels. Based on the scan results, the node generates a random 16-bit PAN ID that does not clash with the PAN IDs of any detected PANs.

When started, each Controller node simply initialises itself as a Controller node.

A 'service discovery' is then performed, as described in Section 1.4.2.

1.4.2 Discovery

A 'service discovery' can be performed by a node to find other suitable nodes with which it can be paired and therefore communicate (see Section 1.4.3). Typically, the Controller nodes perform this discovery.

A 'service discovery' is implemented by broadcasting discovery requests in all three ZigBee RF4CE radio channels. This broadcast can be performed repeatedly for a fixed duration or until a certain number of discovery responses have been received from suitable nodes (these options are selected through the NIB - see Section 2.2.3).

A node will respond (under application control - see Section 2.3.2) depending on the information sent out by the requesting node, which is as follows:

- **Node capabilities:** Node type (Target or Controller), power source (mains or battery) and security supported (yes or no)
- **Vendor information:** Vendor ID assigned by ZigBee RF4CE and a vendor-specific identifier string (e.g. serial number)
- **Application information:** User-defined descriptor for node (e.g. "Lounge TV RC"), list of supported device types (e.g. "TV", "DVD") and a list of profile identifiers indicating supported application profiles (e.g. "CERC")
- **Requested device:** Device type requested through the discovery (e.g. TV)

The responding node will also return the above information (about itself), except the requested device.

1.4.3 Pairing

Before two nodes of an RC network can communicate during normal network operation (i.e. Controller node can send control messages to the Target node), they must be 'paired'. The pairing mechanism is described below:

1. Once a node has received responses to its 'service discovery' (see Section 1.4.2), it uses the returned information to decide which responding node(s) to pair with - for example, a DVD RC may be looking for a DVD player from a certain vendor.
2. The node sends a 'pairing request' to the node(s) that it wishes to pair with.
3. If the receiving node accepts the pairing request, it confirms the pairing by sending a pairing response back to the requesting node.

Both nodes also add the 'pairing link' to their respective pairing tables.

A pairing table entry contains the following information: pairing reference (index in table), source network (short) address, destination logical channel, destination IEEE (MAC) address, destination PAN ID, recipient node capabilities, recipient frame counter, security link key.

Thus, the use of pairing avoids the need for the application on a node to be concerned with node addressing. The application can indicate a destination node for a control message simply by specifying the index of the relevant entry in the local pairing table.

1.5 Communications

Communications in an established RC network generally consist of control messages sent from Controller nodes to Target nodes, and optional acknowledgements from Target nodes to Controller nodes.

A number of transmission options are available for the control messages:

- Acknowledged: Receipt of message is confirmed by destination node
- Unacknowledged: Receipt of message is not confirmed by destination node
- Unicast: Message sent to specific destination node
- Broadcast: Message sent to all possible destination nodes
- Multiple channel: Message sent on all three radio channels in turn
- Single channel: Message sent on expected radio channel

Some of the above options can be combined (e.g. acknowledged unicast in single channel).

A message can be routed between two RC PANs by a node that is a member of both PANs. This node will be a Target node in the source PAN but a Controller node in the destination PAN. If the two PANs operate in different radio channels, it will be necessary for the node to change channels between receiving and re-transmitting the message.

1.6 Application Profiles

The ZigBee RF4CE standard uses application profiles in issuing control messages. An application profile consists of a command set, comprising commands that can be incorporated in the control messages (e.g. increment TV channel). Profiles can be standard or vendor-specific:

- A standard profile contains a standard command set but can also incorporate a vendor-specific command set.
- A vendor-specific profile contains a vendor-specific command set only.

The Generic Device Profile (GDP) is a foundation for the RF4CE profiles defining mandatory policies which all profiles must adopt and a toolbox of commands and procedures to enable generic functionality within a profile.

The ZigBee Remote Control (ZRC) profile defines commands and procedures enabling the control of consumer electronic products such as TVs, DVD players as well as CD players.

The ZigBee Input Device (ZID) interfaces to the ZigBee RF4CE network layer. The ZID profile defines commands and procedures to enable devices such as mice, touchpads, keyboards, wands, Riva wheels and RC pointers.

1.7 Power Saving

The nodes of an RC network need not be fully powered all of the time. Principally, a battery-powered Controller node, such as a remote control unit, should not be permanently active as it may be required only occasionally and it is important to maximise battery life. Additionally, it is important to minimise the power consumption of a Target node, such as a TV, while in the stand-by state.

Power savings in an RC network can be achieved by carefully controlling the time for which the radio receivers in the nodes are active. The following power options may be implemented by the application:

- Enable receiver until further notice - for example, for a fully operating TV (receiver can be enabled in this mode on power-up or coming out of stand-by)
- Enable receiver for fixed period - for example, before engaging full 'power-saving mode' (see below) when a TV enters the stand-by state
- Disable receiver until further notice - for example, to put a remote control unit into a dormant state until it is woken by a button-press

ZigBee RF4CE also provides a power-saving mode in which the receiver on a node is disabled for most of the time but is periodically enabled for a short time. Thus, to communicate with a Target node in power-saving mode, the Controller node must transmit during the Target node's active periods.

The JN516x device can also be put into sleep mode, which provides the most significant power saving. A timeout can be applied to the sleep duration or the device can be woken as the result of a user action, such as pressing a button.

1.8 Stack Architecture

The software that runs on each node of a ZigBee RF4CE network is organised in the stack structure shown in Figure 4.

ZigBee RF4CE is built on the IEEE 802.15.4 wireless network standard, which sits at the bottom of the stack. The vendor's application sits at the top of the stack with the ZigBee RF4CE networking software sitting in the intermediate stack layers.

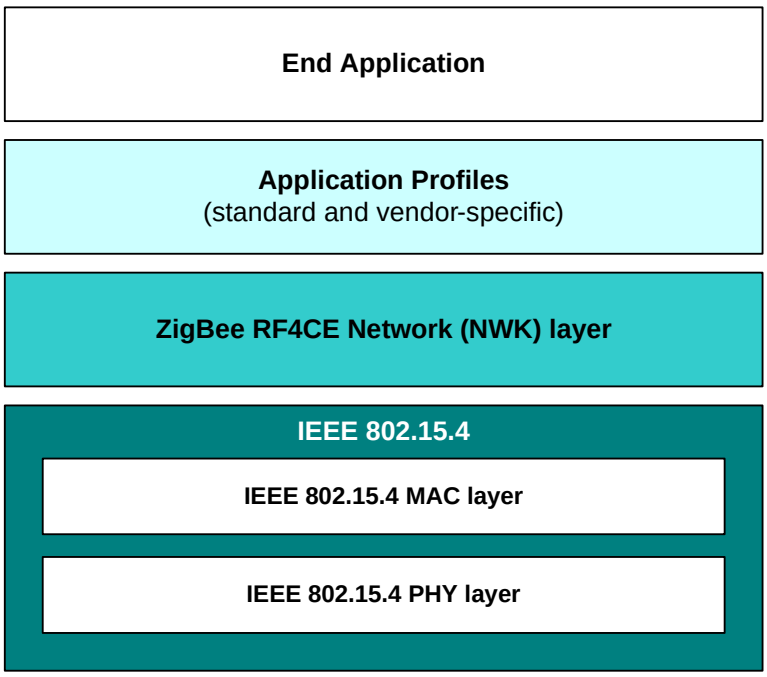

Figure 4: ZigBee RF4CE Stack Architecture

The layers of the ZigBee RF4CE stack are described below (from top to bottom):

- **End Application:** This layer contains the vendor-designed applications that run on the node. An application gives the device its functionality. A single node may run several applications.
- **Application Profiles:** This layer contains the standard and vendor-specific profiles used by the application(s) - see Section 1.6.
- **ZigBee RF4CE Network (NWK) layer:** This layer provides the ZigBee networking functionality and provides the application's interface to the IEEE 802.15.4 layers (see below). The NWK layer contains two services:
  - **NWK data service:** Provides interface to the NWK Layer Data Entity (NLDE) concerned with the packing and unpacking of control messages in NWK frames (which are encapsulated in IEEE 802.15.4 MAC frames for transport across the network - see below).
  - **NWK management service:** Provides interface to the NWK Layer Management Entity (NLME) concerned with network issues such as

initialisation, discovery and pairing, as well as maintenance of the Network Information Base (NIB).

- **IEEE 802.15.4 layer:** This layer is sub-divided into two further layers:
  - **MAC layer:** This layer (also known as the Data Link layer) is responsible for addressing, and is responsible for assembling the IEEE 802.15.4 MAC frames to be transmitted and disassembling the received frames.
  - **PHY layer:** This layer (also known as the Physical layer) is concerned with the interface to the physical transmission medium (radio, in this case), exchanging data bits with this medium as well as exchanging data bits with the MAC layer above.

The NIB, which is managed by the NWK layer, contains a set of attributes detailing certain network properties (see Section 2.2.3).

Note: The stack is not responsible for relaying control messages between PANs (e.g. a TV passing control messages to a DVD player). The relaying functionality is the responsibility of the application.

2. Using the ZigBee RF4CE API

A ZigBee RF4CE Application Programming Interface (API) is available for developing application code for the JN516x device. This chapter describes key operational aspects of this API before the functions of the API are detailed in Chapter 3.

2.1 RF4CE API Installation and Contents

The ZigBee RF4CE API is supplied in ZigBee RF4CE SDK (JN-SW-4060). The SDK must be installed on the development machine.

The ZigBee RF4CE SDK is installed simply by extracting the contents of the file **JN-SW-4060-ZigBee-RF4CE-SDK.zip** into the Jennic directory (**C:\Jennic**, by default) and put the required components to build RF4CE-based application.

- The ZigBee RF4CE library **libRF4CE_JN516x.a** will be installed into the **Components\Library** folder
- The header files **RF4CE_API.h** and **NIB.h** will be installed into the **Components\RF4CE\Include** folder

The ZigBee RF4CE API contains C functions which allow the application to interact with the ZigBee RF4CE Network (NWK) layer of the software stack - for stack details, refer to Section 1.8. The API functions are divided into four sets, as follows:

- **Implementation-specific functions:** Used to configure and save stack settings that depend on the individual application (detailed in Section 3.1)
- **NLDE function:** Used to interact with the NWK Layer Data Entity (NLDE) services (detailed in Section 3.2)
- **NLME functions:** Used to interact with the NWK Layer Management Entity (NLME) services (detailed in Section 3.3)
- **Callback function:** Deals with stack events (detailed in Section 3.4)

JN516x Integrated Peripherals API

The JN516x Integrated Peripherals API can be used at the same time as the ZigBee RF4CE stack. This API is provided in SDK and is described in the *JN516x Integrated Peripherals API User Guide (JN-UG-3087)*. In addition, the Board API (provided for use with evaluation kits) can be used.

It should be noted that the ZigBee RF4CE stack uses Flash memory and the JN516x Wake Timer 1 to store information.

If the JN516x peripherals are configured to generate interrupts, the application must register callback functions as described in the Integrated Peripherals API documentation.

ZRC and ZID Application Profile Functions

ZRC and ZID application profile functions are provided for application use. Please refer to *ZigBee RF4CE Demonstration Application note* (*JN-AN-1158*) for the source code and corresponding header files **ZRC.h** and **ZID.h**. Refer Section 3.5 and Section 3.6 for more details.

2.2 Application Overview

An RF4CE application must first perform the necessary initialisation and network formation activities that are required following a cold start or reset:

- The Target node acts as a PAN Co-ordinator and must be started first. The application which runs on this node must initialise itself as a Co-ordinator/ Target node. During this initialisation, the ZigBee RF4CE stack selects a radio channel and a PAN ID for the PAN (see Section 1.4.1).
- A Controller node application must first initialise itself as a Controller node. The application must then perform a 'service discovery' to find a Target node with which to communicate (see Section 1.4.2) and initiate a 'pairing' with the selected node (see Section 1.4.3).

Once a PAN has been set up, the applications are mainly concerned with user-initiated activities - for example, a button press on a TV remote control unit (Controller node) to change the channel on a TV (Target node).

An RF4CE application is largely event-driven and therefore reacts to events generated either internally or by user actions. The operational aspects of an RF4CE application are described in the sub-sections below before use of the RF4CE API functions is described in Section 2.3.

2.2.1 Tasks and Contexts

This implementation of the ZigBee RF4CE stack does not require a Real-time Operating System (RTOS) or task scheduler, which keeps the implementation small and efficient. All processing is performed in one of two CPU operating contexts: application and interrupt.

Normally, the application runs in the application context and makes calls into the ZigBee RF4CE stack, which therefore also runs in application context to perform the requested operation or to configure a hardware activity. This is shown in Figure 1.

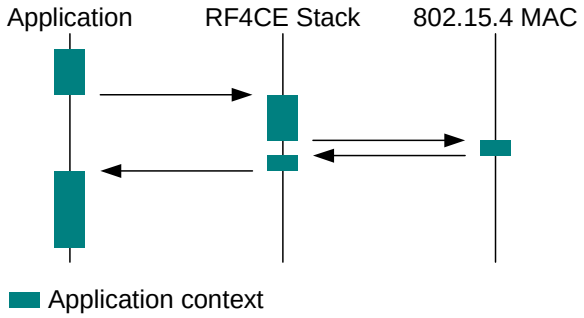

Figure 1: Application Context

On completion of a hardware activity (such as a frame transmission), the hardware generates an interrupt and the processor suspends whatever it is doing in application context in order to service the interrupt. This will result in the ZigBee RF4CE stack running in interrupt context, perhaps calling back to the application with the result of any activity. This is shown in Figure 2.

