BeeStack Consumer Private Profile

Reference Manual

Document Number: BSCONPPRM
Rev. 1.2
09/2011

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

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

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

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

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

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

Contents

About This Book

This reference manual describes the Freescale BeeStack Consumer Private Profile layer (Private Profile) which allows users to simplify the development of ZigBee BeeStack Consumer applications.

The Private Profile resides on top of the BeeStack Consumer layer and implements functionality designed to automate specific tasks.

Audience

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

Organization

This document contains the following chapters:

Chapter 1 Private Profile Overview - Provides an introduction to the Private Profile.

Chapter 2 Private Profile Software Usage - Provides a description of the Private Profile interfaces.

Revision History

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

Revision History

Doc. Version	Date / Author	Description / Location of Changes
1.2	Sept. 2011, Dev Team	Updates for OTAP.

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
PAN	Personal Area Network
NWK	Network Layer
NLDE	Network Layer Data Entity
NLME	Network Layer Management Entity
SAP	Service Access Point
OTAP	Over The Air Programming
Polling list	Map (8 bit array) in which each bit corresponds to a location in the Pair Table. When a bit in the array is set, a poll request will be sent to device represented by this bit in the Pair Table.

Chapter 1
Private Profile Overview

This chapter provides a brief overview of the Private Profile Layer.

1.1 Private Profile Introduction

To aid in the development of applications based on the BeeStack Consumer protocol stack, Freescale has created a Private Profile which implements functionality useful for automating specific tasks. The Private Profile resides in the protocol stack between the BeeStack Consumer layer and the application layer. The application can still access the network layer directly.

The Private Profile implements the following functions:

- Fragmented transmission — Allows the application to easily transmit more data than can fit in the payload of a NLDE data request. The profile fragments the data into pieces small enough for NLDE data request and transmits them one by one to the recipient. The fragmented transmission functionality on the recipient reassembles the data and presents it to the application as a whole.
- Remote pairing — Allows the application to request that a pairing link be created between two nodes in its pair table (e.g. a remote can request that a Video Player and a TV, both already paired with the remote, pair with each other).
- Polling — Allows the application to periodically poll specific devices from its pair table for any data they might have to send back.
- Over the air menus — Provides the application with the ability to send menu content information to a remote device. The remote device can navigate (browse) and/or display the menu information for the device sending menu information. Intended to provide remote user interface capabilities to remote devices.
- Over the air programming : provides the application with the ability to upgrade its program image over the air. A device that stores updated images, called the server, is able to serve these images to clients that wish to update their image.

The following figure shows the software architecture of an application using the Private Profile.

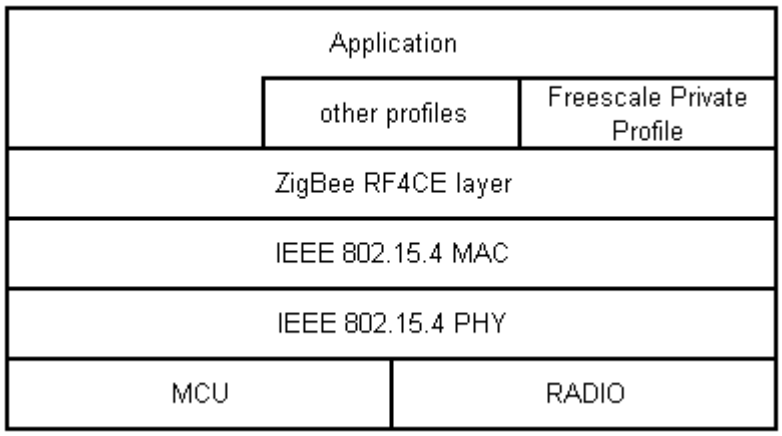

Figure 1-1. Figure 1-1 Private Profile Application Structure

1.2 Private Profile Libraries

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

Table 1-1. Private Profile Libraries

Library	Description
RF4CE_FSLProfile_Framework	Contains the framework to support all the Private Profile functionality. Must be included in any project using the Private Profile. The functionality it provides is always active and doesn't need to be initialized.
RF4CE_FSLProfile_RmtPairOrig	Contains the remote pair originator functionality (the ability to initiate remote pairing of two other nodes)
RF4CE_FSLProfile_RmtPairRecip	Contains the remote pair recipient functionality (the ability to allow the current node to be paired remotely)
RF4CE_FSLProfile_FragTxOrig	Contains the fragmented transmission originator functionality (the ability to transmit fragmented data)
RF4CE_FSLProfile_FragTxRecip	Contains the fragmented transmission recipient functionality (the ability to receive fragmented data)
RF4CE_FSLProfile_PollOrig	Contains the poll originator functionality (the ability to poll other devices)
RF4CE_FSLProfile_PollRecip	Contains the poll recipient functionality (the ability to respond to poll requests from other devices)
RF4CE_FSLProfile_MenuBrowser	Contains the over the air menu browser functionality
RF4CE_FSLProfile_MenuOwner	Contains the over the air menu owner functionality
RF4CE_FSLProfile_MenuDisplayer	Contains the over the air menu displayer functionality
RF4CE_FSLProfile_MenuOwnerLight	Contains a lightweight version of the over the air menu owner functionality (intended to be used in conjunction with the BeeStack Consumer BlackBox application)

Table 1-1. Private Profile Libraries (continued)

Library	Description
RF4CE_FSLProfile_OtapServer	Contains the over the air programming server functionality
RF4CE_FSLProfile_OtapClient	Contains the over the air programming client functionality

Chapter 2
Private Profile Software Usage

The Private Profile performs the following functions:

- Remote device pairing
- Fragmented transmission
- Device polling
- Over the air menus
- Over the air programming

2.1 Private Profile Service Specifications

The Private Profile relies completely on the underlying network layer to perform its tasks. The profile layers of different nodes communicate with each other through the NLDE. Data is sent by issuing the NLDE_DataRequest primitive with vendor specific data (utilizing Freescale's vendor Id and profile Id. Communication is encrypted if the pairing link between the nodes is secured. The NLDE SAP must be configured to redirect messages intended for the profile layer to the profile SAP, so that these don't erroneously reach the application. Refer to the *BeeStack Consumer Private 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. Take care to ensure that the application and the profile never make a simultaneous request to the network layer.

The Private Profile system is shown in Figure 2-1.

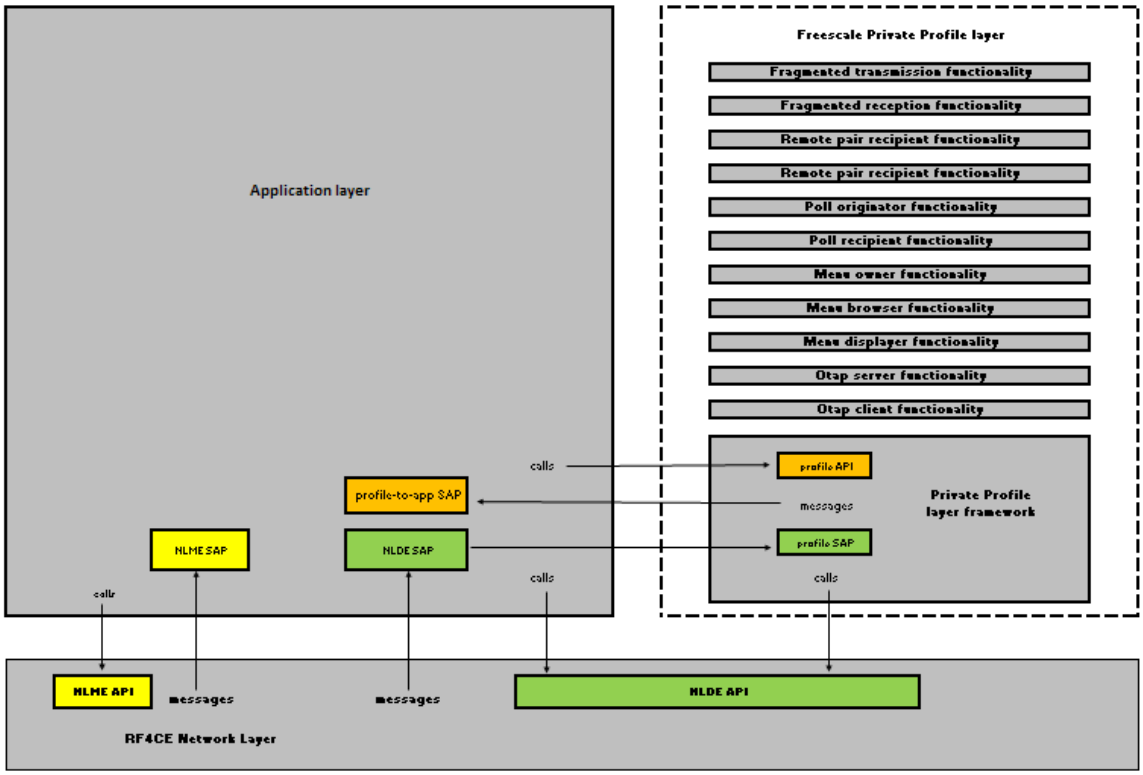

Figure 2-1. Private Profile Layer Components and Interfaces

The Private Profile provides four services accessed through the profile layer API and listed in Table 2-1. These services have the following characteristics:

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

The Private Profile layer SAP provides the interface for confirmation and indication messages.

