ZigBee Remote Control (ZRC) Application Profile

Reference Manual

Document Number: ZRCAPRM
Rev. 1.2
2/2012

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

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

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

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

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

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

Contents

About This Book

This reference manual describes the Freescale Zigbee Remote Control (ZRC) Application Profile implementation which simplifies application development using the ZRC profile. This document replaces the Consumer Electronics Remote Control Applications Profile Reference Manual (CERCAPRM).

The Freescale ZRC profile resides on top of the BeeStack Consumer layer.

Audience

This reference manual is intended for application designers and users of the BeeStack Consumer protocol stack.

Organization

This document contains the following chapters:

Chapter 1 Freescale ZRC Application Profile Overview - Provides an introduction to the Freescale ZRC Application Profile implementation.

Chapter 2 Freescale ZRC Application Profile Software Usage - Provides a description of the Freescale ZRC Application Profile interfaces.

Revision History

The following table summarizes revisions to this manual since the previous release (Rev. 1.1).

Revision History

Date	Description / Location of Changes
March 2012	Minor changes throughout to support March software release.

Conventions

This document uses the following notational conventions:

- Courier monospaced type is used to identify commands, explicit command parameters, code examples, expressions, data types, and directives.
- Italic type is used for emphasis, to identify new terms, and for replaceable command parameters.

Definitions, Acronyms, and Abbreviations

The following list defines the abbreviations used in this document.

API	Application Programming Interface
NWK	Network Layer
NLDE	Network Layer Data Entity
NLME	Network Layer Management Entity
SAP	Service Access Point
SAP	Push Button Pair

Chapter 1
Freescale ZRC Application Profile Overview

This chapter presents a brief overview of the Freescale ZRC Application Profile software. This chapter details the primary concept of the profile implementation and provides an overview of the available libraries and the functionality each of them provide.

NOTE

This document refers only to HS08 platforms. For ARM platform legacy refer to documents from \ Documentation \ BeeStack Consumer Documents \ ARM Legacy Documents\.

1.1 Freescale ZRC Application Profile Introduction

To aid in the development of applications based on BeeStack Consumer, Freescale has developed an implementation of the ZRC application profile. In the protocol stack, the profile resides between the BeeStack Consumer layer and the application layer. The application can still access the network layer directly.

The profile implements the following functionality, available as separate libraries:

1. Controller side push-button pairing.
2. Target side push-button pairing.
3. ZRC command transmission and reception.

The following figure shows the software architecture of an application using the Freescale ZRC Application Profile Implementation.

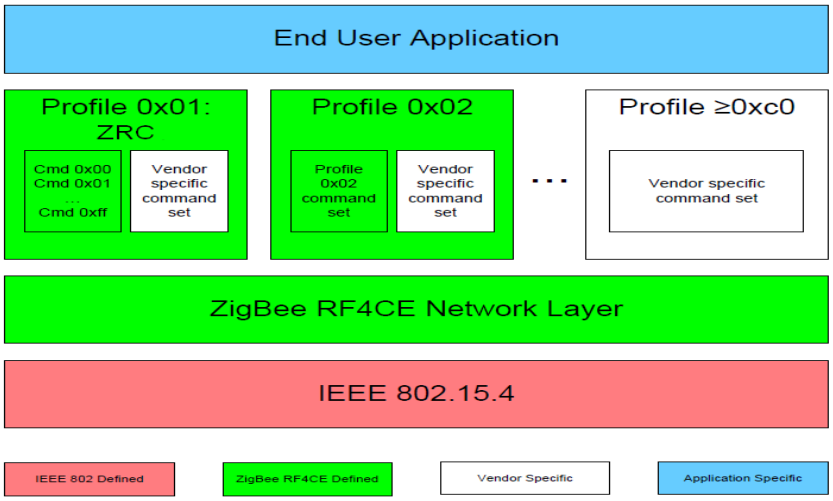

Figure 1-1. Application Software Protocol Stack

1.2 Freescale ZRC Application Profile Libraries

This section describes the suite of profile libraries and the functionality they implement. To use the profile, the application must link the profile framework library and the libraries implementing the desired functionality.

Table 1-1. Profile Libraries

Library	Description
RF4CE_PushButtonTask	Contains the framework to support all push button pair functionality. Must be included in all projects using the ZRC Profile.
RF4CE_PushButtonOrig	Contains the controller side Push-Button Pairing functionality, as defined by the ZRC specification. This library can be used to initiate the push button pair. A node using this feature will start the discovery request followed by the pair request. It can be used on both controller and target nodes.
RF4CE_PushButtonRecip	Contains the target side Push-Button Pairing functionality, as defined by the ZRC specification. A node using this feature will start the auto-discovery process followed by the waiting for a pair request. This library can be used to initiate the push button pair only from a target device.
RF4CE_ZRCProfile_CommandTxRx	Contains ZRC command transmission and reception functionality.

Chapter 2
Freescale ZRC Application Profile Software Usage

The profile implementation is able to perform the following tasks:

- Push Button Pairing (both controller and target side).
- ZRC command transmission and reception.

2.1 Service Specifications

The profile relies completely on the underlying network layer to perform its tasks. The ZRC profile uses the BeeStack Consumer API calls to pass data to the network layer and depends on indication and confirm messages to receive data from the network layer. The NLDE and NLME SAPs must be configured to redirect messages intended for profile layer to the profile SAPs, so that these don't erroneously reach the application. Refer to the ZigBee Remote Control Application Profile User's Guide for a description on how this is accomplished.

NOTE

The BeeStack Consumer network layer handles one request at a time, whether it comes directly from the application or from the profile. Care must be taken to ensure that the application and the profile never make a request to the network layer simultaneously.

The profile system is shown in Figure 2-1.

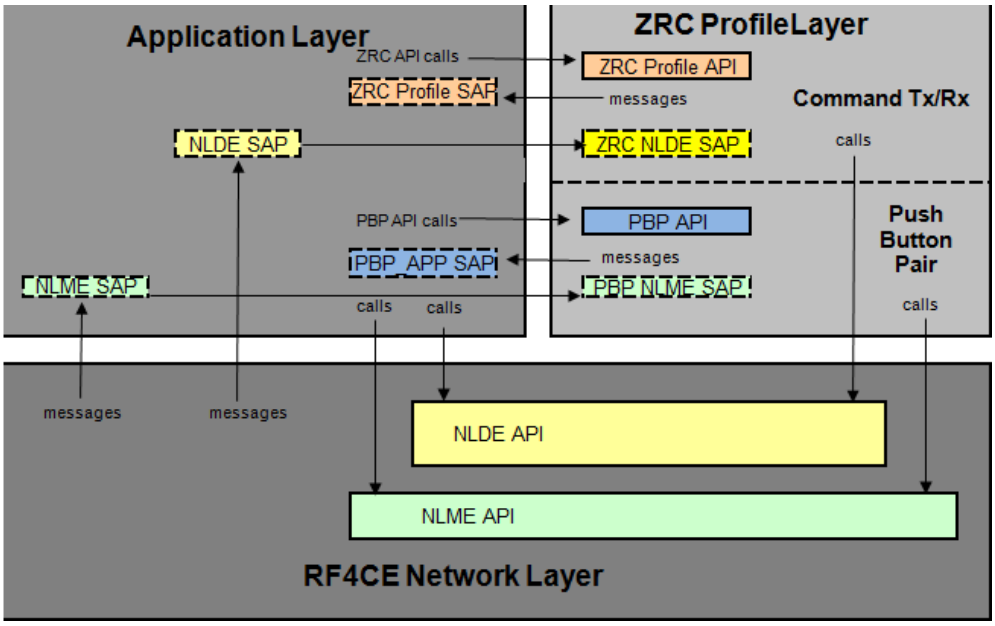

Figure 2-1. Freescale ZRC Profile layer components and Interfaces

NOTE

The Push-Button Pair (PBP) functionality is implemented as a separate framework and can be used as a stand alone layer by other profiles, such as ZID Profile. In this way, the ZRC profile layer is seen as being formed from two small sublayers that run their own tasks: the ZRC Command Tx/Rx sublayer and the Push Button Pair (PBP) sublayer. Note that this document describes the PBP procedures in conjunction with the ZRC profile. The ZRC applications should include the Push-Button Pair libraries and files implementing the desired functionality.