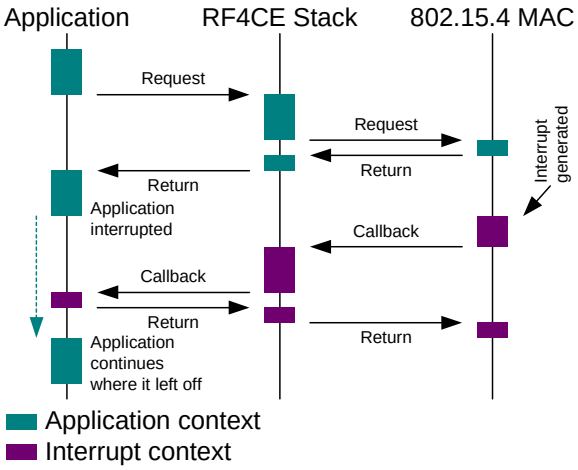

Figure 2: Application and Interrupt Context

It can be seen that the call back to the application is also performed in the interrupt context and therefore the application-supplied callback function should be kept as small as possible. Once the interrupt processing has completed, the application continues from where it left off.

Note: If the application requests the ZigBee RF4CE stack to perform more than one activity at the same time, such as another data request while one is still in progress or a pairing request during a discovery operation, the later requests will be rejected with error code E_RF4CE_STATUS_NOT_PERMITTED. This is an implementation-specific feature.

2.2.2 Calling Protocol

Calls from the Application to the Stack

All function calls into the stack from the application are non-blocking (that is, they will never wait in a busy loop for an external action to complete).

Some calls may cause actions in the hardware that could take a relatively long time, such as transmitting a frame. In these cases, the function will return before the action has completed and the stack will subsequently call an application-supplied function (a callback function - see below) when the action does complete.

The rules for return values from calls into the stack are as follows:

- Calls to functions that will always produce a result quickly will return the result as a return value from the call.
- Calls to functions that may take a while will always return void and the stack will call the application callback function (see below) when the result occurs. Note that if the function completes quickly (e.g. when the input parameters are invalid), the callback function may be called before the original call returns.

Calls from the Stack to the Application

The application must provide a callback function, which is called by the stack on completion of various actions (see Section 2.2.1). This function is **vRF4CE_StackEvent()** and is detailed in Section 3.4.

2.2.3 Network Information Base (NIB)

A Network Information Base (NIB) is maintained which contains a set of attributes relating to various network properties/operations, including:

- Radio channel
- Scan duration for network search
- Frame counter
- Pairing table
- Timeout for responses

The full list of NIB attributes is provided in Appendix A.2 and the attributes are described in the *ZigBee RF4CE Specification*.

Two functions are provided which allow the application to access the NIB:

- **eRF4CE_NlmeSetReq()** can be used to set the value of a NIB attribute
- **eRF4CE_NlmeGetReq()** can be used to obtain the value of a NIB attribute

In addition, the function **vRF4CE_ImpSaveSettings()** can be used to save the NIB settings to non-volatile memory so that they can later be retrieved following a device reset.

2.2.4 Event Handling

The stack forces an event-driven approach to application development. For example, a typical Target node such as a television might go through the following set of states from first being started to being fully operational:

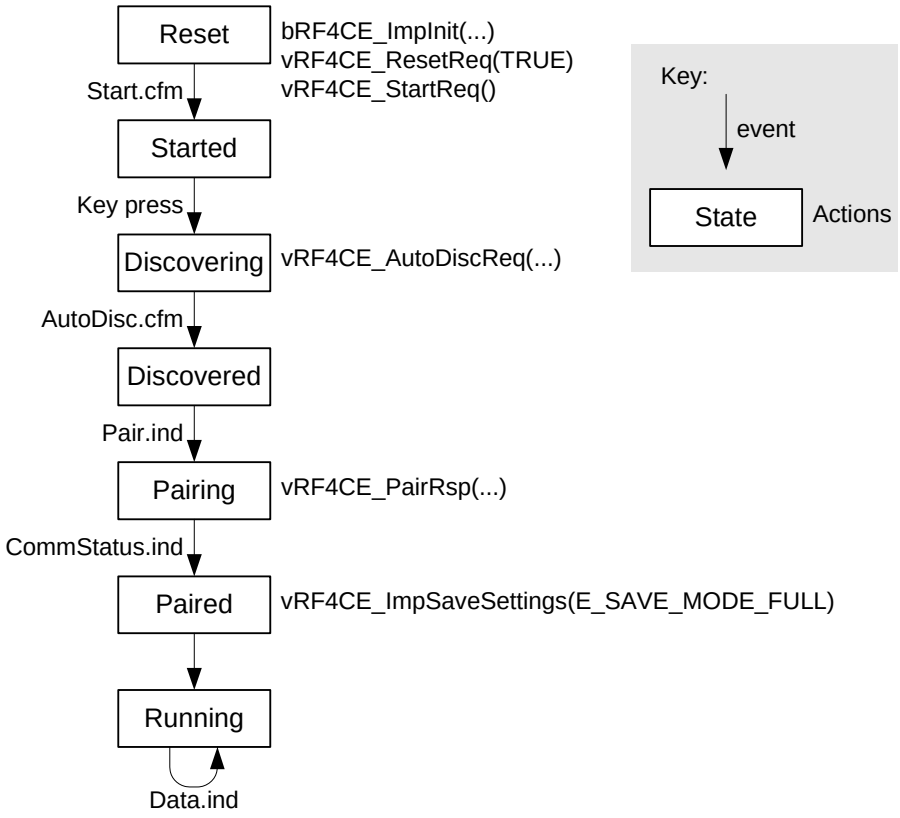

Figure 3: Typical Sequence of Start-up States for Target Node

An event-driven system implies an idle loop for periods when there are no events to process.

The callback function **vRF4CE_StackEvent()** receives all stack events. Peripheral events are received through callback functions that the application registers with the JN516x Integrated Peripherals API. The most efficient way of running the application is often for these callback events to be queued or flagged within the callback function, and for the idle loop to process them after the callback function has returned. In this way, the time spent in interrupt context is minimised, plus the idle loop can be set to doze the processor when processing has finished, reducing power consumption.

2.3 PAN Formation

This section describes use of the ZigBee RF4CE API functions to code the initialisation and PAN formation parts of an RF4CE application:

- Stack initialisation is described in Section 2.3.1
- Service discovery is described in Section 2.3.2
- Pairing is described in Section 2.3.3

The ZigBee RF4CE API functions are individually detailed in Chapter 3.

2.3.1 Stack Initialisation

This section describes how to initialise the ZigBee RF4CE stack on a Target node or Controller node. The **AppColdStart()** function, in which the application is defined, must allow for the possibilities of a first-time cold start and a device reset.

During normal operation, as defined in the *ZigBee RF4CE Specification*, a node stores its configuration so that the settings can be recovered after a reset. This means that nodes will continue to operate following a power cycle without any further set-up being required. The implementation-specific initialisation functions (see Section 3.1) are designed to operate in a specific way in order to achieve this.

The application code should follow the stack initialisation process outlined below:

1. Upon a cold start, the JN516x Integrated Peripherals API and any peripherals, if required, should be initialised first.
2. An optional feature (for development, at least) is to have a way to clear any previous stack configuration. To do this, **vRF4CE_ImpDestroySettings()** should be called.
3. **bRF4CE_ImpInit()** should then be called to configure the stack. The return value from this function indicates whether the stack has previously been configured.
4. If the stack has previously been configured, **vRF4CE_NlmeResetReq()** should be called with the parameter FALSE. This will reset the stack into the most recently saved configuration, including activation of the receiver if appropriate.

   If the stack has not been previously configured, **vRF4CE_NlmeResetReq()** should be called with the parameter TRUE and then **vRF4CE_StartReq()** should be called to start the network. At this point, the application may initialise the NIB settings using **eRF4CE_NlmeSetReq()** - see Section 2.2.3.
5. The program flow should then go into an idle loop awaiting events, as described in Section 2.2.4.

When the stack is started on the Co-ordinator/Target node, it will perform the necessary scans to select a radio channel and PAN ID (see Section 1.4.1).

If the stack was started with an unknown configuration then once the configuration is complete (network started and devices paired), the configuration should be saved by calling **vRF4CE_ImpSaveSettings()** with the parameter E_SAVE_MODE_FULL.

An example start-up function, illustrating Steps 1 to 5, is shown below.

```
#define FLASH_SECTOR    (3)
#define FLASH_START     (FLASH_SECTOR * 0x08000)
#define NODE_CAPABILITY (  RF4CE_NODECAP_TYPE_CONTROLLER  \
                         | RF4CE_NODECAP_PWRSRC_MAINS     \
                         | RF4CE_NODECAP_CHANNORM_CAPABLE)
#define VENDOR_ID       (0xfff1)

PUBLIC void AppColdStart(void)
{
    /* Initialise Peripheral API and any peripherals */
    (void)u32AHI_Init();
    vButtonInitFfd();

    /* Check if flash should be erased */
    if (u8ButtonReadFfd() == (BUTTON_0_MASK | BUTTON_3_MASK))
    {
        vRF4CE_ImpDestroySettings(FLASH_START);
    }

    /* Initialise stack (implementation-specific command) */
    if (bRF4CE_ImpInit(NODE_CAPABILITY, VENDOR_ID,
                       (uint8 *)"Jennic ", FLASH_START,
                       FLASH_SECTOR))
    {
        /* Cold start: reset and clear NIB, then start stack */
        vRF4CE_NlmeResetReq(TRUE);
        vRF4CE_StartReq();

        /* The start request will generate an event when done */
    }
    else
    {
        /* Warm start: reset without clearing NIB */
        vRF4CE_NlmeResetReq(FALSE);

        /* The stack is now running */
    }

    /* Go to idle loop to await events */
    vIdleLoop();
}
```


2.3.2 Service Discovery

Service discovery is the mechanism by which a Controller node finds a Target node with which it can communicate (see Section 1.4.2). Once a Controller node has been initialised (see Section 2.3.1), its application must send out a service discovery request using the function **vRF4CE_NlmeDiscoveryReq()**, which may be called as the result of a user action such as a button-press. This function allows the search to be restricted by specifying:

- Particular PAN ID, or no preference
- Particular Target node (through address), or no preference
- Particular device type (to find), or no preference
- Profile IDs of interest
- Timeout period for response to request

The arrival of a service discovery request can be handled manually or automatically by the Target node:

- **Manual Service Discovery:** The request results in a discovery indication event (E_RF4CE_EV_DISC_IND) which is handled by the callback function **vRF4CE_StackEvent()**. This callback function calls the function **vRF4CE_NlmeDiscoveryResp()** to respond to an individual service discovery request.
- **Automatic Service Discovery:** The application calls the function **vRF4CE_NlmeAutoDiscoveryReq()** once to automatically respond to all subsequent service discovery requests received within a specified time period. This function is called in the main application.

When the response is received by the Controller node, a discovery confirmation event (E_RF4CE_EV_DISC_CFM) or auto-discovery confirmation (E_RF4CE_EV_AUTODISC_CFM) is generated, as appropriate, which is handled by the callback function **vRF4CE_StackEvent()**. In addition, a confirmation of receipt is sent back to the Target node, where it triggers a comm-status indication event (E_RF4CE_EV_COMMSTATUS_IND).

The response includes information on the capabilities, device types and profile IDs of the responding Target node. If more than one response is received, the Controller node will use this information to select the Target node(s) to pair with (see Section 2.3.3).

2.3.3 Pairing (and Unpairing)

In order to send a control message to a node, a Controller node must first be paired with a Target node (see Section 1.4.3). This pairing is initiated by the application on the Controller node. The Target node is selected on the basis of the results of a service discovery (see Section 2.3.2).

1. The application on the Controller node must request the required pairing using the function **vRF4CE_NlmePairReq()**.
2. The arrival of the pairing request at the Target node results in a pairing indication event (E_RF4CE_EV_PAIR_IND), which is handled by the callback function **vRF4CE_StackEvent()**.
3. This callback function must call the function **vRF4CE_NlmePairResp()** to respond to the pairing request.
4. When the response is received by the Controller node, a pairing confirmation event (E_RF4CE_EV_PAIR_CFM) is generated, which is handled by the callback function **vRF4CE_StackEvent()**.
5. In addition, a confirmation of receipt is sent back to the Target node, where it yields a comm-status indication event (E_RF4CE_EV_COMMSTATUS_IND), which is handled by the callback function **vRF4CE_StackEvent()**.

The response will indicate whether the pairing has been accepted or rejected by the Target node. If successful, the response will also include a pairing reference number, and the pairing will be added to the pairing tables on both the Controller node and Target node - the pairing reference number serves as the index of the entry in the pairing table.

Functions also exist to unpair two nodes. The function **vRF4CE_NlmeUnpairReq()** must be called on one node to request that the relevant entry is removed from the local pairing table - the index of the entry in the pairing table must be provided. The function also notifies the paired node that it is being unpaired - the successful transmission of this notification results in an unpairing confirmation event (E_RF4CE_EV_UNPAIR_CFM) on the sending node. On the receiving node, an unpairing indication event (E_RF4CE_EV_UNPAIR_IND) is generated, which is handled by the callback function **vRF4CE_StackEvent()**. This callback function must call the function **vRF4CE_NlmeUnpairResp()** to instruct the stack to remove the relevant entry from the local pairing table.

2.4 Low-power Modes

A battery-powered ZigBee RF4CE node, such as a remote control unit, should spend most of its time in a low-power mode to save energy. A number of low-power modes are available on a ZigBee RF4CE node, as introduced in Section 1.7.

Power savings can be made by carefully controlling when the radio receiver of the node is active (powered and ready to receive transmissions). The function **eRF4CE_NlmeRxEnableReq()** can be used to implement this control and provides three options:

- enable the receiver until further notice
- enable the receiver for a specified period of time (multiple of 16 µs)
- disable the receiver until further notice

In addition, the receiver can be put into a special 'power-saving' mode using this function (see Section 2.4.1). Alternatively, the JN516x device can be put into sleep mode (see Section 2.4.2) using the JN516x Integrated Peripherals API.