Table 2-1. Private Profile Services List

Freescale Profile Service	Initialization	Request	Confirm	Indication	Response
Remote Pairing	Section 2.1.3 Section 2.1.4	Section 2.1.5	Section 2.1.6 Section 2.1.9	Section 2.1.7	Section 2.1.9
Fragmented Transmission	Section 2.1.10 Section 2.1.11	Section 2.1.12 Section 2.1.13 Section 2.1.14	Section 2.1.15	Section 2.1.16 Section 2.1.17	
Polling	Section 2.1.18 Section 2.1.19	Section 2.1.20 Section 2.1.21 Section 2.1.24	Section 2.1.22	Section 2.1.23 Section 2.1.25	
Over The Air Menus	Section 2.1.26 Section 2.1.27 Section 2.1.28	Section 2.1.29 Section 2.1.37 Section 2.1.38 Section 2.1.39 Section 2.1.40	Section 2.1.30 Section 2.1.41	Section 2.1.31 Section 2.1.32 Section 2.1.33 Section 2.1.34 Section 2.1.35 Section 2.1.42	Section 2.1.43
Over The Air Programming	Section 2.1.44 Section 2.1.45	Section 2.1.49 Section 2.1.52	Section 2.1.50 Section 2.1.53	Section 2.1.46 Section 2.1.47 Section 2.1.48 Section 2.1.51	

The Private Profile layer reuses the error codes from the network layer. For a detailed list of all the error codes check the *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 FSLProfile_GetSupportedFeatures

FSLProfile_GetSupportedFeatures instructs the profile layer to interrogate another device about the FSL Private Profile features it supports. This function is included in the main framework library, RF4CE_FSLProfile_Framework.

2.1.1.1 Prototype

FSLProfile_GetSupportedFeatures has the following prototype:

```
uint8_t FSLProfile_GetSupportedFeatures(
                            uint8_t deviceId,
                            bool_t bUseSecurity
                  );
```

The following table specifies the parameters for FSLProfile_GetSupportedFeatures.

Table 2-2. FSLProfile_GetSupportedFeatures Parameters

Name	Type	Valid Range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the node to be interrogated
bUseSecurity	bool_t	{FALSE;TRUE}	Whether to send the interrogation command encrypted or not. Only applicable for a secured pairing link.

The possible return values for the FSLProfile_GetSupportedFeatures API Call Return Values API call are shown in the following table.

Table 2-3. FSLProfile_GetSupportedFeatures API Call Return Values

Type	Possible Values	Description
uint8_t	gNWDenied_c gNWSuccess_c	All possible return values are fully described in Section 2.1.1.3, “Effect on Receipt”.

2.1.1.2 Functionality

FSLProfile_GetSupportedFeatures is used to find out what functionality of the Private Profile is supported by another node.

NOTE

No Freescale Profile process performs any feature interrogation on its own, e.g when initiating a fragmented transmission the originator node does not ask the recipient whether it supports fragmented reception. To ensure that the recipient node(s) support(s) the desired features the application must request the interrogation itself.

2.1.1.3 Effect on Receipt

On receipt of FSLProfile_RmtPairRequest, the profile layer first verifies if all the conditions to begin an interrogation process are met.

If the profile layer is busy with another request the function exits with *gNWDenied_c*. Otherwise the function returns *gNWSuccess_c* and the interrogation process begins. The receiver is enabled indefinitely and the value of the receiver’s initial active period is stored in an internal variable. The profile layer will transmit the interrogation command to the target node and wait for the response. The target node will respond automatically if it has the Private Profile. When the interrogation process is complete the application will be notified via a Get Supported Features Confirm message.

2.1.2 Get Supported Features Confirm

The Get Supported Features Confirm message informs the application that the target interrogation process previously started by calling FSLProfile_GetSupportedFeatures has been completed.

2.1.2.1 Message Structure

The Get Supported Features Confirm message has the following structure:

```
typedef struct fslProfileGetSupportedFeaturesCnf_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
  uint8_t                 supportedFeaturesMap[4];
}fslProfileGetSupportedFeaturesCnf_t;
```

The following table specifies the fields available in the Get Supported Features Confirm message.

Table 2-4. Get Supported Features Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWNoResponse_c gNWSuccess_c or the network layer error code	Indicates either the successful completion of the interrogation process or identifies the error that has occured.
deviceId	uint8_t	0 – (gMaxPairTableEntries_c – 1) or 0xFF	The interrogated node's device ID
Supported Features Map	uint8_t[4]	-	The list of features supported by the interrogated node; should be ignored if status is not *gNWSuccess_c*

2.1.2.2 When Generated

The Get Supported Features is generated by the profile layer when a previously started interrogation process has ended.

The interrogation process can end under the following circumstances:

- An interrogation command could not be transmitted over the air – the status field will contain the network layer error code.
- An interrogation response has not been received in a timely fashion – status will be *gNWNoResponse_c*.
- The interrogation process was completed successfully – status will be *gNWSuccess_c*.

2.1.2.3 Effect on Receipt

On receipt of the Get Supported Features message, the application layer is notified about the completion of the interrogation process. If the reported status is *gNWSuccess_c* the supportedMap field will contain the list of features supported by the interrogated node. The supported features map is a bit map with each set bit indicated a supported and initialized feature and each cleared bit indicating a not supported feature. Bit numbering starts with the right-most, least significant bit (which is bit 0). The following table shows the correspondence between the bits and the supported features:

Table 2-5. Bit index - Supported Features Correspondence

Bit Index	Feature
0	Fragmented transmission
1	Fragmented reception
2	Poll originator
3	Poll recipient
4	Remote pair originator
5	Remote pair recipient
6	OTA menu browser
7	OTA menu owner
8	OTA menu displayer
9	OTAP server
10	OTAP client
11 – 31	Reserved

2.1.3 FSLProfile_InitRmtPairOrigProcedure

FSLProfile_InitRmtPairOrigProcedure initializes the remote pair originator functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_RmtPairOrig library.

2.1.3.1 Prototype

FSLProfile_InitRmtPairOrigProcedure has the following prototype:

```
uint8_t FSLProfile_InitRmtPairOrigProcedure(void);
```

FSLProfile_InitRmtPairOrigProcedure has no parameters.

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

Table 2-6. FSLProfile_InitRmtPairOrigProcedure API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNoTimers_c gNWSuccess_c	All possible return values are fully described in Section 2.1.3.3, “Effect on Receipt”.

2.1.3.2 Functionality

FSLProfile_InitRmtPairOrigProcedure is used to enable the remote pair originator functionality in the profile layer, so that the application can initiate remote pairing.

2.1.3.3 Effect on Receipt

On receipt of the FSLProfile_InitRmtPairOrigProcedure function call the profile layer configures itself to be able to handle the remote pair originator procedure (pairing two devices from the pair table). It also tries to allocate the timers needed by the profile layer (if not already allocated). If no timers can be allocated the function call returns *gNWNoTimers_c* and no additional configuration is done. Otherwise the function call returns *gNWSuccess_c* and the remote pair originator functionality is now ready to be used.

2.1.4 FSLProfile_InitRmtPairRecipProcedure

FSLProfile_InitRmtPairRecipProcedure initializes the remote pair recipient functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_RmtPairRecip library.

2.1.4.1 Prototype

FSLProfile_InitRmtPairRecipProcedure has the following prototype:

```
uint8_t FSLProfile_InitRmtPairRecipProcedure(void);
```

FSLProfile_InitRmtPairRecipProcedure has no parameters.

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

Table 2-7. FSLProfile_InitRmtPairRecipProcedure API Call Return Values

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

2.1.4.2 Functionality

FSLProfile_InitRmtPairRecipProcedure is used to enable the remote pair recipient functionality in the profile layer, so that the application can process a remote pairing request.

2.1.4.3 Effect on Receipt

On receipt of the FSLProfile_InitRmtPairRecipProcedure function call the profile layer configures itself to be able to handle the remote pair recipient procedure (being able to handle remote pairing requests). It also tries to allocate the timers needed by the profile layer (if not already allocated). If no timers can be

allocated the function call returns *gNWNoTimers_c* and no additional configuration is done. Otherwise the function call returns *gNWSuccess_c* and the remote pair recipient functionality is now ready to be used.

2.1.5 FSLProfile_RmtPairRequest

FSLProfile_RmtPairRequest instructs the profile layer to attempt to pair together two devices from its pairing table. To use this function, the application must link to the RF4CE_FSLProfile_RmtPairOrig library.

2.1.5.1 Prototype

FSLProfile_RmtPairRequest has the following prototype:

```
uint8_t FSLProfile_RmtPairRequest(
                                  uint16_t          appRecipRspTimeOut,
                                  uint8_t           deviceId1,
                                  appCapabilities_t dev1AppCapabilities,
                                  uint8_t*          pDev1DeviceTypeList,
                                  uint8_t*          pDev1ProfileIdList,
                                  uint8_t           deviceId2,
                                  appCapabilities_t dev2AppCapabilities,
                                  uint8_t*          pDev2DeviceTypeList,
                                  uint8_t*          pDev2ProfileIdList
                                 );
```

The following table specifies the parameters for FSLProfile_RmtPairRequest.