The profile provides three services, accessed through the API calls, which are listed in Table 2-1 and have the following characteristics:

- Profile API function calls configure or start the services
- The execution status is communicated to the application using confirm messages
- When information arrives over the network the profile layer informs the application layer using indication messages

The profile layer SAPs provides the interface for confirm and indication messages.

Table 2-1. Freescale ZRC Profile Services List

Freescale Profile Service	Initialization	Request	Indication	Response	Confirm
Originator side push-button pairing	Section 2.1.2	Section 2.1.3	Section 2.1.4	Section 2.1.5	Section 2.1.6
Recipient side push-button pairing	Section 2.1.7	Section 2.1.8	Section 2.1.9	Section 2.1.10	Section 2.1.11
Command transmission and reception	Section 2.1.14	Section 2.1.15	Section 2.1.17	-	Section 2.1.19

The profile layer reuses the error codes from the network layer. For a detailed list of all the error codes check the Freescale BeeStack Consumer Reference Manual. The description of each function call/message also includes a list of the error codes it can return and their significance.

2.1.1 PBP_IsIdle

PBP_IsIdle informs the application whether the PBP sublayer is idle. This function is included in the RF4CE_PushButtonTask library.

2.1.1.1 Prototype

PBP_IsIdle has the following prototype:

```
bool_t PBP_IsIdle(void);
```

PBP_IsIdle has no parameters.

The possible return values for the PBP_IsIdle API call are shown in the following table.

Table 2-2. PBP_IsIdle Parameters

Type	Possible Values	Description
bool_t	{FALSE;TRUE}	All possible return values are fully described in Section 2.1.13.3, “Effect on Receipt”.

2.1.1.2 Functionality

PBP_IsIdle is used to test whether the PBP sublayer is idle.

2.1.1.3 Effect on Receipt

On receipt of the PBP_IsIdle function call the PBPlayer checks its internal state to see whether it is idle. If it is idle, the function returns TRUE, otherwise it returns FALSE.

2.1.2 PBP_InitPushButtonPairOrig

PBP_InitPushButtonPairOrig initializes the push button pairing originator functionality in the PBP sublayer. To use this function the application must link to the RF4CE_PushButtonOrig library.

2.1.2.1 Prototype

PBP_InitPushButtonPairOrig has the following prototype:

```
uint8_t PBP_InitPushButtonPairOrig(void);
```

PBP_InitPushButtonPairOrig has no parameters.

The possible return values for the PBP_InitPushButtonPairOrig API call are shown in the following table.

Table 2-3. PBP_InitPushButtonPairOrig API Call Return Values

Type	Possible Values	Description
uint8_t	gNWSuccess_c gNWNoTimers_c	All possible return values are fully described in Section 2.1.2.3, “Effect on Receipt”.

2.1.2.2 Functionality

PBP_InitPushButtonPairOrig is used to enable the originator push-button pairing functionality in the PBP sublayer, so that the application can initiate originator side push-button pairing.

2.1.2.3 Effect on Receipt

On receipt of the PBP_InitPushButtonPairOrig function call the PBP sublayer configures itself to handle the push button pairing originator procedure (controller side push-button pairing as described in the ZRC profile specification). The function also tries to allocate a timer needed by the feature. If the timer could

not be allocated, the function exits with *gNWNoTimers_c* and configuration is aborted. Otherwise the function returns *gNWSuccess_c* and the originator push-button pairing functionality is now ready to be used.

2.1.3 PBP_PushButtonPairOrigRequest

PBP_PushButtonPairOrigRequest instructs the PBP sublayer to initiate the controller-side push-button pairing procedure. To use this function the application must link to the RF4CE_PushButtonOrig library.

2.1.3.1 Prototype

PBP_PushButtonPairOrigRequest has the following prototype:

```
uint8_t PBP_PushButtonPairOrigRequest(
                                        uint16_t          recipPanId,
                                        uint16_t          recipShortAddress,
                                        uint8_t           recipDeviceType,
                                        appCapabilities_t origAppCapabilities,
                                        uint8_t*          origDeviceTypeList,
                                        uint8_t*          origProfileIdList,
                                        uint8_t           discProfileIdListSize,
                                        uint8_t*          discProfileIdList,
                                        uint8_t           keyExTransferCount,
                                        bool_t            bRequestAppAcceptToPair,
                                        uint16_t          timeToWaitAppAcceptToPair
                                        );
```

The following table specifies the parameters for PBP_PushButtonPairOrigRequest.

Table 2-4. PBP_PushButtonPairOrigRequest Parameters

Name	Type	Valid Range	Description
recipPanId	uint16_t	0x0000 – 0xFFFF	The PAN ID of the target
recipShortAddress	uint16_t	0x0000 – 0xFFFF	The target's short address
recipDeviceType	uint8_t	-	The requested device type to discover
origAppCapabilities	appCapabilities_t	-	The current node's application capabilities
origDeviceTypeList	uint8_t*	-	The current node's device type list
origProfileIdList	uint8_t*	-	The current node's profile Id list
discProfileIdListSize	uint8_t	-	The size of the profile Id list to use in the discovery process
discProfileIdList	uint8_t*	-	The profile Id list to use in the discovery process
keyExTransferCount	uint8_t	aplcMinKeyExchangeTransferCount (0x03) – 0xff	The number of transfers to use for exchanging the encryption key.

Table 2-4. PBP_PushButtonPairOrigRequest Parameters (continued)

Name	Type	Valid Range	Description
bRequestApp AcceptToPair	bool_t	TRUE, FALSE	The application's option for requesting (or not) to accept pairing with a successfully discovered device
timeToWaitAppAccept ToPair	uint16_t	0x0000 – 0xFFFF	Time (in ms) to wait for the application to respond if it accepts to start the pair process after the discovery process of the Push Button Pair Originator procedure was successfully completed. Ignored if the bRequestAppAcceptToPair is set to FALSE

These are the parameters used both for the NLME discovery request and for the subsequent pair request.

The possible return values for the PBP_PushButtonPairOrigRequest API call are shown in the following table.

Table 2-5. PBP_PushButtonPairOrigRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDenied_c gNWSuccess_c	All possible return values are fully described in Section 2.1.3.3, "Effect on Receipt".

2.1.3.2 Functionality

PBP_PushButtonPairOrigRequest is used to initiate the originator side push-button pairing procedure.

2.1.3.3 Effect on Receipt

On receipt of PBP_PushButtonPairOrigRequest, the PBP sublayer first verifies if all the conditions to begin a push-button pairing process are met.