Note: Traditionally, a remote control unit is only considered to be a transmitter. However, in a ZigBee RF4CE system, the unit may also need to receive - for example, control message acknowledgements or data to be displayed on an integrated LCD screen.

2.4.1 Power-saving Mode

In power-saving mode, the receiver is disabled most of the time but is periodically enabled for a short time:

- The auto-enable duty cycle is configured through the NIB attribute *nwkDutyCycle*.
- The duration for which the receiver is active during the duty cycle is configured through the NIB attribute *nwkActivePeriod*.

Both of these times must be specified as a number of MAC symbols, where a MAC symbol represents 16 µs. NIB attributes are listed in Appendix A.2 and are set using the function **eRF4CE_NlmeSetReq()**.

Power-saving mode is enabled using the function **eRF4CE_NlmeRxEnableReq()** by setting the duration parameter to be equal to the configured value of the NIB attribute *nwkActivePeriod*.

2.4.2 Sleep Mode

The JN516x device can be put into sleep mode, which provides the largest possible power saving for a ZigBee RF4CE node. This low-power mode is implemented using the JN516x Integrated Peripherals API. The device may leave sleep mode automatically (using a wake timer) or as the result of a user action, such as a button-press (linked to a DIO input).

Note: For more information on sleep mode, refer to the *Integrated Peripherals API User Guide (JN-UG-3066)*.

To put a node to sleep, the application should incorporate the following steps:

1. Wait for any ongoing stack activity to complete (a confirmation or comm-status indication for a previous request or response).
2. If appropriate, switch off the radio receiver by calling **eRF4CE_NlmeRxEnableReq()** with parameter 0.
3. If not already done, configure the peripherals that will be used to wake the JN516x device, such as DIO pins or Wake Timer 0.
4. Call **vRF4CE_ImpSaveSettings()** with parameter E_SAVE_MODE_MINIMAL to store the frame counter value to non-volatile memory (Wake Timer 1 is used for this). It is not normally desirable to call this function with parameter E_SAVE_MODE_FULL for every sleep episode of a remote control unit, as this would cause a Flash memory erase and program cycle.
5. Call the Integrated Peripherals API function **vAHI_Sleep()**, specifying the required sleep mode:
   - E_AHI_SLEEP_OSCON_RAMOFF if the device will be woken by Wake Timer 0.
   - E_AHI_SLEEP_OSCOFF_RAMOFF if the device will not be woken by a wake timer - this is the lowest power sleep mode with the DIOs and wake timers still powered (Wake Timer 1 is still required for frame counter storage).

When the device wakes, it will enter the cold start function **AppColdStart()**, described in Section 2.3.1.

2.5 Using Application Profile Commands

A ZigBee RF4CE application can use the ZigBee Remote Control (ZRC) and ZigBee Input Device (ZID) application profiles, which both interface to the ZigBee RF4CE network layer. Please refer to *ZigBee RF4CE Demonstration Application note* (*JN-AN-1158*) for the implementation details of ZRC and ZID commands.

- **ZigBee Remote Control (ZRC)**: This profile defines commands and procedures to be used by consumer electronics remote control applications. The ZRC commands can be used by an application to send HDMI CEC command codes to a paired receiver. Refer to Section 3.5 for the implementation details of the ZRC commands.
- **ZigBee Input Device (ZID)**: This profile defines commands and procedures to facilitate the use of Human Interface Devices such as mice, touchpads and keyboards. The ZID commands can be used by an application to report the Human Interface Device communication messages to a paired receiver. Refer to Section 3.6 for the implementation details of the ZID commands.

Part II:
Reference Information

3. ZigBee RF4CE API Functions

This chapter details the C functions of the ZigBee RF4CE API. The functions are categorised into four groups:

- Implementation-specific functions - see Section 3.1
- NLDE functions - see Section 3.2
- NLME function - see Section 3.3
- Callback function - see Section 3.4

All of the above functions are defined in the header file **RF4CE_API.h**.

Note: The resources (constants, structures, enumerations) used by the API functions are defined in the same header file and detailed in the appendices of this manual.

3.1 Implementation-specific Functions

This section details the functions in the implementation-specific part of the API. These functions are concerned with initialising the ZigBee RF4CE stack and saving the stack settings in Flash memory.

The Implementation-specific functions are listed below, along with their page references:

bRF4CE_ImpInit

```
bool_t bRF4CE_ImpInit(uint8 u8NodeCapabilities,
                      uint16 u16VendorId,
                      uint8 *pu8VendorString,
                      uint32 u32FlashBaseAddr,
                      uint8 u8FlashSector);
```

Description

This function initialises the ZigBee RF4CE and IEEE 802.15.4 stack layers for operation. Note that before the stack can be used, a start request must be subsequently submitted by calling the function **vRF4CE_NlmeStartReq()**.

The function requires certain node and vendor information, as well as the location in Flash memory where the stack configuration settings will be stored when the function **vRF4CE_ImpSaveSettings()** is called.

The node's capabilities must be specified as a bitmap in the following format:

Bits	Field Name	Values
0	Node type	1 = Target node, 0 = Controller node
1	Power source	1 = AC mains supply, 0 = Other power source
2	Security capable	1 = Frame encryption available, 0 = No encryption
3	Channel normalisation capable	1 = Able to react to channel change request (submitted through channel designator in frame) 0 = Unable to react to channel change request
4-7	Reserved	

On returning, the function indicates whether any record was found in Flash memory indicating that the node was a member of a previous pairing or network.

Parameters

u8NodeCapabilities	Node's ZigBee RF4CE capabilities (see table above)
u16VendorId	Node's vendor ID
pu8VendorString	Pointer to node's vendor string (assumed to be 7 bytes long)
u32FlashBaseAddr	Start address in Flash memory where the stack configuration settings will be stored
u8FlashSector	Flash sector in which stack configuration settings will be stored (both address and sector are required, to support different sector sizes)

Returns

TRUE: No record of any previous pairing or network was found

FALSE: Information on previous pairing or network was found

vRF4CE_ImpSaveSettings

```
void vRF4CE_ImpSaveSettings(teSaveMode eSaveMode);
```

Description

This function saves the current stack configuration settings. There are two levels of save:

- Full save that stores all current values (NIB attributes, etc) to Flash memory
- Basic save that saves just the frame counter to non-volatile memory

Normally, the full save is used after the network and pairings have been set up, and the basic save is used for Controller nodes just before going to sleep.

Note that Wake Timer 1 on the JN516x device is used for the non-volatile storage of the frame counter in the basic save.

Parameters

eSaveMode	The level of the data save to perform: E_SAVE_MODE_FULL: All stack information to be written to Flash memory E_SAVE_MODE_MINIMAL: Only the frame counter to be saved to non-volatile memory

Returns

None

vRF4CE_ImpDestroySettings

```
void vRF4CE_ImpDestroySettings(
                                    uint32 u32FlashBaseAddr);
```

Description

This function invalidates the currently saved stack configuration settings in Flash memory.

If called before **bRF4CE_ImpInit()**, this allows the application to completely reset the device.

Parameters

u32FlashBaseAddr	Start address in Flash memory where the stack configuration settings are stored

Returns

None

3.2 NLDE Function

This section describes the function in the NLDE part of the API.

The NLDE function is listed below, along with its page reference:

vRF4CE_NldeDataReq

```
void vRF4CE_NldeDataReq(uint8 u8PairingRef,
                        uint8 u8ProfileId,
                        uint16 u16VendorId,
                        uint8 u8NsduLength,
                        uint8 *pu8Nsdu,
                        uint8 u8TxOptions);
```

Description

This function is used to send data to a paired peer node or broadcast to all peer nodes (all nodes that are able to receive the radio broadcast).

The data to be sent must be stored in memory as an array of bytes and a pointer must be provided to the start of this array. The pointer and memory buffer can be discarded once the call has completed, even if the transmission has not yet completed.

The transmission options include those described in Section 1.5 and also include the following: option to use destination IEEE (MAC) address or network (short) address; secured or unsecured transmission; option to include channel number in transmission; option to include vendor-specific data or no vendor-specific data. The options can be used in any combination.

Parameters

u8PairingRef	Index of pairing table entry for the destination node (ignored if the data is to be broadcast)
u8ProfileId	ZigBee RF4CE Profile ID
u16VendorId	Vendor ID for a vendor-specific frame (ignored if not a vendor-specific frame)
u8NsduLength	Length, in bytes, of the data to be sent
pu8Nsdu	Pointer to the array of data bytes to be sent.
u8TxOptions	Transmission options (the following values can be combined in a bitwise OR operation): RF4CE_TX_OPT_BROADCAST Included: Transmission is broadcast Excluded: Transmission is unicast to paired device RF4CE_TX_OPT_DEST_IEEE Included: Use IEEE (MAC) destination address Excluded: Use network (short) destination address RF4CE_TX_OPT_ACKNOWLEDGE Included: Request IEEE 802.15.4 acknowledgement Excluded: Unacknowledged transmission RF4CE_TX_OPT_SECURITY Included: Secured transmission Excluded: Unsecured transmission

Continued

RF4CE_TX_OPT_SINGLE_CHAN

Included: Transmit in single channel only
Excluded: Transmit in all channels, if appropriate

RF4CE_TX_OPT_SPECIFY_CHAN

Included: Include channel designator in transmission
Excluded: Do not include channel designator

RF4CE_TX_OPT_VENDOR_DATA

Included: Data is vendor-specific
Excluded: Data is not vendor-specific

Returns

Immediate: None

Callback Event: E_RF4CE_EV_NLDE_CFM

Callback Return Status

E_RF4CE_STATUS_SUCCESS

E_RF4CE_STATUS_NOT_PERMITTED

E_RF4CE_STATUS_INVALID_PARAMETER

E_RF4CE_STATUS_FRAME_COUNTER_EXPIRED

E_RF4CE_STATUS_NO_PAIRING

E_RF4CE_STATUS_NO_RESPONSE

802.15.4 error status

3.3 NLME Functions

This section describes the functions in the NLME part of the API.

The NLME functions are listed below, along with their page references:

vRF4CE_NlmeAutoDiscoveryReq

```
void vRF4CE_NlmeAutoDiscoveryReq(
                    tsRF4CE_AppCap *psRecAppCapabilities,
                    uint8 au8RecDevTypeList[],
                    uint8 au8RecProfileIdList[],
                    uint32 u32AutoDiscDuration);
```

Description

This function puts the node into auto-discovery mode. In this mode, the node will automatically respond to discovery requests that meet the supplied criteria.

The information that must be provided to this function includes a list of supported device types (e.g. television, set-top box) and a list of supported application profiles (e.g. CERC). In addition, the length of time that the node will remain in auto-discovery mode must be specified (as a multiple of 16 µs).

Parameters

psRecAppCapabilities	Pointer to the application capabilities structure for this node (structure is detailed in Appendix B.3)
au8RecDevTypeList[]	List of device types supported by this node. The device types are defined in the *ZigBee RF4CE Device Type List*. The number of entries in the list is supplied in *psRecAppCapabilities* and can be up to RF4CE_MAX_DEVICE_TYPE_LIST_LEN entries
au8RecProfileIdList[]	List of IDs of profiles supported by this node. The profile IDs are defined in the *ZigBee RF4CE Profile ID List*. The number of entries in the list is supplied in *psRecAppCapabilities* and can be up to RF4CE_MAX_PROFILE_ID_LIST_LEN entries
u32AutoDiscDuration	Maximum length of time that the node will stay in auto-discovery mode, in units of MAC symbols, where a MAC symbol represents 16 µs

Returns

Immediate: None

Callback Event: E_RF4CE_EV_AUTODISC_CFM

vRF4CE_NlmeDiscoveryReq

```
void vRF4CE_NlmeDiscoveryReq(
                    uint16 u16DstPanId,
                    uint16 u16DstNwkAddr,
                    tsRF4CE_AppCap *psOrgAppCapabilities,
                    uint8 au8OrgDevTypeList[],
                    uint8 au8OrgProfileIdList[],
                    uint8 u8SearchDevType,
                    uint8 u8DiscProfileIdListSize,
                    uint8 *pu8DiscProfileIdList,
                    uint32 u32DiscDuration);
```

Description

This function requests the start of a 'service discovery' operation.

The resulting discovery request can be sent to a particular PAN or to any PAN. The discovery request can also be sent to a particular node or to any node. In addition, the device type(s) to be searched for can be specified.

Concerning the subsequent responses, the length of time that the local node will wait for discovery responses (in each channel) must be specified (as a multiple of 16 µs). A list of application profiles is also required, against which the profile IDs in the received responses will be compared.

Parameters

u16DstPanId	PAN ID of the node(s) to search for. Can be set to 0xFFFF to indicate no preference
u16DstNwkAddr	Network (short) address of the node to search for. Can be set to 0xFFFF to indicate no preference
psOrgAppCapabilities	Pointer to the application capabilities structure for this node (structure is detailed in Appendix B.3)
au8OrgDevTypeList[]	List of device types supported by this node. The device types are defined in the *ZigBee RF4CE Device Type List*. The number of entries in the list is supplied in *psOrgAppCapabilities* and can be up to RF4CE_MAX_DEVICE_TYPE_LIST_LEN entries
au8OrgProfileIdList[]	List of IDs of profiles supported by this node. The number of entries in the list is supplied in *psOrgAppCapabilities* and can be up to RF4CE_MAX_PROFILE_ID_LIST_LEN entries
u8SearchDevType	Device type to discover, or 0xFF for no preference
u8DiscProfileIdListSize	Size of the list *pu8DiscProfileIdList*. This should not exceed RF4CE_MAX_DISC_PROFILE_ID_LIST_LEN. This is an implementation-specific limitation
pu8DiscProfileIdList	List of profile IDs to match with those in received discovery responses. The profile IDs are defined in the *ZigBee RF4CE Profile ID List*

u32DiscDuration	Maximum length of time that the node will wait for discovery responses on each channel, in units of MAC symbols, where a MAC symbol represents 16 µs

Returns

Immediate: None

Callback Event: E_RF4CE_EV_DISC_CFM

vRF4CE_NlmeDiscoveryResp

```
void vRF4CE_NlmeDiscoveryResp(
                    teRF4CE_Status eStatus,
                    tsIeeeAddr *psDstIeeeAddr,
                    tsRF4CE_AppCap *psRecAppCapabilities,
                    uint8 au8RecDevTypeList[],
                    uint8 au8RecProfileIdList[],
                    uint8 u8DiscReqLqi);
```

Description

This function is called to send a response to a received discovery request. It is not used if auto-discovery is operating.