Table 2-8. FSLProfile_RmtPairRequest Parameters

Name	Type	Valid range	Description
appRecipRspTimeOut	uint16_t	1 - 65535	The amount of time (in milliseconds) each device has to respond to the remote pair request.
deviceId1	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the first node
dev1AppCapabilities	appCapabilities_t		The RF4CE application capabilities of the first node
pDev1DeviceTypeList	uint8_t*	-	The list of device types supported by the first node
pDev1ProfileIdList	uint8_t*	-	The list of profiles supported by the first node
deviceId2	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the second node
dev2AppCapabilities	appCapabilities_t		The RF4CE application capabilities of the second node
pDev2DeviceTypeList	uint8_t*	-	The list of device types supported by the second node
pDev2ProfileIdList	uint8_t*	-	The list of profiles supported by the second node

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

Table 2-9. FSLProfile_RmtPairRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWInvalidParam_c gNWNoMemory_c gNWDenied_c gNWSuccess_c	All possible return values are fully described in Section 2.1.5.3, “Effect on Receipt”.

2.1.5.2 Functionality

FSLProfile_RmtPairRequest is used to establish a pairing link between two nodes in the requester’s pairing table (e.g. a remote that is paired with a TV and a DVD can pair the TV and the DVD together). The established pairing link is a standard ZigBee RF4CE pairing link. All its restrictions apply:

- Two controllers cannot be paired together
- At least one device must be a target

2.1.5.3 Effect on Receipt

On receipt of FSLProfile_RmtPairRequest, the profile layer first verifies if all the conditions to begin a remote pairing process are met.

First the remote pair originator functionality must have been initialized (by calling FSLProfile_InitRmtPairOrigProcedure). If the functionality is uninitialized the function exits with *gNWNotPermitted_c*. If the profile layer is busy with another request the function returns *gNWDenied_c*.

The parameters are then checked for validity. If deviceId1 has the same value as deviceId2, either pair table entry is non-existent, both devices are controllers or one of the provided application capabilities are invalid, the function exits with the *gNWInvalidParam_c* error code.

- If no timers could be allocated for the remote pairing process the function returns *gNWNoTimers_c*.
- If no memory buffers could be allocated for the remote pairing process the function returns *gNWNoMemory_c*.
- Otherwise the remote pairing process is initiated.

If both nodes to be paired are security capable a security key is generated. The remote pair initiator generates the entire pair table entries for both devices to be paired, with the exception that, if one of the two devices is a controller, no short address is generated for it (this must be left for the target, as the remote pair originator does not know what short addresses the target has already allocated).

Remote pair request frames are sent, in turn, to each of the devices to be paired. Each remote pair request frame sent to one device to be paired contains the pair table entry of the other device. Each device has the opportunity to reject pairing, and its negative response will terminate the process. If both devices accept pairing, the second device to receive the remote pair request will send a standard RF4CE ping frame to the first device to test the new pairing link, to which the first device will reply with an RF4CE ping response. The second device will only send its affirmative response to the remote pair originator after the arrival of

the ping response (if the second device rejects pairing it will reply immediately). A field in the pair request frame informs the receiving device whether it is the first or the second device to receive the remote pair request frame.

If both devices are targets, remote pair request frames are sent in order (i.e. device 1 will receive the first remote pair request frame and device 2 the second). The ping request payload is completely random.

If one of the two devices to be paired is a controller, that device will be the first to receive the remote pair request frame. The second device will be a target. It will recognize from the pair table entry received with the remote pair request frame that the first device is a controller and will generate a short address for it. The controller's short address will be included in the ping request frame payload (the ping request frame is sent using the controller's IEEE extended address as the destination address). Upon receiving the ping frame, the controller extracts the short address from the payload and uses it for all further communication with the target, including the ping response frame.

The timeout for the first device's response is appRecipRspTimeOut plus a predefined jitter time(60 ms).

The timeout for the second device's response takes into account the fact that it is send after the ping exchange, for which an extra 200 ms are allowed.

When the remote pairing process is complete the originator application will receive a remote pair confirm message through the profile SAP.

The following message chart illustrates a successful remote pairing:

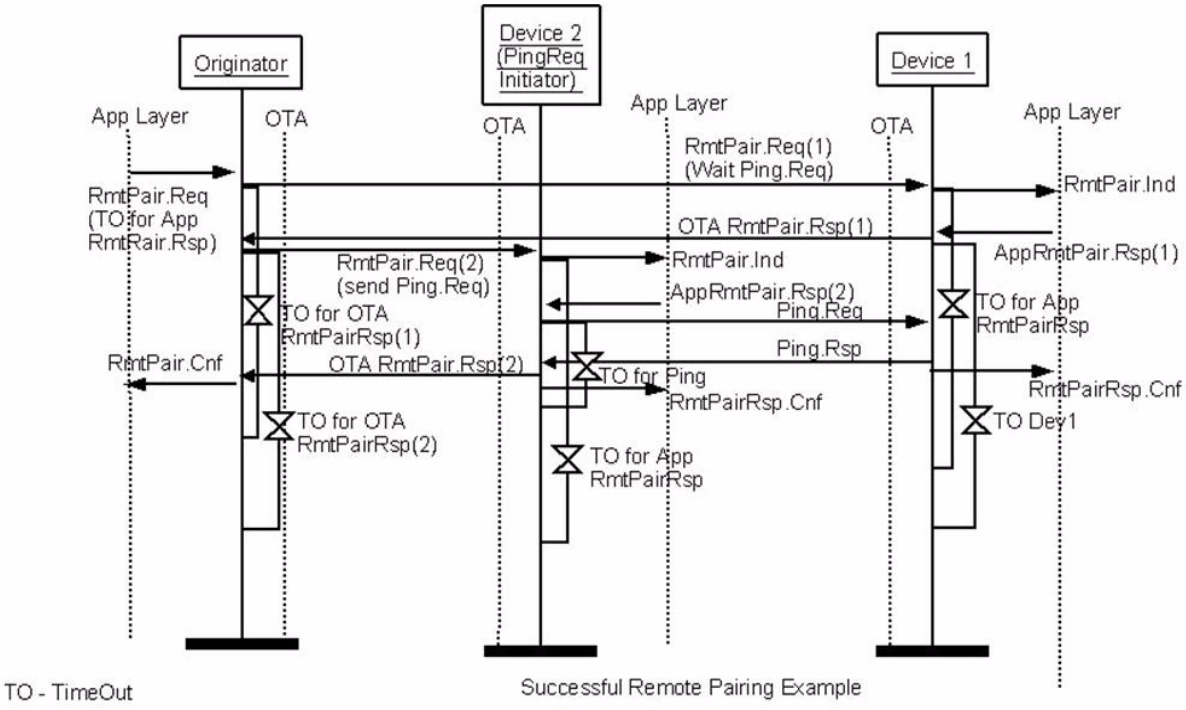

Figure 2-2. Successful Remote Pairing Example

2.1.6 Remote Pair Confirm

The Remote Pair Confirm message informs the application that the remote pairing process previously started by calling FSLProfile_RemotePairRequest has been completed.

2.1.6.1 Message Structure

The remote pair confirm message has the following structure:

```
typedef struct fslProfileRmtPairCnf_tag
{
  uint8_t                 status;
}fslProfileRmtPairCnf_t;
```

The following table specifies the fields available in the Remote Pair Confirm message.

Table 2-10. Remote Pair Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWNoResponse_c gNWSuccess_c gNWNotPermitted_c gNWNoRecipCapacity_c or the network layer error code	Indicates either the successful completion of the remote pairing process or identifies the error that has occured.

2.1.6.2 When Generated

The Remote Pair Confirm message is generated by the profile layer when a previously started remote pairing process has ended.

The remote pairing process can end under the following circumstances:

- A remote pair request frame could not be transmitted over the air – the status field will contain the network error code.
- A remote pair response has not been received in a timely fashion from one of the devices or the pairing link test (i.e. the ping frame exchange) failed – status will be *gNWNoResponse_c*.
- One of the devices has rejected pairing – status will contain the reason for the rejection (either *gNWNotPermitted_c* or *gNWNoRecipCapacity_c*).
- If all information has been exchanged before the expiration of the abort timer interval, the RemotePair process is considered successful and a Remote Pair Confirm message with the status set to *gNWSuccess_c* value is sent to the application.

2.1.6.3 Effect on Receipt

On receipt of the Remote Pair Confirm message, the application layer is notified about the completion of the remote pairing process.

2.1.7 Remote Pair Indication

The remote pair indication message informs the application of a just arrived remote pair request frame and also specifies whether pairing can be accepted (depending on whether any free entries are in the pair table) and the amount of time the application has to respond to the request.

2.1.7.1 Message Structure

The Remote Pair Indication message has the following structure:

```
typedef struct fslProfileRmtPairInd_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
  uint16_t                appRspTimeOut;
  appCapabilities_t       devAppCapabilities;
  uint8_t*                pDeviceTypeList;
  uint8_t*                pProfilesList;
}fslProfileRmtPairInd_t;
```

The following table specifies the the fields available in the Remote Pair Indication message.