If the push-button pairing originator functionality is uninitialized the function exits with *gNWNotPermitted_c*. If the PBP sublayer is busy with another request, the function returns *gNWDenied_c*. If the keyExTransferCount parameter is not in the specified range, this function returns gNWInvalidParam_c.

Otherwise the controller-side push-button pairing procedure is initiated and the function returns *gNWSuccess_c*. The RF4CE network discovery related NIBs (the maximum reported node descriptors, the maximum number of discovery repetitions and the discovery repetition interval) are set according to the ZRC profile specification. This will overwrite any configuration of the discovery process done by the application. These NIBs are NOT restored at the end of the push-button pairing process.

NOTE

After a push-button pairing sequence, when the application wishes to start a discovery process of its own, it must re-configure the discovery related NIBs (as desired) as these have been overwritten by the PBP sublayer.

If a request is sent to the application that it wants to accept and start the pair process with a successfully discovered device (*bRequestAppAcceptToPair* set to TRUE), a Push-Button Pair Originator Continue Indication is sent by the profile to the application during the Push Button Pair process. The application must respond with a PBP_PushButtonPairOrigContinueResponse call within *timeToWaitAppAcceptToPair* ms whether it wants to continue with the pair or abort the Push Button Pair procedure. Otherwise (if the application does not request to accept and start the pair process) the PBP sublayer automatically starts the pair process without requesting application approval.

The application is notified about the completion of the controller-side push-button pairing process through a Push-Button Pair Originator Confirm message.

NOTE

The application must not modify the parameters that have been passed by pointer (i.e. origDeviceTypeList, origProfileIdList and discProfileIdList) until the push-button pairing process has ended.

2.1.4 Push-Button Pair Originator Continue Indication

The Push-Button Pair Originator Continue Indication message informs the application about the successful completion of the Discovery process of the Push-Button Pair Originator procedure, providing in the same time complete information about the discovered device. The scope of this message is to let the application decide whether it wants to continue the Push Button Pair procedure by pairing with the discovered device or not. This message is received only when the application starts the Push-Button Pair Originator procedure with the *bRequestAppAcceptToPair* parameter set to TRUE.

2.1.4.1 Message Structure

The Push-Button Pair Originator Continue Indication message has the following structure:

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

The PBP sublayer implementation uses for the Push-Button Pair Originator Continue Indication message the same structure as the one used for storing the RF4CE Discovery confirm node descriptor information.

The following table specifies the fields available in the Push-Button Pair Originator Continue Indication message.

Table 2-6. Push-Button Pair Originator Continue Indication Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c	Indicates the successful completion of the discovery process.
recipChannel	uint8_t	15, 20, 25	The expected channel of the discovered device.
recipPanId	uint8_t array with 2 elements	A valid PAN identifier	The PAN identifier of the discovered device.
recipMacAddress	uint8_t array with 8 elements]	A valid 802.15.04 IEEE address	The IEEE address of the discovered device..
recipCapabilities	uint8_t	-	The node capabilities of the discovered device.
recipVendorId	uint8_t array with 2 elements	A valid vendor identifier	The vendor ID of the discovered device.
recipVendorString	uint8_t array with 7 elements	A valid vendor string	The vendor string of the discovered device.
recipAppCapabilities	appCapabilities_t	-	The application capabilities of the discovered device.
recipUserString	uint8_t array with 15 elements	A valid user string	The user defined identification string of the discovered device.
recipDeviceTypeList	uint8_t array with 3 elements	Each integer: 0x00 – 0xfe	The list of device types supported by the discovered device
recipProfileIdList	uint8_t array with 7 elements	Each integer: 0x00 – 0xfe	The list of profile IDs supported by the discovered device
requestLQI	uint8_t	0x00 -0xff	The LQI of the discovery request command frame reported by the discovered device.

2.1.4.2 When Generated

The Push-Button Pair Originator Continue Indication message is generated by the PBP sublayer when the discovery process of a previously started push-button pairing Originator process has successfully completed, but only when the application explicitly requested this (*bRequestAppAcceptToPair* parameter set to TRUE).

2.1.4.3 Effect on Receipt

On receipt of the Push-Button Pair Originator Continue Indication message, the application layer is notified about the successful completion of the discovery process of the Push-Button Pairing Originator procedure. It must call PBP_PushButtonPairOrigContinueResponse service to inform the PBP sublayer whether it wants to continue the Push-Button Pair process by pairing with the discovered device or not.

2.1.5 PBP_PushButtonPairOrigContinueResponse

PBP_PushButtonPairOrigContinueResponse instructs the PBP sublayer to start or not the pair process of the Push-Button Pair Originator procedure. If the application does not want to continue with the pair process, the push-button pairing procedure is completed with the status *gNWNotPermitted_c*.

2.1.5.1 Prototype

PBP_PushButtonPairOrigContinueResponse has the following prototype:

```
uint8_t PBP_PushButtonPairOrigContinueResponse(
        bool_t   bContinue
);
```

The following table specifies the parameter for PBP_PushButtonPairOrigContinueResponse.

Table 2-7. PBP_PushButtonPairOrigContinueResponse Parameter

Name	Type	Valid Range	Description
bContinue	uint8_t	TRUE, FALSE	This parameter must be set to TRUE if the application accepts to pair with the successfully discovered device and must be set to FALSE otherwise.

The possible return values for the PBP_PushButtonPairOrigContinueResponse API call are shown in the following table.

Table 2-8. PBP_PushButtonPairOrigContinueResponse API Call Return Values

Type	Possible Values	Description
uint8_t	gNWDenied_c gNWSuccess_c	All possible return values are fully described in Section 2.1.4.3, “Effect on Receipt”.

2.1.5.2 Functionality

PBP_PushButtonPairOrigContinueResponse is used by the application to inform the PBP sublayer if it wants or not to begin the pair process of a previously started Push-Button Pair Originator procedure (controller-side) .

2.1.5.3 Effect on Receipt

On receipt of PBP_PushButtonPairOrigContinueResponse, the PBP sublayer first verifies if all the conditions to begin the pair process are met.

If the PBP sublayer is not expecting this kind of response (it is not in the situation when just finished with success the discovery process of a Push-Button Pair Originator procedure), the function exits with *gNWDenied_c* status.

Otherwise the function returns *gNWSuccess_c* and the profile either starts the pair process (if the *bContinue* parameter is set to TRUE) or drops the controller-side Push-Button Pairing procedure (if the *bContinue* parameter is set to FALSE) .

The application is notified of the completion of the controller-side push-button pairing process via a Push-Button Pair Originator Confirm message.

2.1.6 Push-Button Pair Originator Confirm

The Push-Button Pair Originator Confirm message informs the application about the completion of a push-button pairing originator procedure.

2.1.6.1 Message Structure

The Push-Button Pair Originator Confirm message has the following structure:

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

The PBP sublayer implementation reuses the RF4CE pair confirm message structure.

The following table specifies the fields available in the Push-Button Pair Originator Confirm message.