The information that must be specified in this function, for inclusion in the discovery response, includes a list of supported device types (e.g. television, set-top box) and a list of supported application profiles (e.g. CERC). The perceived radio signal strength of the received discovery request is also specified.

Parameters

eStatus	Outcome of the request (success or no capacity).
psDstIeeeAddr	IEEE (MAC) address of the device that sent the discovery request
psRecAppCapabilities	Pointer to the application capabilities structure for the local node (structure is detailed in Appendix B.3)
au8RecDevTypeList[]	List of device types supported by the local node. The device types are defined in the *ZigBee RF4CE Device Type List*. The number of entries in the list is supplied in *psRecAppCapabilities* and can be up to RF4CE_MAX_DEVICE_TYPE_LIST_LEN entries
au8RecProfileIdList[]	List of IDs of profiles supported by the local node. The number of entries in the list is supplied in *psRecAppCapabilities* and can be up to RF4CE_MAX_PROFILE_ID_LIST_LEN entries
u8DiscReqLqi	LQI value (radio signal strength) of the received discovery request frame

Returns

Immediate: None

Callback Event: E_RF4CE_EV_COMMSTATUS_IND

eRF4CE_NlmeGetReq

```
teRF4CE_Status eRF4CE_NlmeGetReq(
                    teRF4CE_NibAttrib eNibAttribute,
                    uint8 u8NibAttributeIndex,
                    tuRF4CE_NibValue *puNibAttributeValue);
```

Description

This function retrieves the value of a NIB attribute from the stack.

The relevant attribute may be a table or array, in which case an index to the required entry must be specified.

Parameters

eNibAttribute	Identifier of the NIB attribute to read (enumerations are listed in Appendix A.2)
u8NibAttributeIndex	For attributes that are tables or arrays, this is the index of the table/array entry containing the value to be read. For other attributes, this parameter has no meaning
puNibAttributeValue	Pointer to a union of values for storing the read value (union is detailed in Appendix B.6)

Returns

E_RF4CE_STATUS_SUCCESS

E_RF4CE_STATUS_INVALID_INDEX

E_RF4CE_STATUS_UNSUPPORTED_ATTRIBUTE

vRF4CE_NlmePairReq

```
void vRF4CE_NlmePairReq(
                    uint8 u8LogicalChannel,
                    uint16 u16DstPanId,
                    tsIeeeAddr *psDstIeeeAddr,
                    tsRF4CE_AppCap *psOrgAppCapabilities,
                    uint8 *pu8OrgDevTypeList,
                    uint8 *pu8OrgProfileIdList,
                    uint8 u8KeyExTransferCount);
```

Description

This function initiates a pairing operation between the local node and the specified node - that is, sends a pairing request to the remote node.

The function is normally called following a 'service discovery'.

When the pairing request arrives, an E_RF4CE_EV_PAIR_IND event is generated on the remote node. The **vRF4CE_StackEvent()** callback function, which handles this event, must then call **vRF4CE_NlmePairResp()** to respond to the pairing request. When the response is received, an E_RF4CE_EV_PAIR_CFM event is generated on the requesting node.

Parameters

u8LogicalChannel	Channel used by node with which to pair (15, 20 or 25)
u16DstPanId	PAN ID of the node with which to pair
psDstIeeeAddr	IEEE (MAC) address of the node with which to pair
psOrgAppCapabilities	Pointer to the application capabilities structure for the local node (structure is detailed in Appendix B.3)
pu8OrgDevTypeList	List of device types supported by the local node. The device types are defined in the *ZigBee RF4CE Device Type List*. The number of entries in the list is supplied in *psOrgAppCapabilities* and can be up to RF4CE_MAX_DEVICE_TYPE_LIST_LEN entries
pu8OrgProfileIdList	List of IDs of the profiles supported by the local node. The number of entries in the list is supplied in *psOrgAppCapabilities* and can be up to RF4CE_MAX_PROFILE_ID_LIST_LEN entries
u8KeyExTransferCount	The number of transfers to be used during the link key exchange sequence, if security is required for this link

Returns

Immediate: None

Callback Event: E_RF4CE_EV_PAIR_CFM

vRF4CE_NlmePairResp

```
void vRF4CE_NlmePairResp(
                    teRF4CE_Status eStatus,
                    uint16 u16DstPanId,
                    tsIeeeAddr *psDstIeeeAddr,
                    tsRF4CE_AppCap *psRecAppCapabilities,
                    uint8 *pu8RecDevTypeList,
                    uint8 *pu8RecProfileIdList,
                    uint8 u8ProvPairingRef);
```

Description

This function initiates a reply to a received pairing request (from a remote node) by sending back a pairing response.

The function must be called by the callback function **vRF4CE_StackEvent()**, in order to handle an E_RF4CE_EV_PAIR_IND event which is generated when a pairing request is received.

When the response is received, an E_RF4CE_EV_PAIR_CFM event is generated on the requesting node, which confirms receipt of the response by returning a confirmation which, on arrival, produces an E_RF4CE_EV_COMMSTATUS_IND event.

Parameters

eStatus	Outcome of the request (success, no capacity or not permitted)
u16DstPanId	PAN ID of the node with which to pair
psDstIeeeAddr	IEEE (MAC) address of the node with which to pair
psRecAppCapabilities	Pointer to the application capabilities structure for the local node (structure is detailed in Appendix B.3)
pu8RecDevTypeList	List of device types supported by the local node. The device types are defined in the *ZigBee RF4CE Device Type List*. The number of entries in the list is supplied in *psRecAppCapabilities* and can be up to RF4CE_MAX_DEVICE_TYPE_LIST_LEN entries
pu8RecProfileIdList	List of IDs of profiles supported by the local node. The number of entries in the list is supplied in *psRecAppCapabilities* and can be up to RF4CE_MAX_PROFILE_ID_LIST_LEN entries.
u8ProvPairingRef	Provisional pairing entry reference number (index of entry in pairing table) if the pairing was accepted, or 0xFF otherwise

Returns

Immediate: None

Callback Event: E_RF4CE_EV_COMMSTATUS_IND

vRF4CE_NlmeResetReq

void vRF4CE_NlmeResetReq(bool_t *bSetDefaultNib***);**

Description

This function resets the ZigBee RF4CE Network layer and the IEEE 802.15.4 MAC layer to their initial states.

The function provides the option to reset the NIB as well as the stack. If the NIB is not reset, a saved copy of the NIB will be retrieved from Flash memory.

Parameters

bSetDefaultNib Indicates whether NIB is to be reset:
TRUE: Stack and NIB are both reset, and all pairing entries are discarded
FALSE: Stack is reset but not the NIB

Returns

None

eRF4CE_NlmeRxEnableReq

```
teRF4CE_Status eRF4CE_NlmeRxEnableReq(
                                    uint32 u32RxOnDuration);
```

Description

This function allows the radio receiver on the local node to be enabled/disabled. A number of enable/disable options are available. The receiver can also be enabled in power-saving mode.

The enable/disable options are:

- enable the receiver until further notice
- enable the receiver for a specified period of time (multiple of 16 µs)
- disable the receiver until further notice

For more information on the above options and power-saving mode, refer to Section 1.7 and Section 2.4.

Parameters

u32RxOnDuration Indicates the option/mode to be implemented:

0x000000: Disable receiver until further notice

0xFFFFFF: Enable receiver until further notice

Any other value: Enable receiver for specified time - value is interpreted as a number of MAC symbols, where a MAC symbol represents 16 µs. However, if specified value matches the value of the NIB attribute *nwkActivePeriod*, power-saving mode is enabled

Returns

E_RF4CE_STATUS_NOT_PERMITTED

E_RF4CE_STATUS_INVALID_PARAMETER

MAC status code

eRF4CE_NlmeSetReq

```
teRF4CE_Status eRF4CE_NlmeSetReq(
                    teRF4CE_NibAttrib eNibAtribute,
                    uint8 u8NibAttributeIndex,
                    tuRF4CE_NibValue *puNibAttributeValue);
```

Description

This function sets the value of a NIB attribute in the stack.

The relevant attribute may be a table or array, in which case an index to the required entry must be specified.

Parameters

eNibAttribute	Identifier of the NIB attribute to set (enumerations are listed in Appendix A.2)
u8NibAttributeIndex	For attributes that are tables or arrays, this is the index of the table/array entry to be written. For other attributes, this parameter has no meaning
puNibAttributeValue	Pointer to a union of values that holds the value to write (union is detailed in Appendix B.6)

Returns

E_RF4CE_STATUS_SUCCESS

E_RF4CE_STATUS_INVALID_INDEX

E_RF4CE_STATUS_UNSUPPORTED_ATTRIBUTE

vRF4CE_NlmeStartReq

```
void vRF4CE_NlmeStartReq(void);
```

Description

This function starts the local node. The actions taken depend on whether the node is a Controller node or a Target node, as defined by the device capabilities stored in the NIB.

Once initialisation has completed, an E_RF4CE_EV_START_CFM event is generated.

For information on node initialisation, refer to Section 1.4.1.

Parameters

None

Returns

Immediate: None

Callback Event: E_RF4CE_EV_START_CFM

vRF4CE_NlmeUnpairReq

void vRF4CE_NlmeUnpairReq(uint8 *u8PairingRef***);**

Description

This function requests that a paired node is removed from the local pairing table and that the equivalent pairing table entry is removed on the remote node.

Once the unpairing request has been successfully sent to the remote node, an E_RF4CE_EV_UNPAIR_CFM event is generated on the local node. To complete the unpairing, **vRF4CE_NlmeUnpairResp()** must be called on the remote node.

Note that once a pairing table entry has been removed, its index may be re-used for the next new entry in the table.

Parameters

u8PairingRef	Index of pairing table entry that is to be removed

Returns

Immediate: None

Callback Event: E_RF4CE_EV_UNPAIR_CFM

vRF4CE_NlmeUnpairResp

```
void vRF4CE_NlmeUnpairResp(uint8 u8PairingRef);
```

Description

This function allows the application to inform the ZigBee RF4CE stack that the specified entry must be removed from the local pairing table.

The function must be called by the callback **vRF4CE_StackEvent()**, in order to handle an E_RF4CE_EV_UNPAIR_IND event which is generated when an unpairing request is received.

Note that once a pairing table entry has been removed, its index may be re-used for the next new entry in the table.

Parameters

u8PairingRef	Index of the entry to be removed from the pairing table. If there is no corresponding entry, the request is ignored

Returns

None

eRF4CE_NlmeUpdateKeyReq

```
teRF4CE_Status eRF4CE_NlmeUpdateKeyReq(
                    uint8 u8PairingRef,
                    tsRF4CE_LinkKey *psNewLinkKey);
```

Description

This function performs an update of the security key associated with a local pairing table entry.

Parameters

u8PairingRef	Index of relevant pairing table entry
psNewLinkKey	Pointer to new key to be used for the specified pairing entry. The data can be discarded after the call has completed

Returns

E_RF4CE_STATUS_NO_PAIRING
E_RF4CE_STATUS_NOT_PERMITTED
E_RF4CE_STATUS_SUCCESS

3.4 Callback Function

This section describes the callback function that is used to pass events back to the application.

The callback function is listed below, along with its page reference:

vRF4CE_StackEvent

```
void vRF4CE_StackEvent(teRF4CE_EventType eEvent,
                       tuRF4CE_EventParam *puParam);
```

Description

This callback function is supplied by the application and is called by the stack when a stack event has occurred.

The function is usually called in interrupt context and the application should ensure that it processes events quickly or queues them for later processing. The stack is designed to clean up after itself before calling the callback function. It is therefore possible for a request into the stack to be made from within the callback function.

Parameters

eEvent	Type of event that has occurred (enumerations are listed in Appendix A.4)
puParam	Pointer to a union of structures containing further details about the event (union is detailed in Appendix B.19)

Returns

None

3.5 ZRC Command Frame Functions

The following functions are provided in the **ZRC.c/h** file to send ZRC commands for application use.

vZRC_SendUserControlPressed

```
void vZRC_SendUserControlPressed(
                    uint8 u8ReceiverPairingRef,
                    teRC_CmdCode eRC_CmdCode,
                    uint8 u8RC_CmdPayloadLen,
                    uint8 *pau8RC_CmdPayload);
```

Description

This function can be used to send a “user control pressed” command frame allowing a node to request a remote node to perform the specified RC (HDMI CEC) command. This command, containing the RC command code and payload, should be sent when a button is pressed. There should be a pairing reference available for the remote node in the pairing table of the sending device.

Parameters

u8ReceiverPairingRef	Sender's pairing reference in pairing table
eRC_CmdCode	RC command code which shall contain the HDMI CEC operand that corresponds to the user control being pressed
u8RC_CmdPayloadLen	Length of the RC command payload in bytes
pau8RC_CmdPayload	Pointer to the array containing the RC command payload (of variable size)

Returns

None

vZRC_SendUserControlRepeated

```
void vZRC_SendUserControlRepeated(
                    uint8 u8ReceiverPairingRef,
                    teRC_CmdCode eRC_CmdCode,
                    uint8 u8RC_CmdPayloadLen,
                    int8 *pau8RC_CmdPayload);
```

Description

This function can be used to send a “user control repeated” command frame to the receiver. This command, containing the RC command code and payload, should be sent when a button is held down. There should be a pairing reference available for the remote node in the pairing table of the sending device.