Table 2-11. Remote Pair Indication Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c gNWDuplicatePairing_c gNWNoRecipCapacity_c	Specifies whether pairing can be accepted, the device to pair with is already in the pair table or the pair table is full
deviceId	uint8_t	1 – (gMaxPairTableEntries_c – 1)	The position in the pair table where the new entry will reside
appRspTimeOut	uint16_t	1 – 65535	The amount of time (in milliseconds) the application has to respond
devAppCapabilities	appCapabilities_t		The application capabilities of the device to pair with
pDeviceTypeList	uint8_t*		The list of supported device types of the device to pair with
pProfilesList	uint8_t*		The list of supported profiles of the device to pair with

2.1.7.2 When Generated

The Remote Pair Indication message is generated by the arrival of a remote pair request frame at the profile layer. The profile layer first checks if the device to pair with is already in the pairing table. If it is, status will be *gNWDuplicatePairing_c* and the deviceId will point to the existing pairing table entry. If the pairing table is full, status will be *gNWNoRecipCapacity_c* and deviceId will be 0xFF. Otherwise status will be *gNWSuccess_c* and the deviceID will indicate the free position in the pair table where the new entry will be placed.

The appRspTimeOut value and the application capabilities, supported device types and supported profiles of the device to pair with are taken from the remote pair request frame.

2.1.7.3 Effect on Receipt

On receipt of the Remote Pair Indication message the application is informed of a request to pair with another device. The application has a limited amount of time (appRspTimeOut) to respond to the request by calling FSLProfile_RemotePairResponse. If it does not respond within that time frame the remote pairing process times out (no remote pair response is sent to the originator) and the remote pair originator receives a Remote Pair Confirm message with a status of *gNWNoResponse_c*.

2.1.8 FSLProfile_RmtPairResponse

The FSLProfile_RemotePairResponse function call instructs the profile layer to respond to a previously arrived remote pair request frame. To use this function, the application must link to the RF4CE_FSLProfile_RmtPairRecip library.

2.1.8.1 Prototype

FSLProfile_RmtPairResponse has the following prototype:

```
uint8_t FSLProfile_RmtPairResponse(
                    uint8_t status
                  );
```

The following table specifies the parameters for the FSLProfile_RmtPairResponse application service.

Table 2-12. FSLProfile_RmtPairResponse Application Service Parameters

Name	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c gNWNotPermitted_c gNWNoRecipCapacity_c	Specifies whether the application accepts pairing or not.

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

Table 2-13. FSLProfile_RmtPairResponse API Call Return Values

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

2.1.8.2 Functionality

FSLProfile_RemotePairResponse is used to respond to a previously received Remote Pair Indication. The application informs the profile layer whether it accepts the pairing or not.

2.1.8.3 Effect on Receipt

Upon receipt of FSLProfile_RemotePairResponse the profile layer first checks whether the remote pair response can be sent. If the remote pair recipient functionality has not been initialized or no Remote Pair Indication was previously received the function exits with *gNWNotPermitted_c*.

Then parameter validity is verified. If status isn't either *gNWSuccess_c* or *gNWNotPermitted_c* or *gNWNoRecipCapacity_c* the function will exit with *gNWInvalidParameter_c*. Otherwise, the profile layer prepares to respond to the remote pair request frame.

If this is the first device receiving the remote pair request frame, as indicated in the remote pair request frame, the profile layer sends back the remote pair response with the status provided by the application. If the application has rejected pairing the remote pairing process is complete.

If the application has accepted pairing the profile layer starts to wait for an RF4CE ping request frame from the second device. The total amount of time the profile layer will wait for the ping request frame is appRspTimeout + 160 milliseconds. If no ping request frame arrives in this time period the ping exchange and the entire remote pairing process considered to have failed.

If the ping request arrives, a ping response is sent back. If the device is a controller the short address is extracted from the ping request frame and used in all further communications with the second device (including the ping response). The remote pairing is completed when the MAC layer acknowledgement for the ping response arrives.

If this is the second device receiving the remote pair request frame and the application has accepted the pairing, the profile layer will send an RF4CE ping request frame to the first device and wait for the ping response. The total amount of time the profile layer will wait the ping response is 100 milliseconds. If the first device is a controller, as indicated by the pair table entry received with the remote pair request frame, a short address will be generated for it and it will be included in the ping request frame payload. After the ping response is received the profile layer will send the remote pair response to the remote pair originator. If the application has rejected pairing, then no ping request will be sent and the remote pair response will be sent to the originator directly. The other device will time out waiting for the ping request.

2.1.9 Remote Pair Response Confirm

The Remote Pair Response Confirm message informs the application of the final status of a remote pairing process initiated by another device.

2.1.9.1 Message Structure

The Remote Pair Response Confirm message has the following Message Structure:

```
typedef struct fslProfileRmtPairRspCnf_tag
{
```

```
  uint8_t                 status;
  uint8_t                 deviceId;
}fslProfileRmtPairRspCnf_t;
```

The following table specifies the fields available in the Remote Pair Response Confirm message.

Table 2-14. Remote Pair Response Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c gNWNoResponse_c or the error code returned by the network layer	Specifies the final result of a remote pairing process
deviceId	uint8_t	0 – (gMaxPairTableEntries_c – 1)	Position in the table where information about the remotely paired node can be found, in case the Remote Pair process has completed successfully. If remote pairing was not successful, this value should be ignored.

2.1.9.2 When Generated

The Remote Pair Response Confirm message is generated by the profile layer in response to an FSLProfile_RemotePairResponse function call and signals the end of the remote pairing process.

The remote pairing process on a recipient can end under the following circumstances:

- Remote Pairing is completed successfully; the new pairing link is successfully tested with a ping frame exchange – status is *gNWSuccess_c*
- Pairing is rejected, either by the application or because the pair table is full. The response is sent back to the originator immediately and the process ends when the network layer confirms the delivery of the response. The status field in the confirm message informs the application about the response delivery. It will be *gNWSuccess_c* if the response was successfully sent over the air, otherwise it will contain the error reported by the network layer.
- The pairing link test fails, i.e. the ping frame exchange fails – status is *gNWNoResponse_c* or the network layer error code.

2.1.9.3 Effect on Receipt

On receipt of the Remote Pair Response Confirm message, the application layer of a remote pair recipient is notified about the completion of a Remote Pairing Process. If remote pairing was accepted and the process was successfully completed (i.e. status is *gNWSuccess_c*) the provided device Id points to the new entry in the pair table. Otherwise the pair table is un-modified.

2.1.10 FSLProfile_InitFragTxOrigProcedure

FSLProfile_InitFragTxOrigProcedure initializes the fragmented transmission functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_FragTx library.

2.1.10.1 Prototype

FSLProfile_InitFragTxOrigProcedure has the following prototype:

```
uint8_t FSLProfile_InitFragTxOrigProcedure(void);
```

FSLProfile_InitFragTxOrigProcedure has no parameters.

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

Table 2-15. FSLProfile_InitFragTxOrigProcedure API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNoTimers_c gNWSuccess_c	All possible return values are fully described in Section 2.1.10.3, “Effect on Receipt”

2.1.10.2 Functionality

FSLProfile_InitFragTxOrigProcedure is used to enable fragmented transmission functionality in the profile layer, so that the application can transmit fragmented data.

2.1.10.3 Effect on Receipt

On receipt of the FSLProfile_InitFragTxOrigProcedure function call the profile layer configures itself to be able to handle transmission of fragmented data. It also tries to allocate the timers needed by the profile layer (if they are not already allocated). If no timers can be allocated the function returns *gNWNoTimers_c* and no additional configuration is done. Otherwise the function returns *gNWSuccess_c* and the application can initiate the transmission of fragmented data.

2.1.11 FSLProfile_InitFragTxRecipProcedure

FSLProfile_InitFragTxRecipProcedure initializes the fragmented reception functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_FragTxRecip library.

2.1.11.1 Prototype

FSLProfile_InitFragTxRecipProcedure has the following prototype:

```
uint8_t FSLProfile_InitFragTxRecipProcedure(void);
```

FSLProfile_InitFragTxRecipProcedure has no parameters.

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

Table 2-16. FSLProfile_InitFragTxRecipProcedure API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNoTimers_c gNWSuccess_c	All possible return values are fully described in Section 2.1.11.3, “Effect on Receipt”

2.1.11.2 Functionality

FSLProfile_InitFragTxRecipProcedure is used to enable the fragmented reception functionality in the profile layer, so that the application can receive fragmented data.

Table 2-17. FSLProfile_DisplayCompleteIndToBrowserRequest API Call Return Values

2.1.11.3 Effect on Receipt

On receipt of the FSLProfile_InitFragTxRecipProcedure function call the profile layer configures itself to be able to reception of fragmented data. It also tries to allocate the timers needed by the profile layer (if not already allocated). If no timers can be allocated the function returns *gNWNoTimers_c* and no additional configuration is done. Otherwise the function returns *gNWSuccess_c* and the node is ready to receive fragmented data.

2.1.12 FSLProfile_SetFragTxRxBufferStateRequest

The FSLProfile_SetFragTxRxBufferStateRequest service makes a request for the profile layer to change the fragmented transmission buffer state. To use this function, the application must link to either the RF4CE_FSLProfile_FragTxOrig library or the RF4CE_FSLProfile_FragTxRecip library.