Table 2-9. Push-Button Pair Originator Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c or the network layer error code	Indicates either the successful completion of the push-button pairing process or identifies the error that has occurred.
deviceId	uint8_t	1–(gMaxPairTableEntries_c – 1)	The index of the pair table entry of the new pairing link
recipVendorId	uin8_t[2]	-	The recipient's vendor Id
recipVendorString	uint8_t*	-	The recipient's vendor string
recipAppCapabilities	appCapabilities_t	-	The recipient's application capabilities
recipUserString	uint8_t*	-	The recipient's user string
recipDeviceTypeList	uint8_t*	-	The recipient's list of supported device types
recipProfilesList	uint8_t*	-	The recipient's list of supported profiles

NOTE

If the status field has any value other than *gNWSuccess_c* than the push-button pairing process has not been completed successfully and all the other fields of the confirm message should be ignored.

2.1.6.2 When Generated

The Push-Button Pair Originator Confirm message is generated by the PBP sublayer when a previously started originator side push-button pairing process is complete.

2.1.6.3 Effect on Receipt

On receipt of the Push-Button Pair Originator Confirm message, the application layer is notified about the completion of the push-button pairing process.

2.1.7 PBP_InitPushButtonPairRecip

PBP_InitPushButtonPairRecip initializes the push button pairing recipient functionality in the PBP sublayer. To use this function the application must link to the RF4CE_PushButtonRecip library.

2.1.7.1 Prototype

PBP_InitPushButtonPairRecip has the following prototype:

```
uint8_t PBP_InitPushButtonPairRecip(void);
```

PBP_InitPushButtonPairRecip has no parameters.

The possible return values for the PBP_InitPushButtonPairRecip API call are shown in the following table.

Table 2-10. PBP_InitPushButtonPairRecip API Call Return Values

Type	Possible Values	Description
uint8_t	gNWSuccess_c gNWNoTimers_c	All possible return values are fully described in Section 2.1.7.3, “Effect on Receipt”.

2.1.7.2 Functionality

PBP_InitPushButtonPairRecip is used to enable the recipient push-button pairing functionality in the PBP sublayer, so that the application can initiate target-side push button pairing.

2.1.7.3 Effect on Receipt

On receipt of the PBP_InitPushButtonPairRecip function call the PBP sublayer configures itself to be able to handle the push button pairing recipient procedure (target side push-button pairing as described in the ZRC profile specification). The function also tries to allocate a timer needed by the feature. If the timer could not be allocated the function exits with *gNWNoTimers_c* and configuration is aborted. Otherwise the function returns *gNWSuccess_c* and the recipient push-button pairing functionality is now ready to be used.

2.1.8 PBP_PushButtonPairRecipRequest

PBP_PushButtonPairRecipRequest instructs the PBP sublayer to initiate the target-side push-button pairing procedure. To use this function the application must link to the RF4CE_PushButtonRecip library.

2.1.8.1 Prototype

PBP_PushButtonPairRecipRequest has the following prototype:

```
uint8_t PBP_PushButtonPairRecipRequest(
                                        appCapabilities_t origAppCapabilities,
                                        uint8_t*          origDeviceTypeList,
                                        uint8_t*          origProfileIdList,
                                        uint8_t           discLQIThreshold,
                                        bool_t            bRequestAppAcceptToPair,
                                        uint16_t          timeToWaitPairInd,
                                        uint16_t          timeToWaitAppAcceptToPair
                                        )
```

The following table specifies the parameters for PBP_PushButtonPairRecipRequest.

Table 2-11. PBP_PushButtonPairRecipRequest Parameters

Name	Type	Valid Range	Description
origAppCapabilities	appCapabilities_t	-	The current node's application capabilities.
origDeviceTypeList	uint8_t*	-	The current node's device type list.
origProfileIdList	uint8_t*	-	The current node's profile Id list.
discLQIThreshold	uint8_t	-	This is the minimum LQI that the received discovery frames should have to be processed and not dropped by the layer.
bRequestAppAcceptToPair	bool_t	TRUE, FALSE	The application's option whether or not to request to accept the pair process
timeToWaitPairInd	uint16_t	aplcMaxPairIndicationWaitTime (1000 ms) – 65535 ms	Time (in ms) that the profile will wait for a pair indication from the device that successfully discovered the current node.
timeToWaitAppAcceptToPair	uint16_t	0x0000 - 0xFFFF	Time (in ms) that the profile will wait for the application to respond if it accepts the pair process with the device that sent the pair request. Ignored if bRequestAppAcceptToPair is set to FALSE

These are the parameters used for the NLME auto-discovery request primitive.

The possible return values for the PBP_PushButtonPairRecipRequest API call are shown in the following table.

Table 2-12. PBP_PushButtonPairRecipRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDenied_c gNWSuccess_c	All possible return values are fully described in Section 2.1.8.3, “Effect on Receipt”.

2.1.8.2 Functionality

PBP_PushButtonPairRecipRequest is used to initiate the target side push-button pairing procedure.

2.1.8.3 Effect on Receipt

On receipt of PBP_PushButtonPairRecipRequest, the PBP sublayer first verifies if all the conditions to begin a push-button pairing process are met.

If the push-button pairing recipient functionality is uninitialized the function exits with *gNWNotPermitted_c*. If the PBP sublayer is busy with another request the function returns *gNWDenied_c*.

If the value of the timeToWaitPairInd parameter is less than aplcMaxPairIndicationWaitTime (1000 ms) this function sets the timeToWaitPairInd parameter to aplcMaxPairIndicationWaitTime. The parameter offers the possibility to wait for the Pair Indication longer than aplcMaxPairIndicationWaitTime (1000 ms).

Otherwise the target-side push-button pairing procedure is initiated and the function returns *gNWSuccess_c*. The activePeriod NIB is stored in an internal variable and the receiver is enabled indefinitely. The RF4CE discovery LQI threshold NIB is set to the value of discLQIThreshold required by the RF4CE ZRC profile specification. It is not restored at the end of the process.

NOTE

After a push-button pairing sequence, if the application needs to use another discovery process with a different value for the discovery LQI threshold, it must reconfigure the NIB because it is overwritten by the PBP sublayer.

If the application requires a request to accept and start the pair process with a device that was successfully discovered by the node, a Push-Button Pair Recipient Continue Indication is received and the application must respond with a PBP_PushButtonPairOrigContinueResponse call within *timeToWaitAppAcceptToPair* ms. Otherwise (if the application does not request to accept and start the pair process) the pair process is accepted without requesting application approval.

The application is notified of the completion of the target-side push-button pairing process through a Push-Button Pair Recipient Confirm message.

NOTE

The application must not modify the parameters that have been passed by pointer (i.e. origDeviceTypeList and origProfileIdList) until the push-button pairing process has ended.

2.1.9 Push-Button Pair Recipient Continue Indication

The Push-Button Pair Recipient Continue Indication message informs the application about the reception of a pair indication message coming from the same device that successfully discovered the current node while in the Push-Button Pair Recipient (target-side) procedure. This message is received only if the *bRequestAppAcceptToPair* parameter was set to TRUE.

2.1.9.1 Message Structure

The Push-Button Pair Recipient Continue Indication message has the following structure:

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

The PBP implementation uses for the Push-Button Pair Recipient Continue Indication message the same structure as the one used for the RF4CE Pair Indication message.

The following table specifies the fields available in the Push-Button Pair Recipient Continue Indication message.