Parameters

u8ReceiverPairingRef	Sender's pairing reference in pairing table
eRC_CmdCode	RC command code which shall contain the HDMI CEC operand that corresponds to the user control being pressed
u8RC_CmdPayloadLen	Length of the RC command payload in bytes
pau8RC_CmdPayload	Pointer to the array containing the RC command payload (of variable size)

Returns

None

vZRC_SendUserControlReleased

```
void vZRC_SendUserControlReleased(
                    uint8 u8ReceiverPairingRef,
                    teRC_CmdCode eRC_CmdCode);
```

Description

This function can be used to send a “user control released” command frame allowing a node to notify a remote node that an RC command should be terminated following a “user control repeated” command frame. This command, containing the RC command code, should be sent when a button is released.

Parameters

u8ReceiverPairingRef	Sender's pairing reference in pairing table
eRC_CmdCode	RC command code which shall contain the HDMI CEC operand that corresponds to the user control being pressed

Returns

None

vZRC_SendCmdDiscRequest

```
void vZRC_SendCmdDiscRequest(
                    uint8 u8ReceiverPairingRef);
```

Description

This function can be used to send a “command discovery request” command frame allowing a node to query which user control commands are supported on a remote node. There should be a pairing reference available for the remote node in the pairing table of the sending device.

Parameters

u8ReceiverPairingRef	Sender's pairing reference in pairing table

Returns

None

vZRC_SendCmdDiscResponse

```
void vZRC_SendCmdDiscResponse(
                    uint8 u8ReceiverPairingRef,
                    uint8 *pau8CmdsSupported);
```

Description

This function can be used to send a “command discovery response” command frame allowing a node to respond to a “command discovery” request from a remote node, indicating which user control commands are supported. There should be a pairing reference available for the remote node in the pairing table of the sending device.

Parameters

u8ReceiverPairingRef	Sender's pairing reference in pairing table
pau8CmdsSupported	Pointer to the array of 32 bytes containing commands supported on the node

Returns

None

3.6 ZID Command Frame Functions

The following functions are provided in the **ZID.c/h** file to send ZID commands for application use.

vZID_SendGetReport

```
void vZID_SendGetReport(
                uint8 u8ReceiverPairingRef,
                teZID_ReportType eReportType,
                uint8 u8ReportId);
```

Description

This function can be used to send a “get report” command frame allowing the HID adaptor to request a report from a HID class device. There should be a pairing reference available for the remote node in the pairing table of the sending device.

Parameters

u8ReceiverPairingRef	Sender's pairing reference in pairing table
eReportType	Specifies the type of the record to be sent
eReportId	Specifies the report ID of the report being requested. Payload of variable size

Returns

None

vZID_SendReportData

```
void vZID_SendReportData(
                    uint8 u8ReceiverPairingRef,
                    uint8 u8ReportSize,
                    teZID_ReportType eReportType,
                    teZID_ReportID eReportId,
                    teZID_ReportData eReportData);
```

Description

This function can be used to send a “report data” command frame allowing a remote node to send an unsolicited report or to respond to a “get report” command frame. The implementation permits a single report data record to be sent of a fixed size. There should be a pairing reference available for the remote node in the pairing table of the sending device.

Parameters

u8ReceiverPairingRef	Sender's pairing reference in pairing table
u8ReportSize	Specifies the combined length of the eReportType, eReportID and eReportData parameters within the report data record
eReportType	Specifies the type of the record to be sent
eReportId	Specifies the report ID of the report being requested. Payload of variable size
eReportData	Contains the report data payload

Returns

None

4. ZigBee RF4CE API Resources

This chapter describe the enumerations, structures and constants used in the ZigBee RF4CE API. These resources are defined in the header file **RF4CE_API.h**.

4.1 Enumerations

4.1.1 teRF4CE_Status

Return values for calls into the ZigBee RF4CE stack:

Name	Value	Meaning
E_RF4CE_STATUS_SUCCESS	0x00	The requested operation was completed successfully
E_RF4CE_STATUS_NO_ORG_ CAPACITY	0xb0	A pairing link cannot be established since the originator node has reached its maximum number of entries in its pairing table
E_RF4CE_STATUS_NO_REC_ CAPACITY	0xb1	A pairing link cannot be established since the recipient node has reached its maximum number of entries in its pairing table
E_RF4CE_STATUS_NO_PAIRING	0xb2	A pairing table entry could not be found that corresponds to the supplied pairing reference (index)
E_RF4CE_STATUS_NO_ RESPONSE	0xb3	A response frame was not received within the timeout period defined by the NIB attribute *nwkResponseWaitTime*
E_RF4CE_STATUS_NOT_ PERMITTED	0xb4	A pairing request was denied by the recipient node or an attempt to update a security link key was not possible due to one or more nodes not supporting security
E_RF4CE_STATUS_DUPLICATE_ PAIRING	0xb5	A duplicate pairing table entry was detected following the receipt of a pairing request command frame
E_RF4CE_STATUS_FRAME_ COUNTER_EXPIRED	0xb6	The frame counter has reached its maximum value
E_RF4CE_STATUS_DISCOVERY_ ERROR	0xb7	Too many unique matched discovery request or valid response command frames were received than requested
E_RF4CE_STATUS_DISCOVERY_ TIMEOUT	0xb8	No discovery request or response command frames were received during discovery
E_RF4CE_STATUS_SECURITY_ TIMEOUT	0xb9	The security link key exchange or recovery procedure did not complete within the required time
E_RF4CE_STATUS_SECURITY_ FAILURE	0xba	A security link key was not successfully established between both ends of a pairing link
E_RF4CE_STATUS_INVALID_ PARAMETER	0xe8	A parameter in the primitive is either not supported or is out of the valid range
E_RF4CE_STATUS_ UNSUPPORTED_ATTRIBUTE	0xf4	A Set/Get request was issued with the identifier of a NIB attribute that is not supported
E_RF4CE_STATUS_INVALID_INDEX	0xf9	An attempt to write to a NIB attribute that is in a table failed because the specified table index was out of range

4.1.2 teRF4CE_NibAttrib

Attribute IDs for the NIB entries:

Name	**Value**	**Meaning** (from ZigBee RF4CE Spec)
E_RF4CE_NIB_ATTR_ACTIVE_PERIOD	0x60	*nwkActivePeriod*
E_RF4CE_NIB_ATTR_BASE_CHANNEL	0x61	*nwkBaseChannel*
E_RF4CE_NIB_ATTR_DISC_LQI_THRESHOLD	0x62	*nwkDiscoveryLQIThreshold*
E_RF4CE_NIB_ATTR_DISP_REP_INTERVAL	0x63	*nwkDiscoveryRepetitionInterval*
E_RF4CE_NIB_ATTR_DUTY_CYCLE	0x64	*nwkDutyCycle*
E_RF4CE_NIB_ATTR_FRAME_COUNTER	0x65	*nwkFrameCounter*
E_RF4CE_NIB_ATTR_IND_DISC_REQUESTS	0x66	*nwkIndicateDiscoveryRequests*
E_RF4CE_NIB_ATTR_IN_POWER_SAVE	0x67	*nwkInPowerSave*
E_RF4CE_NIB_ATTR_PAIRING_TABLE	0x68	*nwkPairingTable*
E_RF4CE_NIB_ATTR_MAX_DISC_REPETITIONS	0x69	*nwkMaxDiscoveryRepetitions*
E_RF4CE_NIB_ATTR_MAX_FIRST_CSMA_BACKOFFS	0x6a	*nwkMaxFirstAttemptCSMABackoffs*
E_RF4CE_NIB_ATTR_MAX_FIRST_RETRIES	0x6b	*nwkMaxFirstAttemptFrameRetries*
E_RF4CE_NIB_ATTR_MAX_REPORTED_NODE_DESCS	0x6c	*nwkMaxReportedNodeDescriptors*
E_RF4CE_NIB_ATTR_RESPONSE_WAIT_TIME	0x6d	*nwkResponseWaitTime*
E_RF4CE_NIB_ATTR_SCAN_DURATION	0x6e	*nwkScanDuration*
E_RF4CE_NIB_ATTR_USER_STRING	0x6f	*nwkUserString*

4.1.3 tePairState

Pairing state for pairing entries in the NIB:

Name	**Value**	**Meaning**
E_PAIR_EMPTY	0	Pairing entry is empty or invalid
E_PAIR_PROVISIONAL	1	Pairing entry is involved in the process of pairing
E_PAIR_ACTIVE	2	Pairing entry is valid and active

4.1.4 teRF4CE_EventType

Event types, as passed up to the callback function:

Name	Value	Meaning	Parameter structure to use
E_RF4CE_EV_NLDE_IND	0x00	NLDE indication	*tsRF4CE_NldeDataInd*
E_RF4CE_EV_NLDE_CFM	0x01	NLDE confirmation	*tsRF4CE_NldeDataCfm*
E_RF4CE_EV_AUTODISC_CFM	0x02	NLME Auto-discovery confirmation	*tsRF4CE_NlmeAutoDiscoveryCfm*
E_RF4CE_EV_COMMSTATUS_IND	0x03	NLME Comm-status indication	*tsRF4CE_NlmeCommStatusInd*
E_RF4CE_EV_DISC_IND	0x04	NLME Discovery indication	*tsRF4CE_NlmeDiscoveryInd*
E_RF4CE_EV_DISC_CFM	0x05	NLME Discovery confirmation	*tsRF4CE_NlmeDiscoveryCfm*
E_RF4CE_EV_PAIR_IND	0x06	NLME Pair indication	*tsRF4CE_NlmePairInd*
E_RF4CE_EV_PAIR_CFM	0x07	NLME Pair confirmation	*tsRF4CE_NlmePairCfm*
E_RF4CE_EV_START_CFM	0x08	NLME Start confirmation	*tsRF4CE_NlmeStartCfm*
E_RF4CE_EV_UNPAIR_IND	0x09	NLME Unpair indication	*tsRF4CE_NlmeUnpairInd*
E_RF4CE_EV_UNPAIR_CFM	0x0a	NLME Unpair confirmation	*tsRF4CE_NlmeUnpairCfm*

4.1.5 teSaveMode

Save mode options, for use with the function **vRF4CE_ImpSaveSettings()**:

Name	Value	Meaning
E_SAVE_MODE_FULL	0	Save all settings to Flash memory
E_SAVE_MODE_MINIMAL	1	Save frame counter only to non-volatile RAM

4.2 Structures and Unions

Note that elements in structures are not necessarily in the same order as found in the *ZigBee RF4CE Specification*. This is intentional as it allows the structures to be packed more efficiently to reduce RAM consumption.

4.2.1 tsIeeeAddr

IEEE (MAC) address:

Element	Meaning
uint32 u32H	Most significant 32 bits of the address
uint32 u32L	Least significant 32 bits of the address

4.2.2 tsRF4CE_LinkKey

Link key for use with secured paired links:

Element	Meaning
uint8 au8Key[16]	Array of sixteen 8-bit values containing the link key

4.2.3 tsRF4CE_AppCap

Application capabilities, as described in the *ZigBee RF4CE Specification*:

Element	Meaning
uint u1UserStringSpecified	1 if user string is specified, 0 if not (1-bit field)
uint u2NumSupportedDevTypes	Number of supported device types (2-bit field)
uint u1Reserved1	Unused (for padding)
uint u3NumSupportedProfTypes	Number of supported profile IDs (3-bit field)
uint u1Reserved2	Unused (for padding)

4.2.4 tsRF4CE_NodeDesc

Node descriptor, as described in the *ZigBee RF4CE Specification* and as used during discovery:

Element	Meaning
tsIeeeAddr sIeeeAddr	IEEE address of the node
tsRF4CE_AppCap sAppCapabilities	Application capabilities of the node
uint16 u16PanId	PAN ID of the node
uint16 u16VendorId	Vendor ID of the node
uint8 au8VendorString[RF4CE_VENDOR_STRING_LEN]	Vendor string of the node
uint8 au8UserString[RF4CE_USER_STRING_LEN]	User defined string of the node. If the *u1UserStringSpecified* sub-field of *sAppCapabilites* is 0, this element is undefined
uint8 au8DevTypeList[RF4CE_MAX_DEVICE_TYPE_LIST_LEN]	List of supported devices of the node
uint8 au8ProfileIdList[RF4CE_MAX_PROFILE_ID_LIST_LEN]	List of supported profile IDs of the node
uint8 u8LogicalChannel	Logical channel used by the node
uint8 u8NodeCapabilities	Capabilities of the node. The values in this field can be interpreted using the *RF4CE_NODECAP_xxx* constant definitions
uint8 u8DiscReqLqi	LQI of the discovery request frame from the node
teRF4CE_Status eStatus	Status of the discovery request

4.2.5 tsRF4CE_PairingTableEntry

Pairing table entry, as described in the *ZigBee RF4CE Specification* and as used in the NIB pairing table:

Element	Meaning
tsIeeeAddr sDstIeeeAddr	IEEE address of the destination device
tsRF4CE_LinkKey sSecurityLinkKey	Link key used to secure this link
uint32 u32RecFrameCounter	Frame counter most recently received from recipient node
uint16 u16SrcNwkAddr	Network address to be used by the source device
uint16 u16DstPanId	PAN ID of the destination device
uint16 u16DstNwkAddr	Network address of the destination device
uint8 u8DestLogicalChan	Logical channel used by the destination device
uint8 u8RecNodeCapabilities	Node capabilities of the recipient device. The values in this field can be interpreted using the *RF4CE_NODECAP_xxx* constant definitions.
tePairState eState	Status of the pairing entry (implementation-specific feature)