The fragmented transmission buffer is an application buffer used by the Freescale profile layer in fragmented transmissions. Its maximum length is specified by the application and must be less than 64 Kbytes. This buffer can be in one of the following states:

- *gFragTxRxBufferFree_c* - The buffer is free/available
- *gFragTxRxBufferBusyApp_c* - The buffer is reserved for the application. No fragmented reception can begin when the buffer is in this state (fragmented transmission requests are discarded by the profile layer). When the profile layer receives the last fragment of a transmission, it sets the buffer state to *gFragTxRxBufferBusyApp_c* and notifies the application that a fragmented reception has been completed and the received data is in the buffer. Also, before requesting a fragmented transmission the application should set the buffer state to *gFragTxRxBufferBufferBusyApp_c*, write the data to be transmitted in the buffer, and then pass the control to the profile layer.
- *gFragTxRxBufferBusyProfile_c* – The buffer is reserved by the profile (when profile is busy either sending or receiving fragmented data, the buffer state is set to *gFragTxRxBufferBusyProfile_c*). The application should neither read from, nor write to the buffer when it is in this state.

2.1.12.1 Prototype

The FSLProfile_SetFragTxRxBufferStateRequest API function call has the following prototype:

```
uint8_t FSLProfile_SetFragTxRxBufferStateRequest(
                        profileFragTxRxBufferState_t newBufferState
                    );
```

The following table specifies the parameters for the FSLProfile_SetFragTxRxBufferStateRequest service.

Table 2-18. FSLProfile_SetFragTxRxBufferStateRequest Service Parameters

Name	Type	Valid values	Description
newBufferState	profileFragTxRxBufferState_t	gFragTxRxBufferFree_c gFragTxRxBufferBusyApp_c	Specifies the new fragmented transmission buffer state.

The possible return values for the FSLProfile_SetFragTxRxBufferStateRequest function calll are shown in the following table.

Table 2-19. FSLProfile_SetFragTxRxBufferStateRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWInvalidParam_c gNWNotPermitted_c gNWDenied_c gNWSuccess_c	All error codes are discussed in Section 2.1.12.3, “Effect on Receipt”

2.1.12.2 Functionality

The FSLProfile_SetFragTxRxBufferStateRequest function changes the state of the fragmented transmission buffer to the requested value.

2.1.12.3 Effect on Receipt

On receipt of FSLProfile_SetFragTxRxBufferStateRequest service, the profile layer will check if the new state requested can be set by the application and if the current state can be changed. After that the previous fragmented transmission buffer state will be replaced with the requested one.

2.1.12.4 Returns

- *gNWInvalidParam_c* – when the desired buffer state is neither *gFragTxRxBufferBusyProfile_c* nor *gFragTxRxBufferBusyApp_c* nor *gFragTxRxBufferFree_c*
- *gNWNotPermitted_c* – when the desired state can’t be set by the application (the application can’t set the buffer state to *gFragTxRxBufferBusyProfile_c*)
- *gNWDenied_c* – when the application is not allowed to change the buffer state because it is in use by the profile layer (the application can’t change the buffer state from *gFragTxRxBufferBusyProfile_c*)

- *gNWSuccess_c* – when the fragmented transmission buffer state was changed to the requested value

2.1.13 FSLProfile_GetFragTxRxBufferStateRequest

FSLProfile_GetFragTxRxBufferStateRequest makes a request to the profile layer to return the current fragmented transmission buffer state. To use this function, the application must link to either the RF4CE_FSLProfile_FragTxOrig library or the RF4CE_FSLProfile_FragTxRecip library.

2.1.13.1 Prototype

The FSLProfile_GetFragTxRxBufferStateRequest function has the following prototype:

```
profileFragTxRxBufferState_t FSLProfile_GetFragTxRxBufferStateRequest(void);
```

FSLProfile_GetFragTxRxBufferStateRequest does not pass any parameters to profile layer.

The possible return values for the FSLProfile_GetFragTxRxBufferStateRequest Request service API call are shown in the following table.

Table 2-20. FSLProfile_GetFragTxRxBufferStateRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gFragTxRxBufferFree_c gFragTxRxBufferBusyApp_c gFragTxRxBufferBusyProfile_c	gFragTxRxBufferFree_c - The buffer is free gFragTxRxBufferBusyApp_c – The buffer is reserved for the application gFragTxRxBufferBusyProfile_c – The buffer is reserved for the profile

2.1.13.2 Functionality

FSLProfile_GetFragTxRxBufferStateRequest is used to find out the current state of the fragmented transmission buffer.

2.1.13.3 Effect on Receipt

On receipt of FSLProfile_GetFragTxRxBufferStateRequest, the profile layer will return the current fragmented transmission buffer state.

2.1.14 FSLProfile_FragTxRequest

The FSLProfile_FragTxRequest API function allows an application to request the transmission of data to another node from the pair table. To use this function, the application must link to the RF4CE_FSLProfile_FragTxOrig library.

2.1.14.1 Prototype

The FSLProfile_FragTxRequest function has the following prototype:

```
uint8_t FSLProfile_FragTxRequest(
                  uint8_t  deviceId,
                  uint16_t dataLen,
                  uint8_t* pData,
                  bool_t            bUseSecurity
                );
```

The following table specifies the parameters for FSLProfile_FragTxRequest.

Table 2-21. FSLProfile_FragTxRequest Request Service Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c – 1)	Pair table entry index of the data recipient
data	uint8_t*	-	A pointer to the data to transmit
length	uint16_t	1 – 65535	The amount of data to transmit
bUseSecurity	bool_t		Whether to use secured transmission or not. This only applies for secured pairing links.

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

Table 2-22. FSLProfile_FragTxRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWDenied_c gNWNotPermitted_c gNWInvalidParam_c gNWSuccess_c	All possible return values are fully described in Section 2.1.14.3, “Effect on Receipt”

2.1.14.2 Functionality

FSLProfile_FragTxRequest is used to start a fragmented transmission to a node in the pairing table.

2.1.14.3 Effect on Receipt

On receipt of FSLProfile_FragTxRequest, the profile layer verifies if all the conditions to begin a fragmented transmission are met.

If the fragmented transmission functionality has not been initialized the function exits with *gNWNotPermitted_c*. If the profile layer is busy with another request the function exits with *gNWDenied_c*. If the profile layer is not busy with another application request but the buffer state is *gFragTxRxBufferBusyProfile_c* (i.e. a fragmented reception is in progress) the function exits with *gNWNotPermitted_c*.

The profile layer then checks the validity of the parameters. If the data length is 0 or the data pointer is null, the function exits with *gNWInvalidParameter_c* error code.

The profile layer then checks its internal state machine and status variables, to determine if it can accept the request or not. Once the validity of all parameters has been verified, the profile layer accepts the request for processing, and the *gNWSuccess_c* value is returned. The fragmented transmission buffer state is changed to *gFragTxRxBufferBusyProfile_c*.

When the fragmented transmission is complete, the profile layer will inform the application about the status of the operation using a fragmented transmission confirm message. Power saving is disabled during fragmented transmission.

2.1.15 Fragmented Transmission Confirm

The fragmented transmission confirm message informs the application about the completion of a previously started fragmented transmission process.

2.1.15.1 Message Structure

The fragmented confirm message has the following structure:

```
typedef struct fslProfileFragCnf_tag
{
  uint8_t                 status;
  uint16_t                fragRxMaxAcceptedLen;
}fslProfileFragCnf_t;
```

The following table specifies the fields available in the Fragmented Transmission Confirm message.

Table 2-23. Fragmented Transmission Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWNoTimers_c gNWTimeOut_c gNWNoMemory_c gNWNoResponse_c gNWNoRecipCapacity_c gNWSuccess_c or any error returned by the network layer	Specifies how the fragmented transmission was completed
fragRxMaxAcceptedLen	uint16_t	1 – 65535	The maximum amount of data the recipient can receive in a single transmission. This field should be ignored if the status is neither *gNWSuccess_c* not *gNWNoRecipCapacity_c*

2.1.15.2 When Generated

The Fragmented Transmission Confirm message is generated by the profile layer entity at the conclusion of a fragmented transmission request.

When the profile layer is processing a fragmented transmission request, it performs the following:

- Sends a fragmented transmission start packet to the destination node (the packet contains the transfer length) and waits for the the response (which contains the maximum amount of data the receiver can accept and a status of *gNWSuccess_c* or *gNWNoRecipCapacity_c* depending on whether the requested transmission length exceeds the receivers capacity or not).
- If timeout occurs, the profile layer sends the fragmented transmission confirm message with a status of *gNWNoResponse_c*.
- If the response status is *gNWNoRecipCapacity_c*, the profile layer sends the fragmented transmission confirm message with a status of *gNWNoRecipCapacity_c*.
- If the response status is *gNWSuccess_c*, the profile layer begins sending the fragments
- The fragments are sent using single-channel, unacknowledged transmission. Periodically, status updates about what fragments have been received are requested from the recipient. If a timeout occurs while waiting for any of the status updates, the fragmented transmission process is aborted and the profile layer sends the fragmented transmission confirm message with a status of *gNWNoResponse_c*. Otherwise, any fragments that have been reported as not received are retransmitted, this time using multiple-channel, acknowledged transmission.
- When the recipient reports that all fragments have arrived, the profile layer sends the fragmented transmission confirm message with a status of *gNWSuccess_c*.
- If at any point the profile layer fails to transmit any frame over the air, the fragmented transmission process is aborted and the fragmented transmission confirm message contains the network layer error code.