Table 2-13. Push-Button Pair Recipient Continue Indication Message Structure

Field	Type	Possible Values	Description
Status	uint8_t	gNWSuccess_c,	The status of the pair indication message received by the PBP sublayer. Can only have one of 2 values: • *gNWSuccess_c* • gNWDuplicatePairing_c If the pairIndication was received by the PBP sublayer profile with the status set to *gNWNoRecipCapacity_c*, the profile will automatically send a pair response with the status set to *gNWNoRecipCapacity_c* to the device requesting the pair. In this case the application will not receive any Push-Button Pair Recipient Continue message, but only the Push-Button Pair Recipient Confirm message with the status set to *gNWNoRecipCapacity_c* after the pair response is sent.

Table 2-13. Push-Button Pair Recipient Continue Indication Message Structure

origPanId	uint8_t array with 2 elements		The PAN identifier of the device requesting the pair
origMacAddress	uint8_t array with 8 elements	A valid IEEE address	The IEEE address of the device requesting the pair
origCapabilities	uint8_t	-	The node capabilities of the device requesting the pair
origVendorId	uint8_t array with 2 elements	A valid Vendor ID	The Vendor Id of the device requesting the pair
origVendorString	uint8_t*	7 octets	The Vendor string of the device requesting the pair
origAppCapabilities	appCapabilities_t	-	The application capabilities of the device requesting the pair
origUserString	uint8_t*	NULL or 15 characters	The user defined identification string of the device requesting the pair
origDeviceTypeList	uint8_t*	Each integer: 0x00 – 0xfe	The list of device types supported by the device requesting the pair
origProfileIdList	uint8_t*	Each integer: 0x00 – 0xff	The list of profile IDs supported by the device requesting the pair.
keyExTransferCount	uint8_t	0x00 – 0xff	The number of transfers the target should use to exchange the link key with the pairing originator.
deviceId	uint8_t	0x00 – (nwkcMaxPairingTableEntries – 1), 0xff	Next free pairing reference that is used if this pairing process is successful.

2.1.9.2 When Generated

The Push-Button Pair Recipient Continue Indication message is generated by the PBP sublayer when the pair process of a previously started push-button pairing process (on target-side) begins (a successfully pair indication is received).

2.1.9.3 Effect on Receipt

On receipt of the Push-Button Pair Recipient Continue Indication message, the application layer is notified about the reception of a successful pair indication in the push-button pairing process and must call the PBP_PushButtonPairRecipContinueResponse function to accept or to not pair.

2.1.10 PBP_PushButtonPairRecipContinueResponse

PBP_PushButtonPairRecipContinueResponse instructs the PBP sublayer whether or not to accept the pair process of the target-side push-button pairing procedure (if the application does not want to accept the pair process, the push-button pairing procedure finishes with the *gNWNotPermitted_c* status).

Prototype

PBP_PushButtonPairRecipContinueResponse has the following prototype:

```
uint8_t PBP_PushButtonPairRecipContinueResponse(
        bool_t   bContinue
);
```

The following table specifies the parameter for PBP_PushButtonPairRecipContinueResponse.

Table 2-14. PBP_PushButtonPairRecipContinueResponse Parameter

Name	Type	Valid Range	Description
bContinue	uint8_t	TRUE, FALSE	This parameter must be set to TRUE if the application accepts the pair and must be set to FALSE otherwise.

The possible return values for the PBP_PushButtonPairRecipContinueResponse API call are shown in the following table.

Table 2-15. PBP_PushButtonPairOrigContinueResponse API Call Return Values

Type	Possible Values	Description
uint8_t	gNWDenied_c gNWSuccess_c	All possible return values are fully described in Section 2.1.8.3, “Effect on Receipt”.

2.1.10.1 Functionality

PBP_PushButtonPairRecipContinueResponse is used by the application to inform the PBP sublayer whether or not to accept the pair process of a previously started push-button pairing procedure (on target-side) .

2.1.10.2 Effect on Receipt

On receipt of the PBP_PushButtonPairRecipContinueResponse, the PBP sublayer first verifies if all the conditions are met.

If the PBP sublayer does not expect this response (it has not received a successful pair indication in the push-button pairing procedure), the function exits with *gNWDenied_c status.*

Otherwise the function returns *gNWSuccess_c* and the profile either starts the pair process (if the *bContinue* parameter is set to TRUE) or drops the target-side Push-Button Pairing procedure (if the *bContinue* parameter is set to FALSE) .

The application is notified of the completion of the target-side push-button pairing process through a Push-Button Pair Recipient Confirm message.

2.1.11 Push-Button Pair Recipient Confirm

The Push-Button Pair Recipient Confirm message informs the application about the completion of a push-button pairing recipient procedure.

2.1.11.1 Message Structure

The Push-Button Pair Recipient Confirm message has the following structure:

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

The PBP implementation reuses the RF4CE pair indication message structure

The following table specifies the fields available in the Push-Button Pair Recipient Confirm message.

Table 2-16. Push-Button Pair Recipient Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c or the network layer error code	Indicates either the successful completion of the push-button pairing process or identifies the error that has occured.
origPanId	uin8_t[2]	-	The originator's PanId
origMacAddress	uin8_t[8]	-	The originator's MAC address
origCapabilities	uint8_t	-	The originator's capabilities
origVendorId	uin8_t[2]	-	The originator's vendor Id
origVendorString	uint8_t*	-	The originator's vendor string
origAppCapabilities	appCapabilities_t	-	The originator's application capabilities
origUserString	uint8_t*	-	The originator's user string
origDeviceTypeList	uint8_t*	-	The originator's list of supported device types
origProfilesList	uint8_t*	-	The originator's list of supported profiles

Table 2-16. Push-Button Pair Recipient Confirm Message Structure

keyExTransferCount	uint8_t	0x00 – 0xff	The number of transfers the target should use to exchange the link key with the pairing originator.
deviceId	uint8_t	1–(gMaxPairTableEntries_c – 1)	The index of the pair table entry of the new pairing link

NOTE

If the status field has any value other than *gNWSuccess_c* than the push-button pairing process has not been completed successfully and all the other fields of the confirm message should be ignored.

2.1.11.2 When Generated

The Push-Button Pair Recipient Confirm message is generated by the PBP sublayer when a previously started controller-side push-button pairing process is complete.

2.1.11.3 Effect on Receipt

On receipt of the Push-Button Pair Recipient Confirm message, the application layer is notified about the completion of the push-button pairing process. The activePeriod NIB is restored to the initial value and the PBP sublayer is ready to accept new requests.

2.1.12 PBP_AbortProcess

The PBP_AbortProcess function call instructs the PBP sublayer to abort any ongoing process. This function is included in the RF4CE_PushButtonTask library.

2.1.12.1 Function Prototype

PBP_AbortProcess has the following prototype:

```
uint8_t PBP_AbortProcess(void);
```

PBP_AbortProcess has no parameters

The possible return values for the PBP_AbortProcess API call are shown in the following table.

Table 2-17. PBP_AbortProcess API Call Return Values

Type	Possible Values	Description
uint8_t	gNWDenied_c gNWNotPermitted_c gNWSuccess_c	All possible return values are fully described in Section 2.1.19.3, “Effect on Receipt”.

2.1.12.2 Functionality

PBP_AbortProcess allows the application to terminate any ongoing PBP process (i.e. push-button pairing, both target and originator side).

2.1.12.3 Effect on Receipt

Upon receipt of PBP_AbortProcess the PBP sublayer first checks whether a process is currently running. If the PBP sublayer is idle the function exits with *gNWDenied_c*.

Otherwise, if the network is in a pair request process (for the originator) or in a pair response process (for the recipient) the function exits with *gNWNotPermitted_c* status because the pair process can not be aborted.