4.2.6 tuRF4CE_NibValue

Union of types used for NIB values (for passing values using the NIB 'Get' and 'Set' functions):

Element	**Meaning** (from ZigBee RF4CE Spec)
uint32 u32ActivePeriod	*nwkActivePeriod*
uint8 u8BaseChannel	*nwkBaseChannel*
uint8 u8DiscoveryLqiThreshold	*nwkDiscoveryLQIThreshold*
uint32 u32DiscoveryRepetitionInterval	*nwkDiscoveryRepetitionInterval*
uint32 u32DutyCycle	*nwkDutyCycle*
uint32 u32FrameCounter	*nwkFrameCounter*
bool_t bIndicateDiscoveryRequests	*nwkIndicateDiscoveryRequests*
bool_t bInPowerSave	*nwkInPowerSave*
tsRF4CE_PairingTableEntry sPairingTableEntry	*nwkPairingTable*. The NIB Get and Set functions only allow access to one table entry at a time
uint8 u8MaxDiscoveryRepetitions	*nwkMaxDiscoveryRepetitions*
uint8 u8MaxFirstAttemptCsmaBackoffs	*nwkMaxFirstAttemptCSMABackoffs*
uint8 u8MaxFirstAttemptFrameRetries	*nwkMaxFirstAttemptFrameRetries*
uint8 u8MaxReportedNodeDescriptors	*nwkMaxReportedNodeDescriptors*
uint32 u32ResponseWaitTime	*nwkResponseWaitTime*
uint8 u8ScanDuration	*nwkScanDuration*
uint8 au8UserString[RF4CE_USER_STRING_LEN]	*nwkUserString*

4.2.7 tuAddr

Union of IEEE (MAC) and network (short) addresses:

Element	**Meaning**
tsIeeeAddr sIeeeAddr	IEEE address
uint16 u16NwkAddr	Network address

4.2.8 tsRF4CE_NldeDataInd

Structure containing data for NLDE data indication events:

Element	Meaning
uint8 *pu8Nsdu	Pointer to the payload. This ceases to be valid after returning from the callback
uint16 u16VendorId	Vendor ID of the source node
uint8 u8PairingRef	Pairing reference for the node
uint8 u8ProfileId	Profile ID of the data frame
uint8 u8NsduLength	Payload length
uint8 u8RxLinkQuality	Link quality of the frame
uint8 u8RxFlags	Received frame characteristics. The values in this field can be interpreted using the *RF4CE_NLDE_RXFLAGS_xxx* constant definitions

4.2.9 tsRF4CE_NldeDataCfm

Structure containing data for NLDE data confirm events:

Element	Meaning
uint8 u8PairingRef	Pairing reference of the destination of the transmitted frame
teRF4CE_Status eStatus	Status of the transmission attempt

4.2.10 tsRF4CE_NlmeAutoDiscoveryCfm

Structure containing data for NLME auto-discovery confirm events:

Element	Meaning
tsIeeeAddr *psSrcIeeeAddr	Pointer to the IEEE address of the node that sent a discovery request to which this node responded
teRF4CE_Status eStatus	Status of the auto-discovery

4.2.11 tsRF4CE_NlmeCommStatusInd

Structure containing data for NLME comm-status indication events:

Element	Meaning
tuAddr uDstAddr	Address of the destination device
uint16 u16DstPanId	PAN ID of the destination device
uint8 u8PairingRef	Pairing table entry for the recipient node, or 0xff in the case that this comm-status is the result of a discovery response
uint8 u8DstAddrMode	Addressing mode of *uDstAddr*. 0 means network address, 1 means IEEE address
teRF4CE_Status eStatus	Status of the transmission

4.2.12 tsRF4CE_NlmeDiscoveryInd

Structure containing data for NLME discovery indication events:

Element	Meaning
tsRF4CE_AppCap sOrgAppCapabilities	Application capabilities of the discovery request originator
tsIeeeAddr *psSrcIeeeAddr	IEEE address of the discovery request originator
uint8 *pu8OrgVendorString	Vendor string of the discovery request originator. Length is RF4CE_VENDOR_STRING_LEN bytes
uint8 *pu8OrgUserString	User string of the discovery request originator. Length is RF4CE_USER_STRING_LEN bytes, and string is only valid if indicated in the *sOrgAppCapabilities* element
uint8 *pu8OrgDevTypeList	Supported device types of the discovery request originator
uint8 *pu8OrgProfileIdList	Supported profile IDs of the discovery request originator
uint16 u16OrgVendorId	Vendor ID of the discovery request originator
uint8 u8OrgNodeCapabilities	Node capabilities of the discovery request originator. The values in this field can be interpreted using the *RF4CE_NODECAP_xxx* constant definitions
uint8 u8SearchDevType	Device type being discovered, or 0xff if no preference has been indicated
uint8 u8RxLinkQuality	LQI value of the received discovery request frame
teRF4CE_Status eStatus	Status of the pairing table

4.2.13 tsRF4CE_NlmeDiscoveryCfm

Structure containing data for NLME discovery confirm events:

Element	Meaning
tsRF4CE_NodeDesc *psNodeDescList	Pointer to an array of node descriptors. The array will have no more than RF4CE_NWKC_MAX_NODE_DESC_LIST_SIZE entries
uint8 u8NumNodes	Number of entries in the *psNodeDescList* array
teRF4CE_Status eStatus	Status of the discovery attempt

4.2.14 tsRF4CE_NlmePairInd

Structure containing data for NLME pair indication events:

Element	Meaning
tsRF4CE_AppCap sOrgAppCapabilities	Application capabilities of the pair request originator
tsIeeeAddr *psSrcIeeeAddr	IEEE address of the pair request originator
uint16 u16SrcPanId	PAN ID of the pair request originator
uint16 u16OrgVendorId	Vendor ID of the pair request originator
uint8 *pu8OrgVendorString	Vendor string of the pair request originator
uint8 *pu8OrgUserString	User string of the pair request originator, if indicated in the *sOrgApp-Capabilities* element
uint8 *pu8OrgDevTypeList	Supported device type list of the pair request originator
uint8 *pu8OrgProfileIdList	Supported profile ID list of the pair request originator
uint8 u8OrgNodeCapabilities	Node capabilities of the pair request originator. The values in this field can be interpreted using the *RF4CE_NODECAP_xxx* constant definitions
uint8 u8KeyExTransferCount	The pairing originator's desired number of exchanges to perform during key exchange
uint8 u8ProvPairingRef	Provisional pairing reference, or 0xff if pairing table is full
teRF4CE_Status eStatus	Status of the provisional pairing

4.2.15 tsRF4CE_NlmePairCfm

Structure containing data for NLME pair confirm events:

Element	Meaning
tsRF4CE_AppCap sRecAppCapabilities	Application capabilities of the originator of the pair response
uint16 u16RecVendorId	Vendor ID of the originator of the pair response
uint8 u8PairingRef	Pairing reference for this pairing, or 0xff if pairing was unsuccessful
uint8 *pu8RecVendorString	Vendor string of the originator of the pair response
uint8 *pu8RecUserString	User string of the originator of the pair response, if indicated in *sRecAppCapabilities* element
uint8 *pu8RecDevTypeList	Supported device type list of the originator of the pair response
uint8 *pu8RecProfileIdList	Supported profile ID list of the originator of the pair response
teRF4CE_Status eStatus	Status of the pairing attempt

4.2.16 tsRF4CE_NlmeStartCfm

Structure containing data for NLME start confirm events:

Element	Meaning
teRF4CE_Status eStatus	Status of the start attempt

4.2.17 tsRF4CE_NlmeUnpairInd

Structure containing data for NLME unpair indication events:

Element	Meaning
uint8 u8PairingRef	Pairing table entry that has been unpaired

4.2.18 tsRF4CE_NlmeUnpairCfm

Structure containing data for NLME unpair confirm events:

Element	Meaning
teRF4CE_Status eStatus	Status of the unpair attempt
uint8 u8PairingRef	Pairing table entry that has been unpaired

4.2.19 tuRF4CE_EventParam

Union of the structures that may be returned in the callback function when a stack event occurs:

Element	Meaning
tsRF4CE_NldeDataInd sNldeDataInd	Structure to use with E_RF4CE_EV_NLDE_IND event
tsRF4CE_NldeDataCfm sNldeDataCfm	Structure to use with E_RF4CE_EV_NLDE_CFM event
tsRF4CE_NlmeAutoDiscoveryCfm sNlmeAutoDiscoveryCfm	Structure to use with E_RF4CE_EV_AUTODISC_CFM event
tsRF4CE_NlmeCommStatusInd sNlmeCommStatusInd	Structure to use with E_RF4CE_EV_COMMSTATUS_IND event
tsRF4CE_NlmeDiscoveryInd sNlmeDiscoveryInd	Structure to use with E_RF4CE_EV_DISC_IND event
tsRF4CE_NlmeDiscoveryCfm sNlmeDiscoveryCfm	Structure to use with E_RF4CE_EV_DISC_CFM event
tsRF4CE_NlmePairInd sNlmePairInd	Structure to use with E_RF4CE_EV_PAIR_IND event
tsRF4CE_NlmePairCfm sNlmePairCfm	Structure to use with E_RF4CE_EV_PAIR_CFM event
tsRF4CE_NlmeStartCfm sNlmeStartCfm	Structure to use with E_RF4CE_EV_START_CFM event
tsRF4CE_NlmeUnpairInd sNlmeUnpairInd	Structure to use with E_RF4CE_EV_UNPAIR_IND event
tsRF4CE_NlmeUnpairCfm sNlmeUnpairCfm	Structure to use with E_RF4CE_EV_UNPAIR_CFM event

4.3 Constants

4.3.1 RF4CE Implicit Constants

Constants defined throughout the *ZigBee RF4CE Specification* and used in the API:

Name	Value	Meaning
RF4CE_VENDOR_STRING_LEN	7	Length of vendor string, in bytes
RF4CE_USER_STRING_LEN	15	Length of user string, in bytes
RF4CE_MAX_DEVICE_TYPE_LIST_LEN	3	Maximum length of device type list
RF4CE_MAX_PROFILE_ID_LIST_LEN	7	Maximum length of profile ID list
RF4CE_MAX_DISC_PROFILE_ID_LIST_LEN	7	Maximum length of discovery profile ID list. Implementation-specific: the *ZigBee RF4CE Specification* suggests that this should be 255 but it has been limited for resource reasons

4.3.2 RF4CE Constants

Constants explicitly defined by the *ZigBee RF4CE Specification*:

Name	Value	Meaning (from ZigBee RF4CE Spec)
RF4CE_NWKC_CHANNEL_MASK	0x2108000	*nwkcChannelMask*
RF4CE_NWKC_FRAME_CNT_WINDOW	1024	*nwkcFrameCounterWindow*
RF4CE_NWKC_MAC_BCN_PAYLOAD_LEN	2	*nwkcMACBeaconPayloadLength*
RF4CE_NWKC_MAX_DUTY_CYCLE	62500	*nwkcMaxDutyCycle*
RF4CE_NWKC_MAX_KEY_SEED_WAIT_TIME	3750	*nwkcMaxKeySeedWaitTime*
RF4CE_NWKC_MAX_NODE_DESC_LIST_SIZE	8*	*nwkcMaxNodeDescListSize*
RF4CE_NWKC_MAX_PAIRING_TABLE_ENTRIES	8*	*nwkcMaxPairingTableEntries*
RF4CE_NWKC_MAX_SECURITY_TX_POWER	-15	*nwkcMaxSecCmdTxPower*
RF4CE_NWKC_MIN_ACTIVE_PERIOD	1050	*nwkcMinActivePeriod*
RF4CE_NWKC_MIN_CONT_PAIRING_TABLE_SIZE	1	*nwkcMinControllerPairingTableSize*
RF4CE_NWKC_MIN_NODE_DESC_LIST_SIZE	3	*nwkcMinNodeDescListSize*
RF4CE_NWKC_MIN_NWK_HDR_OVERHEAD	5	*nwkcMinNWKHeaderOverhead*
RF4CE_NWKC_MIN_TARG_PAIRING_TABLE_SIZE	5	*nwkcMinTargetPairingTableSize*
RF4CE_NWKC_PROTOCOL_ID	0xce	*nwkcProtocolIdentifier*
RF4CE_NWKC_PROTOCOL_VERSION	1	*nwkcProtocolVersion*

* Implementation-specific value

4.3.3 Node Capability Constants

Bit-field definitions for node capability values (constants should be bitwise ORed together to create the desired node capability value):

Name	Value	Meaning
RF4CE_NODECAP_TYPE_CONTROLLER	0	Node is a controller
RF4CE_NODECAP_TYPE_TARGET	1	Node is a target
RF4CE_NODECAP_PWRSRC_MAINS	2	Node has a mains power source
RF4CE_NODECAP_PWRSRC_NOT_MAINS	0	Node does not have a mains power source
RF4CE_NODECAP_SECURITY_CAPABLE	4	Node is capable of security
RF4CE_NODECAP_SECURITY_INCAPABLE	0	Node is not capable of security
RF4CE_NODECAP_CHANNORM_CAPABLE	8	Node is capable of performing channel normalisation
RF4CE_NODECAP_CHANNORM_INCAPABLE	0	Node is not capable of performing channel normalisation

4.3.4 Transmit Option Constants

Bit-field definitions for transmit options (constants should be bitwise ORed together to create the desired transmit option for use with an NLDE data request):