2.1.15.3 Effect on Receipt

On receipt of the Fragmented Transmission Confirm message, the application layer of the initiating device is notified of the result of the fragmented transmission. The state of the fragmented transmission buffer is set to *gFragTxRxBufferFree_c* by the profile layer. Power saving is restored to the state it was before the fragmented transmission process.

2.1.16 Fragmented Transmission Start Indication

The Fragmented Transmission Start Indication message informs the application layer about the start of a fragmented reception initiated by one of the nodes in its Pair Table. The incoming data will be saved in the fragmented transmission buffer.

2.1.16.1 Message Structure

The Fragmented Transmission Start Indication message has the following structure:

```
typedef struct fslProfileFragStartInd_tag
{
  uint8_t                 deviceId;
  uint16_t                fragDataLen;
}fslProfileFragStartInd_t;
```

The following table specifies the fields available in the Fragmented Transmission Start Indication message.

Table 2-24. Fragmented Transmission Start Indication Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c – 1)	The pair table entry index of the fragmented transmission originator
fragDataLen	uin16_t	1 – 65535	The amount of data that will arrive

2.1.16.2 When Generated

The Fragmented Transmission Start Indication message is generated by the profile layer upon the arrival of a fragmented transmission start packet from a paired device but only if the transmission can proceed. If the data cannot be received (e.g. because the buffer is not in the *gFragTxRxBufferFree_c* state, or the buffer is too small to hold all the data to be received) no indication message is generated.

2.1.16.3 Effect on Receipt

On receipt of the Fragmented Transmission Start Indication message, the application layer is notified about the start of a fragmented reception, initiated by one of the nodes in its pairing table.

A fragmented transmission start response frame has been sent back to the originator. The state of the fragmented transmission buffer has been changed to *gFragTxRxBufferBusyProfile_c*.

2.1.17 Fragmented Transmission Indication

The Fragmented Transmission Indication message informs the application layer on the receiving end of the transfer about the end of a fragmented transmission initiated by one of the nodes in its pairing table.

2.1.17.1 Message Structure

The fragmented transmission indication message has the following structure:

```
typedef struct fslProfileFragInd_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
  bool_t                  bFragSecured;
```

```
  uint16_t                fragDataLen;
}fslProfileFragInd_t;
```

The following table specifies the fields available in the Fragmented Transmission Indication message.

Table 2-25. Fragmented Transmission Indication Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWTimeOut_c gNWNoMemory_c gNWFailed_c gNWSuccess_c	Indicates how the fragmented transmission has ended
deviceId	uint8_t	0 – (gMaxPairTableEntries_c – 1)	Specifies the position in the Pair Table of the originator node
bFragSecured	uint8_t	{TRUE, FALSE}	Indicates whether the transmission was secured or not
fragDataLen	uin16_t	1 – 65535	The total amount of data received.

2.1.17.2 When Generated

The Fragmented Transmission Indication message is generated by the profile layer when the last data fragment from one of the nodes in its pairing table is received.

2.1.17.3 Effect on Receipt

On receipt of the Fragmented Transmission Indication message, the application layer is notified about the end of a fragmented reception, initiated by one of the nodes in its pairing table. The Fragmented Transmission Indication message will always be received after a Fragmented Transmission Start Indication message. The received data is placed in the fragmented transmission buffer, starting with position 0. At the moment when this message is received by application, the state of the buffer is already set, by the profile layer, to *gFragTxRxBufferBusyApp_c*.

2.1.18 FSLProfile_InitPollOrigProcedure

FSLProfile_InitPollOrigProcedure initializes poll originator functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_PollOrig library.

2.1.18.1 Prototype

FSLProfile_InitPollOrigProcedure has the following prototype:

```
uint8_t FSLProfile_InitPollOrigProcedure(void);
```

FSLProfile_InitPollOrigProcedure has no parameters.

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

Table 2-26. FSLProfile_InitPollOrigProcedure API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNoTimers_c gNWSuccess_c	All possible return values are fully described in Section 2.1.18.3, “Effect on Receipt”

2.1.18.2 Functionality

FSLProfile_InitPollOrigProcedure is used to enable the polling originator functionality in the profile layer, so that the application can initiate the polling operation.

2.1.18.3 Effect on Receipt

On receipt of the FSLProfile_InitPollOrigProcedure function call the profile layer configures itself to be able to handle the polling originator procedure. It also tries to allocate the timers needed by the profile layer (if not already allocated). If no timers can be allocated the function call returns *gNWNoTimers_c* and no additional configuration is done. Otherwise the function returns *gNWSuccess_c* and the node can poll other devices.

2.1.19 FSLProfile_InitPollRecipProcedure

FSLProfile_InitPollRecipProcedure initializes the poll recipient functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_PollRecip library.

2.1.19.1 Prototype

FSLProfile_InitPollRecipProcedure has the following prototype:

```
uint8_t FSLProfile_InitPollRecipProcedure(void);
```

FSLProfile_InitPollRecipProcedure has no parameters.

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

Table 2-27. FSLProfile_InitPollRecipProcedure API Call Return Values

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

2.1.19.2 Functionality

FSLProfile_InitPollRecipProcedure is used to enable the polling recipient functionality in the profile layer, so that the application can respond to polling requests.

2.1.19.3 Effect on Receipt

On receipt of the FSLProfile_InitPollRecipProcedure function call the profile layer configures itself to be able to handle the polling recipient procedure. It also tries to allocate the timers needed by the profile layer (if not already allocated). If no timers can be allocated the function call returns *gNWNoTimers_c* and no additional configuration is done. Otherwise the function call returns *gNWSuccess_c* and the poll recipient functionality is now ready to be used (the node can respond to poll request frames from other devices).

2.1.20 FSLProfile_PollConfigRequest

FSLProfile_PollConfigRequest instructs the profile layer to configure the polling period and the amount of time the receiver is enabled during polling. A node starts with polling un-configured. Polling must be configured first before any polling is started. To use this function, the application must link to the RF4CE_FSLProfile_PollOrig library.

2.1.20.1 Prototype

FSLProfile_PollConfigRequest has the following prototype:

```
uint8_t FSLProfile_PollConfigRequest(
                    uint32_t pollInterval,
                    uint16_t rxOnInterval
                  );
```

The following table specifies the parameters for FSLProfile_PollConfigRequest.

Table 2-28. FSLProfile_PollConfigRequest Parameters

Name	Type	Dir	Valid range	Description
pollInterval	uint32_t	IN	100 – 262000(miliseconds)	The polling period.
rxOnInterval	uint16_t	IN	0 - pollInterval (milliseconds)	The amount of time the receiver will be enabled in order to wait for incoming data

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

Table 2-29. FSLProfile_PollConfigRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNoTimers_c gNWInvalidParam_c gNWSuccess_c	All possible return values are fully described in Section 2.1.20.3, “Effect on Receipt”

2.1.20.2 Functionality

FSLProfile_PollConfigRequest configures the poll parameters. It neither stops an existing polling process nor does it start a new one. Polling must be stopped and restarted for the new parameters to take effect.

2.1.20.3 Effect on Receipt

On receipt of FSLProfile_PollConfigRequest, the profile layer verifies if all the conditions to set the polling period and receiver enabled interval are met.

If pollInterval is out of bounds, or rxOnInterval is larger than pollInterval the function returns *gNWInvalidParameter_c*. If no timer could be allocated for the polling process, the function exits with *gNWNoTimers_c*.

Once the validity of the parameters has been verified, the profile layer saves the poll parameters in its internal variables and exits with *gNWSuccess_c*.

2.1.21 FSLProfile_PollRequest

FSLProfile_PollRequest starts or stops the polling process for a specified device Id. This function cannot be called without previously having configured the poll parameters. To use this function, the application must link to the RF4CE_FSLProfile_PollOrig library.

2.1.21.1 Prototype

FSLProfile_PollRequest has the following prototype:

```
uint8_t FSLProfile_PollRequest(
                    uint8_t deviceId,
                    bool_t bPollEnable
                  );
```

The following table specifies the parameters for FSLProfile_PollRequest.

Table 2-30. FSLProfile_PollRequest Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c – 1) or 0xFF	Pair table index of the entry of the device to poll. If deviceId is 0xFF polling is enabled or disabled for all devices in the pairing table
bPollEnable	bool_t	{TRUE,FALSE}	Enables / disables polling the given device.

The possible return values of the FSLProfile_PollRequest call are shown in the following table.

Table 2-31. FSLProfile_PollRequest Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDeviceIdNotPaired_c gNWInvalidParam_c gNWSuccess_c	All possible return values are fully described in Section 2.1.21.3, “Effect on Receipt”

2.1.21.2 Functionality

FSLProfile_PollRequest is used to start or stop polling either a specific device or all devices in the pair table.

2.1.21.3 Effect on Receipt

On receipt of FSLProfile_PollRequest, the profile layer first verifies if all the conditions to handle the request are met.

If the poll originator functionality has not been enabled the function exits with *gNWNotPermitted_c*.