Otherwise an event is sent to the PBP task to abort its current process and the function returns *gNWSuccess_c*. The process is aborted the next time the PBP task is run by the task scheduler.

2.1.13 ZRCProfile_IsIdle

ZRCProfile_IsIdle informs the application whether the ZRC Command Tx/Rx sublayer is idle. This function is included in the RF4CE_ZRCProfile_CommandTxRx library.

2.1.13.1 Prototype

ZRCProfile_IsIdle has the following prototype:

```
bool_t ZRCProfile_IsIdle(void);
```

ZRCProfile_IsIdle has no parameters.

The possible return values for the ZRCProfile_IsIdle API call are shown in the following table.

Table 2-18. ZRCProfile_IsIdle Parameters

Type	Possible Values	Description
bool_t	{FALSE;TRUE}	All possible return values are fully described in Section 2.1.13.3, “Effect on Receipt”.

2.1.13.2 Functionality

ZRCProfile_IsIdle is used to test whether the ZRC Command Tx/Rx sublayer is idle.

2.1.13.3 Effect on Receipt

On receipt of the ZRCProfile_IsIdle function call the ZRC Command Tx/Rx sublayer checks its internal state to see whether it is idle. If it is idle, the function returns TRUE, otherwise it returns FALSE.

2.1.14 ZRCProfile_InitCommandTxRx

ZRCProfile_InitCommandTxRx initializes the ZRC command transmission functionality in the ZRC profile. To use this function the application must link to the RF4CE_ZRCProfile_CommandTxRx library.

2.1.14.1 Prototype

ZRCProfile_InitCommandTxRx has the following prototype:

```
void ZRCProfile_InitCommandTxRx(void);
```

ZRCProfile_InitCommandTxRx has no parameters.

ZRCProfile_InitCommandTxRx does not return any value.

2.1.14.2 Functionality

ZRCProfile_InitCommandTxRx is used to enable the command transmission and reception functionality in the ZRC Command Tx/Rx sublayer, so that the application can send ZRC profile data commands without having to create a ZRC compliant NLDE_DataRequest payload itself.

2.1.14.3 Effect on Receipt

On receipt of the ZRCProfile_InitCommandTxRx function call the ZRC Command Tx/Rx sublayer configures itself to be able to handle the ZRC command transmission.

2.1.15 ZRCProfile_CommandRequest

ZRCProfile_CommandRequest instructs the ZRC profile layer to transmit a ZRC command to a paired device. To use this function the application must link to the RF4CE_ZRCProfile_CommandTxRx library.

2.1.15.1 Prototype

ZRCProfile_CommandRequest has the following prototype:

```
uint8_t ZRCProfile_CommandRequest(
                                  uint8_t   deviceId,
                                  uint8_t   commandCode,
                                  uint8_t   command,
                                  uint8_t*  vendorId,
                                  uint8_t   payloadLength,
                                  uint8_t*  payload,
                                  uint8_t   txOptions
                                  );
```

The following table specifies the parameters for ZRCProfile_CommandRequest.

Table 2-19. ZRCProfile_CommandRequest Parameters

Name	Type	Valid Range	Description
deviceId	uint8_t	-	The pair table entry index of the command recipient
commandCode	uint8_t	gZRC_CmdCode_UserCtrlPressed_c gZRC_CmdCode_UserCtrlReleased_c gZRC_CmdCode_DiscoveryRequest_c gZRC_CmdCode_UserCtrlPressedAndRepeat_c	The ZRC command code. IfgZRC_CmdCode_UserCtrlPressedAndRepeat_c is used (e.g. when an RC button is held down), the profile sends the User Control Pressed command frame and starts the repetition of the specified RC command ID (sending the User Control Repeated command frames). Otherwise, the function sends the specified user control command code.
command	uint8_t	-	The ZRC command ID (command code dependent). If the command code is: - gZRC_CmdCode_UserCtrlPressed_c, specifies the command Id of the pressed key. - gZRC_CmdCode_UserCtrlReleased_c, specifies the command Id of the released key. - gZRC_CmdCode_DiscoveryRequest_c, this parameter is ignored. -gZRC_CmdCode_UserCtrlPressedAndRepeat_c, specifies the command Id of the pressed and repeated key.
vendorId	uint8_t*	-	The vendor Id to put in the RF4CE data request frame
payloadLength	uint8_t	-	The ZRC command payload length
payload	uint8_t*	-	The ZRC command payload
txOptions	uint8_t	-	RF4CE Transmission options, as described by the ZigBee RF4CE specification; these are passed to the BeeStack Consumer layer

The possible return values for the ZRCProfile_CommandRequest API call are shown in the following table.

Table 2-20. ZRCProfile_CommandRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDenied_c gNWNoMemory_c gNWSuccess_c gNWInvalidParam_c	All possible return values are fully described in Section 2.1.15.3, “Effect on Receipt”.

2.1.15.2 Functionality

ZRCProfile_CommandRequest is used to transmit a ZRC command to a paired device.

2.1.15.3 Effect on Receipt

On receipt of ZRCProfile_CommandRequest, the ZRC Command Tx/Rx sublayer first verifies if all the conditions to send a ZRC command are met.

If the ZRC command transmission functionality is uninitialized the function exits with *gNWNotPermitted_c*. If the ZRC Command Tx/Rx sublayer is busy with another request the function returns *gNWDenied_c*. If no memory buffer could be allocated to construct the NLDE data request payload the function exits with *gNWNoMemory_c*.

If the data is not vendor specific (i.e. the vendor specific data bit in the TxOptions field is not set), the function performs parameters validation (as described in the ZRCProfile_CommandRequest table). If the parameters are not valid, the function exits with gNWInvalidParam_c.

Otherwise the profile layer constructs the NLDE data request payload from the provided parameters and transmits it to the recipient through the network layer. If the application has requested a vendor specific transmission by setting the corresponding bit in the txOptions parameter, the ZRC command code and ZRC command ID are ignored and the NLDE Data Request payload consists entirely of the ZRC command payload.

If the data is not vendor specific, the NLDE Data Request payload depends on the ZRC command code, as shown in the following table:

Table 2-21. NLDE Data Request payload depending on the ZRC command code

ZRC Command Code	NLDE Data Request payload (In addition to the ZRC command code)
gZRC_CmdCode_UserCtrlPressed_c	The command ID, interpreted as the RC command code + the ZRC command payload. When receiving a ZRCProfile_CommandRequest with this ZRC command code, the ZRC profile sends over the air the User Control Pressed command frame.
gZRC_CmdCode_UserCtrlReleased_c	The command ID; the rest of the parameters are ignored. The command Id should match the command Id of a previous sent gZRC_CmdCode_UserCtrlPressedAndRepeat_c request. If the ZRC command ID to be released cannot be matched with none of the gZRC_CmdCode_UserCtrlPressedAndRepeat_c commands currently in transmission, the ZRCProfile_CommandRequest function exits with gNWInvalidParam_c.