Name	Value	Meaning
RF4CE_TX_OPT_BROADCAST	1	Broadcast instead of unicast transmission
RF4CE_TX_OPT_DEST_IEEE	2	Use the destination node's IEEE address instead of its network address
RF4CE_TX_OPT_ACKNOWLEDGE	4	Request an 802.15.4-level acknowledgement
RF4CE_TX_OPT_SECURITY	8	Use security
RF4CE_TX_OPT_SINGLE_CHAN	16	Transmit only on a single channel, instead of trying all channels if the first channel fails
RF4CE_TX_OPT_SPECIFY_CHAN	32	Specify the preferred channel in the frame header
RF4CE_TX_OPT_VENDOR_DATA	64	Format frame as vendor-specific rather than standard RF4CE

4.3.5 Device Type Constants

Device type constants taken from the *ZigBee RF4CE Device Type List*:

Name	Value
RF4CE_DEVICE_TYPE_REMOTE_CONTROL	0x01
RF4CE_DEVICE_TYPE_TELEVISION	0x02
RF4CE_DEVICE_TYPE_PROJECTOR	0x03
RF4CE_DEVICE_TYPE_PLAYER	0x04
RF4CE_DEVICE_TYPE_RECORDER	0x05
RF4CE_DEVICE_TYPE_VIDEO_PLAYER_RECORDER	0x06
RF4CE_DEVICE_TYPE_AUDIO_PLAYER_RECORDER	0x07
RF4CE_DEVICE_TYPE_AUDIO_VIDEO_RECORDER	0x08
RF4CE_DEVICE_TYPE_SET_TOP_BOX	0x09
RF4CE_DEVICE_TYPE_HOME_THEATER	0x0a
RF4CE_DEVICE_TYPE_MEDIA_CENTER	0x0b
RF4CE_DEVICE_TYPE_GAME_CONSOLE	0x0c
RF4CE_DEVICE_TYPE_SATELLITE_RADIO	0x0d
RF4CE_DEVICE_TYPE_IR_EXTENDER	0x0e
RF4CE_DEVICE_TYPE_MONITOR	0x0f
RF4CE_DEVICE_TYPE_GENERIC	0xfe

Part III: Appendices

A. Enumerations

A.1 teRF4CE_Status

Return values for calls into the ZigBee RF4CE stack:

Name	Value	Meaning
E_RF4CE_STATUS_SUCCESS	0x00	The requested operation was completed successfully
E_RF4CE_STATUS_NO_ORG_ CAPACITY	0xb0	A pairing link cannot be established since the originator node has reached its maximum number of entries in its pairing table
E_RF4CE_STATUS_NO_REC_ CAPACITY	0xb1	A pairing link cannot be established since the recipient node has reached its maximum number of entries in its pairing table
E_RF4CE_STATUS_NO_PAIRING	0xb2	A pairing table entry could not be found that corresponds to the supplied pairing reference (index)
E_RF4CE_STATUS_NO_ RESPONSE	0xb3	A response frame was not received within the timeout period defined by the NIB attribute *nwkResponseWaitTime*
E_RF4CE_STATUS_NOT_ PERMITTED	0xb4	A pairing request was denied by the recipient node or an attempt to update a security link key was not possible due to one or more nodes not supporting security
E_RF4CE_STATUS_DUPLICATE_ PAIRING	0xb5	A duplicate pairing table entry was detected following the receipt of a pairing request command frame
E_RF4CE_STATUS_FRAME_ COUNTER_EXPIRED	0xb6	The frame counter has reached its maximum value
E_RF4CE_STATUS_DISCOVERY_ ERROR	0xb7	Too many unique matched discovery request or valid response command frames were received than requested
E_RF4CE_STATUS_DISCOVERY_ TIMEOUT	0xb8	No discovery request or response command frames were received during discovery
E_RF4CE_STATUS_SECURITY_ TIMEOUT	0xb9	The security link key exchange or recovery procedure did not complete within the required time
E_RF4CE_STATUS_SECURITY_ FAILURE	0xba	A security link key was not successfully established between both ends of a pairing link
E_RF4CE_STATUS_INVALID_ PARAMETER	0xe8	A parameter in the primitive is either not supported or is out of the valid range
E_RF4CE_STATUS_ UNSUPPORTED_ATTRIBUTE	0xf4	A Set/Get request was issued with the identifier of a NIB attribute that is not supported
E_RF4CE_STATUS_INVALID_INDEX	0xf9	An attempt to write to a NIB attribute that is in a table failed because the specified table index was out of range

A.2 teRF4CE_NibAttrib

Attribute IDs for the NIB entries:

Name	**Value**	**Meaning** (from ZigBee RF4CE Spec)
E_RF4CE_NIB_ATTR_ACTIVE_PERIOD	0x60	*nwkActivePeriod*
E_RF4CE_NIB_ATTR_BASE_CHANNEL	0x61	*nwkBaseChannel*
E_RF4CE_NIB_ATTR_DISC_LQI_THRESHOLD	0x62	*nwkDiscoveryLQIThreshold*
E_RF4CE_NIB_ATTR_DISP_REP_INTERVAL	0x63	*nwkDiscoveryRepetitionInterval*
E_RF4CE_NIB_ATTR_DUTY_CYCLE	0x64	*nwkDutyCycle*
E_RF4CE_NIB_ATTR_FRAME_COUNTER	0x65	*nwkFrameCounter*
E_RF4CE_NIB_ATTR_IND_DISC_REQUESTS	0x66	*nwkIndicateDiscoveryRequests*
E_RF4CE_NIB_ATTR_IN_POWER_SAVE	0x67	*nwkInPowerSave*
E_RF4CE_NIB_ATTR_PAIRING_TABLE	0x68	*nwkPairingTable*
E_RF4CE_NIB_ATTR_MAX_DISC_REPETITIONS	0x69	*nwkMaxDiscoveryRepetitions*
E_RF4CE_NIB_ATTR_MAX_FIRST_CSMA_BACKOFFS	0x6a	*nwkMaxFirstAttemptCSMABackoffs*
E_RF4CE_NIB_ATTR_MAX_FIRST_RETRIES	0x6b	*nwkMaxFirstAttemptFrameRetries*
E_RF4CE_NIB_ATTR_MAX_REPORTED_NODE_DESCS	0x6c	*nwkMaxReportedNodeDescriptors*
E_RF4CE_NIB_ATTR_RESPONSE_WAIT_TIME	0x6d	*nwkResponseWaitTime*
E_RF4CE_NIB_ATTR_SCAN_DURATION	0x6e	*nwkScanDuration*
E_RF4CE_NIB_ATTR_USER_STRING	0x6f	*nwkUserString*

A.3 tePairState

Pairing state for pairing entries in the NIB:

Name	**Value**	**Meaning**
E_PAIR_EMPTY	0	Pairing entry is empty or invalid
E_PAIR_PROVISIONAL	1	Pairing entry is involved in the process of pairing
E_PAIR_ACTIVE	2	Pairing entry is valid and active

A.4 teRF4CE_EventType

Event types, as passed up to the callback function:

Name	Value	Meaning	Parameter structure to use
E_RF4CE_EV_NLDE_IND	0x00	NLDE indication	*tsRF4CE_NldeDataInd*
E_RF4CE_EV_NLDE_CFM	0x01	NLDE confirmation	*tsRF4CE_NldeDataCfm*
E_RF4CE_EV_AUTODISC_CFM	0x02	NLME Auto-discovery confirmation	*tsRF4CE_NlmeAutoDiscoveryCfm*
E_RF4CE_EV_COMMSTATUS_IND	0x03	NLME Comm-status indication	*tsRF4CE_NlmeCommStatusInd*
E_RF4CE_EV_DISC_IND	0x04	NLME Discovery indication	*tsRF4CE_NlmeDiscoveryInd*
E_RF4CE_EV_DISC_CFM	0x05	NLME Discovery confirmation	*tsRF4CE_NlmeDiscoveryCfm*
E_RF4CE_EV_PAIR_IND	0x06	NLME Pair indication	*tsRF4CE_NlmePairInd*
E_RF4CE_EV_PAIR_CFM	0x07	NLME Pair confirmation	*tsRF4CE_NlmePairCfm*
E_RF4CE_EV_START_CFM	0x08	NLME Start confirmation	*tsRF4CE_NlmeStartCfm*
E_RF4CE_EV_UNPAIR_IND	0x09	NLME Unpair indication	*tsRF4CE_NlmeUnpairInd*
E_RF4CE_EV_UNPAIR_CFM	0x0a	NLME Unpair confirmation	*tsRF4CE_NlmeUnpairCfm*

A.5 teSaveMode

Save mode options, for use with the function **vRF4CE_ImpSaveSettings()**:

Name	Value	Meaning
E_SAVE_MODE_FULL	0	Save all settings to Flash memory
E_SAVE_MODE_MINIMAL	1	Save frame counter only to non-volatile RAM

B. Structures and Unions

Note that elements in structures are not necessarily in the same order as found in the *ZigBee RF4CE Specification*. This is intentional as it allows the structures to be packed more efficiently to reduce RAM consumption.

B.1 tsIeeeAddr

IEEE (MAC) address:

Element	Meaning
uint32 u32H	Most significant 32 bits of the address
uint32 u32L	Least significant 32 bits of the address

B.2 tsRF4CE_LinkKey

Link key for use with secured paired links:

Element	Meaning
uint8 au8Key[16]	Array of sixteen 8-bit values containing the link key

B.3 tsRF4CE_AppCap

Application capabilities, as described in the *ZigBee RF4CE Specification*:

Element	Meaning
uint u1UserStringSpecified	1 if user string is specified, 0 if not (1-bit field)
uint u2NumSupportedDevTypes	Number of supported device types (2-bit field)
uint u1Reserved1	Unused (for padding)
uint u3NumSupportedProfTypes	Number of supported profile IDs (3-bit field)
uint u1Reserved2	Unused (for padding)

B.4 tsRF4CE_NodeDesc

Node descriptor, as described in the *ZigBee RF4CE Specification* and as used during discovery:

Element	Meaning
tsIeeeAddr sIeeeAddr	IEEE address of the node
tsRF4CE_AppCap sAppCapabilities	Application capabilities of the node
uint16 u16PanId	PAN ID of the node
uint16 u16VendorId	Vendor ID of the node
uint8 au8VendorString[RF4CE_VENDOR_STRING_LEN]	Vendor string of the node
uint8 au8UserString[RF4CE_USER_STRING_LEN]	User defined string of the node. If the *u1UserStringSpecified* sub-field of *sAppCapabilites* is 0, this element is undefined
uint8 au8DevTypeList[RF4CE_MAX_DEVICE_TYPE_LIST_LEN]	List of supported devices of the node
uint8 au8ProfileIdList[RF4CE_MAX_PROFILE_ID_LIST_LEN]	List of supported profile IDs of the node
uint8 u8LogicalChannel	Logical channel used by the node
uint8 u8NodeCapabilities	Capabilities of the node. The values in this field can be interpreted using the *RF4CE_NODECAP_xxx* constant definitions
uint8 u8DiscReqLqi	LQI of the discovery request frame from the node
teRF4CE_Status eStatus	Status of the discovery request

B.5 tsRF4CE_PairingTableEntry

Pairing table entry, as described in the *ZigBee RF4CE Specification* and as used in the NIB pairing table:

Element	Meaning
tsIeeeAddr sDstIeeeAddr	IEEE address of the destination device
tsRF4CE_LinkKey sSecurityLinkKey	Link key used to secure this link
uint32 u32RecFrameCounter	Frame counter most recently received from recipient node
uint16 u16SrcNwkAddr	Network address to be used by the source device
uint16 u16DstPanId	PAN ID of the destination device
uint16 u16DstNwkAddr	Network address of the destination device
uint8 u8DestLogicalChan	Logical channel used by the destination device
uint8 u8RecNodeCapabilities	Node capabilities of the recipient device. The values in this field can be interpreted using the *RF4CE_NODECAP_xxx* constant definitions.
tePairState eState	Status of the pairing entry (implementation-specific feature)

B.6 tuRF4CE_NibValue

Union of types used for NIB values (for passing values using the NIB 'Get' and 'Set' functions):

Element	**Meaning** (from ZigBee RF4CE Spec)
uint32 u32ActivePeriod	*nwkActivePeriod*
uint8 u8BaseChannel	*nwkBaseChannel*
uint8 u8DiscoveryLqiThreshold	*nwkDiscoveryLQIThreshold*
uint32 u32DiscoveryRepetitionInterval	*nwkDiscoveryRepetitionInterval*
uint32 u32DutyCycle	*nwkDutyCycle*
uint32 u32FrameCounter	*nwkFrameCounter*
bool_t bIndicateDiscoveryRequests	*nwkIndicateDiscoveryRequests*
bool_t bInPowerSave	*nwkInPowerSave*
tsRF4CE_PairingTableEntry sPairingTableEntry	*nwkPairingTable*. The NIB Get and Set functions only allow access to one table entry at a time
uint8 u8MaxDiscoveryRepetitions	*nwkMaxDiscoveryRepetitions*
uint8 u8MaxFirstAttemptCsmaBackoffs	*nwkMaxFirstAttemptCSMABackoffs*
uint8 u8MaxFirstAttemptFrameRetries	*nwkMaxFirstAttemptFrameRetries*
uint8 u8MaxReportedNodeDescriptors	*nwkMaxReportedNodeDescriptors*
uint32 u32ResponseWaitTime	*nwkResponseWaitTime*
uint8 u8ScanDuration	*nwkScanDuration*
uint8 au8UserString[RF4CE_USER_STRING_LEN]	*nwkUserString*

B.7 tuAddr

Union of IEEE (MAC) and network (short) addresses:

Element	**Meaning**
tsIeeeAddr sIeeeAddr	IEEE address
uint16 u16NwkAddr	Network address

B.8 tsRF4CE_NldeDataInd

Structure containing data for NLDE data indication events:

Element	Meaning
uint8 *pu8Nsdu	Pointer to the payload. This ceases to be valid after returning from the callback
uint16 u16VendorId	Vendor ID of the source node
uint8 u8PairingRef	Pairing reference for the node
uint8 u8ProfileId	Profile ID of the data frame
uint8 u8NsduLength	Payload length
uint8 u8RxLinkQuality	Link quality of the frame
uint8 u8RxFlags	Received frame characteristics. The values in this field can be interpreted using the *RF4CE_NLDE_RXFLAGS_xxx* constant definitions