If there is no information in the Pair Table at the position indicated by deviceId parameter, then the function exits with *gNWDeviceIdNotPaired_c*. If the deviceId parameter exceeds the boundaries of the Pair Table, but is different from 0xFF, the function returns the *gNWInvalidParameter_c* value. If the poll interval and the receiver enabled interval were not already set, the function returns the *gNWInvalidParameter_c* value.

After the validity of the parameters has been verified, the profile layer accepts the FSLProfile_PollRequest for processing, and the *gNWSuccess_c* value is returned.

If the application has requested that polling be enabled (bPollEnable is *TRUE*) then the device with the given device ID is added to the internal list of devices to be polled (the poll list). If the device ID is *0xFF* then all devices in the pair table are added to the poll list. An interval timer is started with pollInterval (configured with FSLProfile_PollConfigRequest) as its period. Its callback function triggers a polling attempt which polls all devices in the poll list in sequence. (Even if the poll list was not empty prior to the call to FSLProfile_PollRequest, the timer is restarted).

If the application has requested that polling be disabled (bPollEnable is *FALSE*) then the device with the given device ID is removed from the poll list. If deviceId is *0xFF* then the entire poll list is emptied. If the

poll list is empty (either because deviceId was given as *0xFF* or because the given device was the last is the poll list) the polling timer is stopped.

A polling attempt consists of sending a poll request frame to all devices in the poll list one at a time and then waiting for their responses. If the response indicates that data is not available the profile layer will move to the next device in the list. If the response indicates that data is available the profile layer keeps the receiver enabled for a period of time equal to rxOnInterval (configured with FSLProfile_PollConfigRequest) before moving on to the next device. The initial receiver active period is stored in an internal variable and restored at the end of a polling attempt.

A special case occurs when rxOnInterval is configured to 0. In this case, a special poll event message is sent to the application for each poll response frame received. A field in the poll event message indicates whether data is available or not.

2.1.22 Poll Confirm

The FSL Profile Poll Confirm message informs the application that polling has stopped without the application requesting it or that an error has occured during poling.

2.1.22.1 Message Structure

The poll confirm message has the following structure:

```
typedef struct fslProfilePollCnf_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
} fslProfilePollCnf_t;
```

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

Table 2-32. Poll Confirm Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Which device the profile layer was currently attempting to poll
status	uint8_t	gNWAborted_c gNWNoMemory_c	The reason why polling has stopped or the error that has occured

2.1.22.2 When Generated

The Poll Confirm message is generated by the profile layer when polling has stopped for some reason or when an error has occured during polling.

2.1.22.3 Effect on Receipt

The poll confirm message can arrive in two situations:

1. The application has requested the BeeStack Consumer layer to un-pair with all devices in the polling list. Whenever the profile layer detects that the node is no longer paired with a device in the polling list that device is removed from the polling list. If the polling list is empty the polling timer is stopped and a poll confirm message is sent to the application with a status of *gNWAborted_c* and a deviceId of 0xFF.
2. No message buffers could be allocated for the poll request frame. The application receives a poll confirm message with a status of *gNWNoMemory_c* and the pair table entry index of the device the profile layer just failed to poll. Polling continues with the next device in the list.

2.1.23 Poll Indication

The Poll Indication message informs the application layer about the arrival of a poll request from one of the nodes in the Pair Table.

2.1.23.1 Message Structure

The Poll Indication message has the following structure:

```
typedef struct fslProfilePollInd_tag
{
  uint8_t                 deviceId;
  uint8_t                 rxOnInterval;
} fslProfilePollInd_t;
```

The following table specifies the fields available in the Poll Indication message.

Table 2-33. Poll Indication Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c – 1)	The device that sent the poll request
rxOnInterval	uint8_t	-	The amount of time the poll originator will keep its receiver enabled

2.1.23.2 When Generated

The poll indication message is generated whenever a poll request frame arrives from a device for which the application has indicated that data is available, using FSLProfile_PollDataAvailable. Also a poll response frame is sent back to the poll originator, informing it that data is available and causing it to keep its receiver enabled for a predetermined period of time (configured by the originator application using FSLProfile_PollConfigRequest). If data is not available, no indication message is generated and the poll response frame informs the originator that data is not available, causing it to move on to the next device in its poll list.

2.1.23.3 Effect on Receipt

On receipt of the Poll Indication message, the application is informed that it has a window of opportunity to transmit data to the device conducting the poll, as the device will keep its receiver enabled the period of time indicated by rxOnInterval. The data available status is not modified, the application will continue to receive poll indications whenever a poll request arives.

2.1.24 FSLProfile_PollDataAvailable

FSLProfile_PollDataAvailable service is used by the application layer to inform the network layer that it has data to send to a device already in the pairing table. To use this function, the application must link to the RF4CE_FSLProfile_PollRecip library.

2.1.24.1 Prototype

FSLProfile_PollDataAvailable has the following prototype:

```
uint8_t FSLProfile_PollDataAvailable(
                    uint8_t deviceId,
                    bool_t bDataAvailable
                  );
```

The following table specifies the parameters for FSLProfile_PollDataAvailable.

Table 2-34. FSLProfile_PollDataAvailable Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c – 1) or 0xFF	the position in the Pair Table of the node which needs to be informed about the presence of application data designated to it.. If deviceId is 0xFF, all nodes in the pair table are affected.
bDataAvailable	bool_t	{TRUE, FALSE}	whether data is available or not

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

Table 2-35. FSLProfile_PollDataAvailable Call Return Values

Type	Possible Values	Description
uint8_t	gNWDeviceIdNotPaired_c gNWInvalidParam_c gNWSuccess_c	All possible return values are fully described in Section 2.1.24.3, “Effect on Receipt”.

2.1.24.2 Functionality

FSLProfile_PollDataAvailable is used to configure the way the network layer responds to incoming poll request frames.

2.1.24.3 Effect on Receipt

On receipt of FSLProfile_PollDataAvailable, the profile layer first checks parameter validity.

If the deviceId parameter exceeds the boundaries of the Pair Table, the function returns the *gNWInvalidParameter_c* value. If there is no information in the Pair Table at the position indicated by deviceId parameter, then the function exits with the *gNWDeviceIdNotPaired_c* value.

Otherwise, the internal data available map is modified according to the function call parameters.

If bDataAvailable is *TRUE* the given device is added to the data available map, otherwise it is removed from the map. If deviceId is *0xFF* then either all devices in the pair table are added to the map (if bDataAvailable is *TRUE*) or the entire map is cleared. The function returns *gNWSuccess_c*.

Whenever a poll request frame is received by the profile layer, if the originator is in the data available map, the profile layer sends an automatic response informing the originator that data is available. Also, a poll indication message is sent to the application. If the originator is not in the data available map the automatic response informs the originator that data is not available and no poll indication message is sent to the application.

The data available map is not modified in any way by incoming poll requests. The data available map can only be modified by the application by calling FSLProfile_PollDataAvailable.

2.1.25 Poll Event

The Poll Event message informs the application of the arrival of a poll response when rxOnInterval is set to 0.

2.1.25.1 Message Structure

The Poll Event message has the following structure:

```
typedef struct fslProfilePollEvent_tag
{
  uint8_t                 deviceId;
  bool_t                  bDataAvailable;
} fslProfilePollEvent_t;
```

The following table specifies the fields available in the Poll Event message.

Table 2-36. Poll Event Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c – 1)	The device that responded to the poll request
bDataAvailable	bool_t	{TRUE, FALSE}	Indicates whether data is available

2.1.25.2 When Generated

The poll event message is generated whenever rxOnInterval is set to 0 and the profile layer receives a poll response frame.

2.1.25.3 Effect on Receipt

On receipt of the Poll Event message, the application is informed whether the responding device has data available for it.

2.1.26 FSLProfile_InitMenuBrowserProcedure

FSLProfile_InitMenuBrowserProcedure initializes menu browsing functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_MenuBrowser library.

2.1.26.1 Prototype

FSLProfile_InitMenuBrowserProcedure has the following prototype:

```
uint8_t FSLProfile_InitMenuBrowserProcedure(void);
```

FSLProfile_InitMenuBrowserProcedure has no parameters.

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

Table 2-37. FSLProfile_InitMenuBrowserProcedure API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNoTimers_c gNWSuccess_c	All possible return values are fully described in Section 2.1.26.3, “Effect On Receipt”.

2.1.26.2 Functionality

FSLProfile_InitMenuBrowserProcedure is used to enable the menu browsing functionality in the profile layer.

2.1.26.3 Effect On Receipt

On receipt of the FSLProfile_InitMenuBrowserProcedure function call the profile layer configures itself to be able to transmit menu browsing requests over the air. It also tries to allocate the timers needed by the profile layer (if not already allocated). If no timers can be allocated the function call returns *gNWNoTimers_c* and no additional configuration is done. Otherwise the function call returns *gNWSuccess_c* and the menu browsing functionality is ready to be used.

2.1.27 FSLProfile_InitMenuOwnerLightProcedure

FSLProfile_InitMenuOwnerLighProcedure initializes the light menu owner functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_MenuOwnerLight library.