Table 2-21. NLDE Data Request payload depending on the ZRC command code

gZRC_CmdCode_ DiscoveryRequest_c	All the parameters are ignored. The ZRC profile sends over the air the discovery request command frame and waits the discovery response.
gZRC_CmdCode_UserCtrlPressedAndRepeat_c	The command ID, interpreted as the RC command code + the ZRC command payload. The profile sends first a User Control Pressed command frame with the RC command code and ZRC payload specified in parameters. After this frame is sent, it starts to automatically repeat the RC command ID this time inside a User Control Repeated command frame, at the interval specified by the aplKeyRepeatInterval attribute (gZrcAttr. keyRepeatInterval). Each transmission of the user control repeated command frame is signaled to the application if the Freescale specific attribute receiveKeyRepeatCnf is set to TRUE. Otherwise, the profile repeats the RC command ID without notifying the application layer. The RC command is repeated until the application sends a release command (the commandCode parameter in the function should be set to gZRC_CmdCode_UserCtrlReleased_c). The release command has to specify the same RC command ID to end the repetitions. If the ZRC command ID to be released is not matching any of the IDs of the frames currently being repeated, then the function exits with gNWInvalidParam_c and the profile will continue to repeat the RC command ID.

When the transmission of the ZRC command Discovery Request is completed (whether successful or not), the application is notified through a ZRC Discovery Command Confirm message.

NOTE

Usually, the command Discovery Request is sent to the remote node when the pairing process is successfully ended. On the remote node, the application should keep the receiver open for a while, so that to receive the Discovery Request command frame. When Discovery Request frame is received, the ZRC profile of the remote node will respond by sending automatically a Discovery Response frame which contains the ZRC supported commands. The supported ZRC commands are taken from gaZRCCmdSupportedBitMap bitmap which can be configured by the application at compile time in the ZRCProfileCommands.h file.

When any command transmission except for the ZRC command Discovery Request is completed, the application is notified by the profile through a ZRC command Confirm message. In the case of Discovery Request, the application will get the Confirm message when the Discovery Response frame is received or a timeout has occured. Note that each transmission of the User Control Repeated command frame is notified to the application only if the Freescale specific attribute receiveKeyRepeatCnf is set TRUE.

NOTE

The ZRC profile can simultaneously process a maximum of two transmission requests (requests for sending either vendor specific data or ZRC commands) at a time. If more than two requests are sent at a time, the profile will only process the first two requests and the rest will be denied. On receive side, the ZRC profile can simultaneously handle the reception of a maximum of two RC commands. (e.g. two RC buttons are simultaneously held down).

2.1.16 ZRC Command Indication

The ZRC Command Indication message informs the application of a just arrived ZRC data packet.

2.1.16.1 Message Structure

The ZRC Command Indication message has the following structure:

```
typedef struct zrcProfileCommandInd_tag
{
  uint8_t     deviceId;
  uint8_t     dataLength;
  uint8_t     vendorId[2];
  uint8_t     LQI;
  uint8_t     rxFlags;
  uint8_t     commandAction;
  uint8_t     commandId;
  uint8_t*    pData;
}zrcProfileCommandInd_t;
```

The following table specifies the the fields available in the ZRC Command Indication message.

Table 2-22. ZRC Send Command Indication Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	1 – (gMaxPairTableEntries_c – 1)	The index of the originator's pair table entry
dataLength	uint8_t	-	The length of the command payload
vendorId	uint8_t	-	The originator's vendor Id
LQI	uint8_t	-	The link quality indicator of the received command packet
rxFlags	uint8_t	-	The rxFlags of the NLDE Data Indication message. See the ZigBee RF4CE specification for an exact description
commandCode	uint8_t	gZRC_CmdCode_UserCtrlPressed_c gZRC_CmdCode_UserCtrlRepeated_c gZRC_CmdCode_UserCtrlReleased_c	The ZRC command code
command	uint8_t	-	The ID of the ZRC command (ZRC command code dependent)
pData	uint8_t*	-	The command payload

2.1.16.2 When Generated

The ZRC Send Command Indication is generated by the arrival of a NLDE Data Indication message with a ZRC profile ID (value 0x01).

2.1.16.3 Effect on Receipt

On receipt of the Command Indication message the application is informed of the arrival of a ZRC command sent by the device indicated in the message. If the data is vendor specific (i.e. the vendor specific data bit in the rxFlags field is set), the *commandCode* and *command* fields should be ignored and the ZRC Command payload contains the full NLDE Data Indication payload. If the data is not vendor specific, then the *command* and *pData* parameters have the following significance, depending on the ZRC command code (i.e. depending on the value of the *commandCode* field):

Table 2-23. Significance of the command and pData parameters (Depending on the ZRC Command Code)

ZRC command code	NLDE Data Request payload (in addition to the ZRC command code)
gZRC_CmdCode_UserCtrlPressed_c gZRC_CmdCode_UserCtrlRepeated_c	command contains the RC command code and pData contains the RC command payload (if any)
gZRC_CmdCode_UserCtrlReleased_c	the pData parameter should be ignored

2.1.17 ZRC Command Confirm

The ZRC Command Confirm message informs the application about the completion of a ZRC command transmission process.There is however an exception: the ZRC Discovery Request command generates a ZRC Discovery Command Confirm message when the transmission process is completed.

2.1.17.1 Message Structure

The ZRC Command Confirm message has the following structure:

```
typedef struct zrcProfileCommandCnf_tag
{
  uint8_t     status;
  uint8_t     deviceId;
  uint8_t     commandCode;
  uint8_t     command;
}zrcProfileCommandCnf_t;
```

The following table specifies the fields available in the ZRC Command Confirm message.

Table 2-24. ZRC Send Command Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c or the network layer error code	Indicates either the successful completion of the command transmission process or identifies the error that has occured.
deviceId	uint8_t	1 – (gMaxPairTableEntries_c – 1)	The index of the pair table entry of the command recipient

Table 2-24. ZRC Send Command Confirm Message Structure

commandcode	uint8_t	gZRC_CmdCode_UserCtrlPressed_c gZRC_CmdCode_UserCtrlRepeated_c gZRC_CmdCode_UserCtrlReleased_c	The ZRC command code; will only be used if txOptions does not indicate vendor specific data.
command	uint8_t	-	The command Id; its meaning depends on the ZRC command code.

2.1.17.2 When Generated

The ZRC Command Confirm message is generated by the ZRC Command Tx/Rx sublayer when a previously started command transmission process is complete (i.e. the NLDE Data Confirm message is received).

2.1.17.3 Effect on Receipt

On receipt of the ZRC Command Confirm message, the application layer is notified about the completion of the command transmission process.

2.1.18 ZRC Discovery Command Confirm

The ZRC Command Confirm message notifies the application about the completion of the transmission process of a ZRC Discovery Request command .

2.1.18.1 Message Structure

The ZRC Discovery Command Confirm message has the following structure:

```
typedef struct zrcProfileDiscoveryCmdCnf_tag
{
  uint8_t     status;
  uint8_t     deviceId;
  uint8_t     cmdSupportedBitMap[gCmdsSupportedFieldLength_c];
}zrcProfileDiscoveryCmdCnf_t;
```

The following table specifies the fields available in the ZRC Discovery Command Confirm message message.

Table 2-25. ZRC Send Command Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c or the network layer error code	Indicates either the successful completion of the command transmission process or identifies the error that has occured.
deviceId	uint8_t	1 – (gMaxPairTableEntries_c – 1)	The index of the pair table entry of the command recipient
cmdSupportedBitMap	uint8_t	-	The bitmap containing the supported commands of the remote note.

2.1.18.2 When Generated

The ZRC Command Confirm message is generated by the ZRC Command Tx/Rx sublayer when a previously started transmission process of a ZRC Discovery Request command is complete (i.e. the NLDE Data Confirm message is received).