B.9 tsRF4CE_NldeDataCfm

Structure containing data for NLDE data confirm events:

Element	Meaning
uint8 u8PairingRef	Pairing reference of the destination of the transmitted frame
teRF4CE_Status eStatus	Status of the transmission attempt

B.10 tsRF4CE_NlmeAutoDiscoveryCfm

Structure containing data for NLME auto-discovery confirm events:

Element	Meaning
tsIeeeAddr *psSrcIeeeAddr	Pointer to the IEEE address of the node that sent a discovery request to which this node responded
teRF4CE_Status eStatus	Status of the auto-discovery

B.11 tsRF4CE_NlmeCommStatusInd

Structure containing data for NLME comm-status indication events:

Element	Meaning
tuAddr uDstAddr	Address of the destination device
uint16 u16DstPanId	PAN ID of the destination device
uint8 u8PairingRef	Pairing table entry for the recipient node, or 0xff in the case that this comm-status is the result of a discovery response
uint8 u8DstAddrMode	Addressing mode of *uDstAddr*. 0 means network address, 1 means IEEE address
teRF4CE_Status eStatus	Status of the transmission

B.12 tsRF4CE_NlmeDiscoveryInd

Structure containing data for NLME discovery indication events:

Element	Meaning
tsRF4CE_AppCap sOrgAppCapabilities	Application capabilities of the discovery request originator
tsIeeeAddr *psSrcIeeeAddr	IEEE address of the discovery request originator
uint8 *pu8OrgVendorString	Vendor string of the discovery request originator. Length is RF4CE_VENDOR_STRING_LEN bytes
uint8 *pu8OrgUserString	User string of the discovery request originator. Length is RF4CE_USER_STRING_LEN bytes, and string is only valid if indicated in the *sOrgAppCapabilities* element
uint8 *pu8OrgDevTypeList	Supported device types of the discovery request originator
uint8 *pu8OrgProfileIdList	Supported profile IDs of the discovery request originator
uint16 u16OrgVendorId	Vendor ID of the discovery request originator
uint8 u8OrgNodeCapabilities	Node capabilities of the discovery request originator. The values in this field can be interpreted using the *RF4CE_NODECAP_xxx* constant definitions
uint8 u8SearchDevType	Device type being discovered, or 0xff if no preference has been indicated
uint8 u8RxLinkQuality	LQI value of the received discovery request frame
teRF4CE_Status eStatus	Status of the pairing table

B.13 tsRF4CE_NlmeDiscoveryCfm

Structure containing data for NLME discovery confirm events:

Element	Meaning
tsRF4CE_NodeDesc *psNodeDescList	Pointer to an array of node descriptors. The array will have no more than RF4CE_NWKC_MAX_NODE_DESC_LIST_SIZE entries
uint8 u8NumNodes	Number of entries in the *psNodeDescList* array
teRF4CE_Status eStatus	Status of the discovery attempt

B.14 tsRF4CE_NlmePairInd

Structure containing data for NLME pair indication events:

Element	Meaning
tsRF4CE_AppCap sOrgAppCapabilities	Application capabilities of the pair request originator
tsIeeeAddr *psSrcIeeeAddr	IEEE address of the pair request originator
uint16 u16SrcPanId	PAN ID of the pair request originator
uint16 u16OrgVendorId	Vendor ID of the pair request originator
uint8 *pu8OrgVendorString	Vendor string of the pair request originator
uint8 *pu8OrgUserString	User string of the pair request originator, if indicated in the *sOrgApp-Capabilities* element
uint8 *pu8OrgDevTypeList	Supported device type list of the pair request originator
uint8 *pu8OrgProfileIdList	Supported profile ID list of the pair request originator
uint8 u8OrgNodeCapabilities	Node capabilities of the pair request originator. The values in this field can be interpreted using the *RF4CE_NODECAP_xxx* constant definitions
uint8 u8KeyExTransferCount	The pairing originator's desired number of exchanges to perform during key exchange
uint8 u8ProvPairingRef	Provisional pairing reference, or 0xff if pairing table is full
teRF4CE_Status eStatus	Status of the provisional pairing

B.15 tsRF4CE_NlmePairCfm

Structure containing data for NLME pair confirm events:

Element	Meaning
tsRF4CE_AppCap sRecAppCapabilities	Application capabilities of the originator of the pair response
uint16 u16RecVendorId	Vendor ID of the originator of the pair response
uint8 u8PairingRef	Pairing reference for this pairing, or 0xff if pairing was unsuccessful
uint8 *pu8RecVendorString	Vendor string of the originator of the pair response
uint8 *pu8RecUserString	User string of the originator of the pair response, if indicated in *sRecAppCapabilities* element
uint8 *pu8RecDevTypeList	Supported device type list of the originator of the pair response
uint8 *pu8RecProfileIdList	Supported profile ID list of the originator of the pair response
teRF4CE_Status eStatus	Status of the pairing attempt

B.16 tsRF4CE_NlmeStartCfm

Structure containing data for NLME start confirm events:

Element	Meaning
teRF4CE_Status eStatus	Status of the start attempt

B.17 tsRF4CE_NlmeUnpairInd

Structure containing data for NLME unpair indication events:

Element	Meaning
uint8 u8PairingRef	Pairing table entry that has been unpaired

B.18 tsRF4CE_NlmeUnpairCfm

Structure containing data for NLME unpair confirm events:

Element	Meaning
teRF4CE_Status eStatus	Status of the unpair attempt
uint8 u8PairingRef	Pairing table entry that has been unpaired

B.19 tuRF4CE_EventParam

Union of the structures that may be returned in the callback function when a stack event occurs:

Element	Meaning
tsRF4CE_NldeDataInd sNldeDataInd	Structure to use with E_RF4CE_EV_NLDE_IND event
tsRF4CE_NldeDataCfm sNldeDataCfm	Structure to use with E_RF4CE_EV_NLDE_CFM event
tsRF4CE_NlmeAutoDiscoveryCfm sNlmeAutoDiscoveryCfm	Structure to use with E_RF4CE_EV_AUTODISC_CFM event
tsRF4CE_NlmeCommStatusInd sNlmeCommStatusInd	Structure to use with E_RF4CE_EV_COMMSTATUS_IND event
tsRF4CE_NlmeDiscoveryInd sNlmeDiscoveryInd	Structure to use with E_RF4CE_EV_DISC_IND event
tsRF4CE_NlmeDiscoveryCfm sNlmeDiscoveryCfm	Structure to use with E_RF4CE_EV_DISC_CFM event
tsRF4CE_NlmePairInd sNlmePairInd	Structure to use with E_RF4CE_EV_PAIR_IND event
tsRF4CE_NlmePairCfm sNlmePairCfm	Structure to use with E_RF4CE_EV_PAIR_CFM event
tsRF4CE_NlmeStartCfm sNlmeStartCfm	Structure to use with E_RF4CE_EV_START_CFM event
tsRF4CE_NlmeUnpairInd sNlmeUnpairInd	Structure to use with E_RF4CE_EV_UNPAIR_IND event
tsRF4CE_NlmeUnpairCfm sNlmeUnpairCfm	Structure to use with E_RF4CE_EV_UNPAIR_CFM event

C. Constants

C.1 RF4CE Implicit Constants

Constants defined throughout the *ZigBee RF4CE Specification* and used in the API:

Name	Value	Meaning
RF4CE_VENDOR_STRING_LEN	7	Length of vendor string, in bytes
RF4CE_USER_STRING_LEN	15	Length of user string, in bytes
RF4CE_MAX_DEVICE_TYPE_LIST_LEN	3	Maximum length of device type list
RF4CE_MAX_PROFILE_ID_LIST_LEN	7	Maximum length of profile ID list
RF4CE_MAX_DISC_PROFILE_ID_LIST_LEN	7	Maximum length of discovery profile ID list. Implementation-specific: the *ZigBee RF4CE Specification* suggests that this should be 255 but it has been limited for resource reasons

C.2 RF4CE Constants

Constants explicitly defined by the *ZigBee RF4CE Specification*:

Name	**Value**	**Meaning** (from ZigBee RF4CE Spec)
RF4CE_NWKC_CHANNEL_MASK	0x2108000	*nwkcChannelMask*
RF4CE_NWKC_FRAME_CNT_WINDOW	1024	*nwkcFrameCounterWindow*
RF4CE_NWKC_MAC_BCN_PAYLOAD_LEN	2	*nwkcMACBeaconPayloadLength*
RF4CE_NWKC_MAX_DUTY_CYCLE	62500	*nwkcMaxDutyCycle*
RF4CE_NWKC_MAX_KEY_SEED_WAIT_TIME	3750	*nwkcMaxKeySeedWaitTime*
RF4CE_NWKC_MAX_NODE_DESC_LIST_SIZE	8*	*nwkcMaxNodeDescListSize*
RF4CE_NWKC_MAX_PAIRING_TABLE_ENTRIES	8*	*nwkcMaxPairingTableEntries*
RF4CE_NWKC_MAX_SECURITY_TX_POWER	-15	*nwkcMaxSecCmdTxPower*
RF4CE_NWKC_MIN_ACTIVE_PERIOD	1050	*nwkcMinActivePeriod*
RF4CE_NWKC_MIN_CONT_PAIRING_TABLE_SIZE	1	*nwkcMinControllerPairingTableSize*
RF4CE_NWKC_MIN_NODE_DESC_LIST_SIZE	3	*nwkcMinNodeDescListSize*
RF4CE_NWKC_MIN_NWK_HDR_OVERHEAD	5	*nwkcMinNWKHeaderOverhead*
RF4CE_NWKC_MIN_TARG_PAIRING_TABLE_SIZE	5	*nwkcMinTargetPairingTableSize*
RF4CE_NWKC_PROTOCOL_ID	0xce	*nwkcProtocolIdentifier*
RF4CE_NWKC_PROTOCOL_VERSION	1	*nwkcProtocolVersion*

* Implementation-specific value

C.3 Node Capability Constants

Bit-field definitions for node capability values (constants should be bitwise ORed together to create the desired node capability value):

Name	Value	Meaning
RF4CE_NODECAP_TYPE_CONTROLLER	0	Node is a controller
RF4CE_NODECAP_TYPE_TARGET	1	Node is a target
RF4CE_NODECAP_PWRSRC_MAINS	2	Node has a mains power source
RF4CE_NODECAP_PWRSRC_NOT_MAINS	0	Node does not have a mains power source
RF4CE_NODECAP_SECURITY_CAPABLE	4	Node is capable of security
RF4CE_NODECAP_SECURITY_INCAPABLE	0	Node is not capable of security
RF4CE_NODECAP_CHANNORM_CAPABLE	8	Node is capable of performing channel normalisation
RF4CE_NODECAP_CHANNORM_INCAPABLE	0	Node is not capable of performing channel normalisation

C.4 Transmit Option Constants

Bit-field definitions for transmit options (constants should be bitwise ORed together to create the desired transmit option for use with an NLDE data request):

Name	Value	Meaning
RF4CE_TX_OPT_BROADCAST	1	Broadcast instead of unicast transmission
RF4CE_TX_OPT_DEST_IEEE	2	Use the destination node's IEEE address instead of its network address
RF4CE_TX_OPT_ACKNOWLEDGE	4	Request an 802.15.4-level acknowledgement
RF4CE_TX_OPT_SECURITY	8	Use security
RF4CE_TX_OPT_SINGLE_CHAN	16	Transmit only on a single channel, instead of trying all channels if the first channel fails
RF4CE_TX_OPT_SPECIFY_CHAN	32	Specify the preferred channel in the frame header
RF4CE_TX_OPT_VENDOR_DATA	64	Format frame as vendor-specific rather than standard RF4CE

C.5 Device Type Constants

Device type constants taken from the *ZigBee RF4CE Device Type List*:

Name	Value
RF4CE_DEVICE_TYPE_REMOTE_CONTROL	0x01
RF4CE_DEVICE_TYPE_TELEVISION	0x02
RF4CE_DEVICE_TYPE_PROJECTOR	0x03
RF4CE_DEVICE_TYPE_PLAYER	0x04
RF4CE_DEVICE_TYPE_RECORDER	0x05
RF4CE_DEVICE_TYPE_VIDEO_PLAYER_RECORDER	0x06
RF4CE_DEVICE_TYPE_AUDIO_PLAYER_RECORDER	0x07
RF4CE_DEVICE_TYPE_AUDIO_VIDEO_RECORDER	0x08
RF4CE_DEVICE_TYPE_SET_TOP_BOX	0x09
RF4CE_DEVICE_TYPE_HOME_THEATER	0x0a
RF4CE_DEVICE_TYPE_MEDIA_CENTER	0x0b
RF4CE_DEVICE_TYPE_GAME_CONSOLE	0x0c
RF4CE_DEVICE_TYPE_SATELLITE_RADIO	0x0d
RF4CE_DEVICE_TYPE_IR_EXTENDER	0x0e
RF4CE_DEVICE_TYPE_MONITOR	0x0f
RF4CE_DEVICE_TYPE_GENERIC	0xfe

Revision History

Version	Date	Comments
1.0	27-Oct-2010	First release
1.1	06-Dec-2012	- Updated for the JN516x device - Added support for ZRC and ZID application profile commands

Important Notice

Limited warranty and liability - Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use - NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications - Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control - This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

NXP Laboratories UK
(Formerly Jennic Ltd)
Furnival Street
Sheffield
S1 4QT
United Kingdom

Tel: +44 (0)114 281 2655
Fax: +44 (0)114 281 2951

For the contact details of your local NXP office or distributor, refer to:

www.nxp.com/jennic