2.1.18.3 Effect on Receipt

On receipt of the ZRC Command Confirm message, the application layer is notified of the completion of the transmission process of a ZRC Discovery Request command.

2.1.19 ZRCProfile_AbortProcess

The ZRCProfile_AbortProcess function call instructs the ZRC Command Tx/Rx sublayer to abort any ongoing process. This function is included in the RF4CE_ZRCProfile_CommandTxRx library.

2.1.19.1 Function Prototype

ZRCProfile_AbortProcess has the following prototype:

```
uint8_t ZRCProfile_AbortProcess(void);
```

ZRCProfile_AbortProcess has no parameters

The possible return values for the ZRCProfile_AbortProcess API call are shown in the following table.

Table 2-26. ZRCProfile_AbortProcess API Call Return Values

Type	Possible Values	Description
uint8_t	gNWDenied_c gNWNotPermitted_c gNWSuccess_c	All possible return values are fully described in Section 2.1.19.3, “Effect on Receipt”.

2.1.19.2 Functionality

ZRCProfile_AbortProcess allows the application to terminate any ongoing ZRC process (i.e. push-button pairing, both target and originator side, and command transmission).

2.1.19.3 Effect on Receipt

Upon receipt of ZRCProfile_AbortProcess the ZRC Command Tx/Rx sublayer first checks whether a process is currently running. If the ZRC Command Tx/Rx sublayer is idle the function exits with *gNWDenied_c*.

Otherwise an event is sent to the ZRC Command Tx/Rx sublayer task to abort its current process and the function returns *gNWSuccess_c*. The process is aborted the next time the ZRC task is run by the task scheduler.

2.2 ZRC Attributes

The ZRC attributes are declared and initialized in the ZRCProfileGlobals.c file as follows:

```
zrcAttrData_t gZrcAttr ={
                         gDefaultKeyRepeatInterval_c,
                         gDefaultKeyRepeatWaitTime_c,
                         gDefaultExTransferCount_c,
                         gDefaultReceiveKeyRepeatCnf_c
                        };



typedef struct zrcAttrData_tag{
  uint8_t  keyRepeatInterval;
  uint16_t keyRepeatWaitTime;
  uint8_t  keyExTransferCount;
  uint8_t  receiveKeyRepeatCnf;
}zrcAttrData_t;
```

Each attribute has the following meaning:

- keyRepeatInterval - The interval in milliseconds at which user command repeat frames will be transmitted (a key pressed is followed by key repetitions – using gZRC_CmdCode_UserCtrlPressedAndRepeat_c command code).
- keyRepeatWaitTime - The duration that a recipient of a user control repeated command frame waits before terminating a repeated operation.
- keyExTransferCount - The value of the KeyExTransfer-Count parameter passed to the pair request primitive during the push button pairing procedure.
- gDefaultReceiveKeyRepeatCnf_c – It is a Freescale specific attribute and when it is set (TRUE) the profile signals the application (via a ZRC Command Confirm message) that an user control repeated command frame was sent (whether successful or not) over the air.

The ZRC attributes have the following identifiers assigned:

```
#define gKeyRepeatIntervalAttrId_c   0x80
#define gKeyRepeatWaitTimeAttrId_c   0x81
#define gKeyExTransferCountAttrId_c  0x82
#define gReceiveKeyRepeatCnfAttrId_c 0x90  /* This is a Freescale specific attribute Id */
```

To initialize these attributes, configure the following macros from the ZRCProfileGlobals.c file:

```
/*  The interval in milliseconds at which
    user command repeat frames will be transmitted.
    Range: 0 - 100(aplcMaxKeyRepeatInterval) milliseconds */
#define gDefaultKeyRepeatInterval_c 50

/* The duration that a recipient of a user control repeated command frame
   waits before terminating a repeated operation.
   Range: 200 (2*aplcMaxKeyRepeatInterval) - 65535 (0xffff) */
#define gDefaultKeyRepeatWaitTime_c 200
```

```
/* The number of seeds exchange transmissions used to generate a security key.
   Range: 0x03(aplcMinKeyExchangeTransferCount) - 0xff */
#define gDefaultExTransferCount_c   0x24

/* Receive confirm(on Application Layer) when a
   user control repeat command frame is sent; This is a Freescale specific attribute */
#define gDefaultReceiveKeyRepeatCnf_c FALSE
```

The following functions can be used to set/get a specific attribute:

2.2.1 ZRCProfile_GetRequest

The ZRCProfile_GetRequest function gets a specific attribute. This function is included in the RF4CE_ZRCProfile_CommandTxRx library.

2.2.1.1 Function Prototype

ZRCProfile_GetRequest has the following prototype:

```
uint8_t ZRCProfile_GetRequest(
                               uint8_t attrId, /* IN*/
                               uint8_t* pAttrLength, /* OUT */
                               uint8_t* pAttrValue /* OUT*/
                              );
```

The following table specifies the parameters for ZRCProfile_GetRequest.

Table 2-27. ZRCProfile_GetRequest Parameters

Name	Type	Valid Range	Description
attrId	uint8_t	gKeyRepeatIntervalAttrId_c gKeyRepeatWaitTimeAttrId_c gKeyExTransferCountAttrId_c gReceiveKeyRepeatCnfAttrId_c	The attribute identifier
pAttrLength	uint8_t*	-	The returned attribute length.
pAttrValue	uint8_t*	-	The returned attribute value.

The possible return values for the ZRCProfile_GetRequest API call are shown in the following table.

Table 2-28. ZRCProfile_GetRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWUnsupportedAttribute_c gNWSuccess_c	All possible return values are fully described below.

2.2.1.2 Functionality

This function searches the attribute after the attrId parameter. If the attribute is not found, the function returns gNWUnsupportedAttribute_c status. Otherwise, the function exits with gNWSuccess_c status and returns the attribute length and value.

2.2.2 ZRCProfile_SetRequest

The ZRCProfile_SetRequest function sets a specific attribute. This function is included in the RF4CE_ZRCProfile_CommandTxRx library.

2.2.2.1 Function Prototype

ZRCProfile_SetRequest has the following prototype:

```
uint8_t ZRCProfile_SetRequest(
                               uint8_t attrId, /* IN */
                               uint8_t*pAttrValue /* IN */
                              );
```

The following table specifies the parameters for ZRCProfile_SetRequest.

Table 2-29. ZRCProfile_SetRequest Parameters

Name	Type	Valid Range	Description
attrId	uint8_t	gKeyRepeatIntervalAttrId_c gKeyRepeatWaitTimeAttrId_c gKeyExTransferCountAttrId_c gReceiveKeyRepeatCnfAttrId_c	The attribute identifier
pAttrValue	uint8_t*	-	The attribute value.

The possible return values for the ZRCProfile_SetRequest API call are shown in the following table.

Table 2-30. ZRCProfile_SetRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWInvalidParam_c gNWUnsupportedAttribute_c gNWSuccess_c	All possible return values are fully described below.

2.2.2.2 Functionality

This function searches the attribute after the attrId parameter. If the attribute is not found, the function returns the gNWUnsupportedAttribute_c status. Next it will check if the pAttrValue parameter is valid (is within range). If the parameter doesn't have a valid value, it will exit with the gNWInvalidParam_c. Otherwise, it will set the attribute and return the gNWSuccess_c status.