2.1.27.1 Prototype

FSLProfile_InitMenuOwnerLightProcedure has the following prototype:

```
uint8_t FSLProfile_InitMenuOwnerLightProcedure(void);
```

FSLProfile_InitMenuOwnerLightProcedure has no parameters.

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

Table 2-38. FSLProfile_InitMenuOwnerLightProcedure API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNoTimers_c gNWSuccess_c	All possible return values are fully described in Section 2.1.27.3, “Effect on receipt”

2.1.27.2 Functionality

FSLProfile_InitMenuOwnerLightProcedure is used to enable the light menu owner functionality in the profile layer, allowing the application to receive menu browsing requests messages from the menu browser and to send menu displaying requests to the menu displayer.

2.1.27.3 Effect on receipt

On receipt of the FSLProfile_InitMenuOwnerLightProcedure function call the profile layer configures itself to be able to pass menu browsing request frames received over the air to the application and to send menu displaying requests from the application in turn. It also tries to allocate the timers needed by the profile layer (if not already allocated). If no timers can be allocated the function call returns *gNWNoTimers_c* and no additional configuration is done. Otherwise the function call returns *gNWSuccess_c* and the light menu owner functionality is ready to be used.

2.1.28 FSLProfile_InitMenuDisplayerProcedure

FSLProfile_InitMenuDisplayerProcedure initializes the menu displaying functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_MenuDisplayer library.

2.1.28.1 Prototype

FSLProfile_InitMenuDisplayerProcedure has the following prototype:

```
uint8_t FSLProfile_InitMenuDisplayerProcedure(uint16_t waitMenuEntryTimeout);
```

The following table specifies the parameters for FSLProfile_InitMenuDisplayerProcedure.

Table 2-39. FSLProfile_InitMenuDisplayerProcedure Parameters

Name	Type	Valid range	Description
waitMenuEntry Timeout	uint16_t	0 - 65535	The maximum amount of time in milliseconds to wait for the next menu entry when receiving a menu window

The possible return values for the FSLProfile_InitMenuDisplayerProcedure API call are shown in the following table:

Table 2-40. FSLProfile_InitMenuDisplayerProcedure API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNoTimers_c gNWSuccess_c	All possible return values are fully described in Section 2.1.28.3, “Effect on receipt”.

2.1.28.2 Functionality

FSLProfile_InitMenuDisplayerProcedure is used to enable the menu displaying functionality in the profile layer.

2.1.28.3 Effect on receipt

On receipt of the FSLProfile_InitMenuDisplayerProcedure function call the profile layer configures itself to be able to handle incoming menu display request frames and sets the maximum amount of time to wait between two menu entry frames. It also tries to allocate the timers needed by the profile layer (if not already allocated). If no timers can be allocated the function call returns *gNWNoTimers_c* and no additional configuration is done. Otherwise the function call returns *gNWSuccess_c* and the menu displayer functionality is then ready to be used.

2.1.29 FSLProfile_BrowseMenuRequest

FSLProfile_BrowseMenuRequest instructs the profile layer to transmit a menu browse command frame to a menu owner. To use this function, the application must link to the RF4CE_FSLProfile_MenuBrowser library.

2.1.29.1 Prototype

FSLProfile_BrowseMenuRequest has the following prototype:

```
uint8_t FSLProfile_BrowseMenuRequest(
        uint8_t deviceId,
        menuBrowseDirection_t direction,
        bool_t bUseSecurity
);
```

The following table specifies the parameters for FSLProfile_BrowseMenuRequest Parameters.

Table 2-41. FSLProfile_BrowseMenuRequest Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the menu owner
direction	menuBrowse Direction_t	menuBrowseUp_c menuBrowseDown_c menuBrowseLeft_c menuBrowseRight_c menuBrowseOk_c menuBrowseExit_c menuBrowseRefresh_c menuBrowseMax_c	The menu navigation direction
bUseSecurity	bool_t	{FALSE, TRUE}	Whether to use secured transmission or not

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

Table 2-42. FSLProfile_BrowseMenuRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDenied_c gNWInvalidParam_c gNWDeviceIdNotPaired_c gNWSuccess_c	All possible return values are fully described in Section 2.1.29.3, “Effect on Receipt”.

2.1.29.2 Functionality

FSLProfile_BrowseMenuRequest is used to transmit a menu navigation command from a menu browser to a menu owner, usually in response to a user pressing a menu navigation key.

2.1.29.3 Effect on Receipt

On receipt of FSLProfile_BrowseMenuRequest, the profile layer verifies if all the conditions to transmit a menu browse command are met.

If the menu browsing functionality has not been initialized the function exits with *gNWNotPermitted_c*. If the profile layer is busy with another request the function exits with *gNWDenied_c*.

The profile layer then checks the validity of the parameters. If the menu owner's deviceId is outside the pair table the function returns *gNWInvalidParam_c*. If there is no information in the pair table at the position indicated by deviceId parameter, then the function exits with *gNWDeviceIdNotPaired_c*. If the direction is invalid, the function exits with *gNWInvalidParam_c*. If a secured transmission has been requested but the pairing link is not secured the function exits with *gNWInvalidParam_c*.

Otherwise the request is accepted for processing and the function returns *gNWSuccess_c*.

The profile layer now constructs a menu browse command frame and sends it to the menu owner using acknowledged transmission. Once the menu owner's MAC layer replies with an ack the application will be notified via a menu browse confirm message.

2.1.30 Menu Browse Confirm

The Menu Browse Confirm message informs the application about the transmission status of a menu browse command frame.

2.1.30.1 Message Structure

The menu browse confirm message has the following structure:

```
typedef struct fslProfileMenuBrowseCnf_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
}fslProfileMenuBrowseCnf_t;
```

The following table specifies the fields available in the Menu Browse Confirm message.

Table 2-43. Menu Browse Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c or the network layer error code	Indicates either the successful transmission of the menu browse command frame or identifies the error that has occured.
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Identifies the menu owner to which the menu browse command was transmitted.

2.1.30.2 When Generated

The Menu Browse Confirm message is generated by the profile layer when the previously requested transmission of a menu browse request command frame has been completed.

2.1.30.3 Effect on receipt

When the Menu Browse Confirm message arrives the application is notified of the completion status of the menu browse command frame transmission. The status field is copied from the NLDE Data Confirm message.

2.1.31 Menu Browse Complete Indication

The Menu Browse Complete Indication message informs the application about the result of its menu browsing request.

2.1.31.1 Message Structure

The menu Browse Complete Indication message has the following structure:

```
typedef struct fslProfileMenuBrowseCompleteInd_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
  uint8_t                 userString[gSizeOfUserString_c];
}fslProfileMenuBrowseCompleteInd_t;
```

The following table specifies the fields available in the Menu Browse Complete Indication message.

Table 2-44. Menu Browse Complete Indication Message Structure

Field	Type	Possible Values	Description
status	uint8_t	-	The status reported by the menu owner
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Identifies the menu owner to which the menu browsing command was transmitted.
userString	uint8_t[gSizeOfUserString_c]	gNWSuccess_c or the network layer error code	The menu displayer's user string.

2.1.31.2 When Generated

The Menu Browse Complete Indication message is generated by the profile layer when the menu owner informs it about the result of a previous menu browsing request.

2.1.31.3 Effect on receipt

When the Menu Browse Complete Indication message arrives the application is notified about how its menu browsing request was handled by the menu owner and the menu displayer. The status field will contain the status reported by the menu owner. Typically the status will be either *gNWSuccess_c*, if the menu browsing command was successfully executed, or *gNWNoResponse_c* if the menu display frames could not be sent to the displayer. However the menu owner is free to send any value here.

2.1.32 Display Menu Header Indication

The Display Menu Header Indication message informs the application of a menu displayer about the arrival of display menu header request from a menu owner.

2.1.32.1 Message Structure

The Display Menu Header indication message has the following structure:

```
typedef struct fslProfileDisplayMenuHeaderInd_tag
{
  uint8_t                 deviceId;
  uint8_t                 idxSelectedEntry;
  uint8_t                 nrMenuItemsInWindow;
  uint16_t                nrMenuItemsInMenu;
  uint16_t                firstEntryNumber;
  uint8_t                 contentType;
  uint8_t                 menuTextLength;
  uint8_t*                pMenuText;
}fslProfileDisplayMenuHeaderInd_t;
```

The following table specifies the fields available in the Display Menu Header Indication message.

Table 2-45. Display Menu Header Indication Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Identifies the menu owner which has sent the menu header.
idxSelected Entry	uint8_t	-	Identifies the menu entry that is selected
nrMenuItem sInWindow	uint8_t	-	The number of menu items in a window
nrMenuItem sInMenu	uint16_t	-	The total number of items in the menu
firstEntryNu mber	uint16_t	-	The number of the first entry
contentType	uint8_t	-	The type of content in the menu
menuTextlen gth	uint8_t	-	The length of the text describing the menu
pMenuText	uint8_t*	-	Some text describing the menu

2.1.32.2 When Generated

The Display Menu Header Indication message is generated by the profile layer when it receives a display menu header request frame.

2.1.32.3 Effect on receipt

When the Display Menu Header Indication message arrives the application is notified about the arrival of a display menu header request. The profile layer prepares for the reception of a number of display menu entry request frames equal to nrMenuItemsInWindow, an entire menu window. The receiver is enabled indefinitely and the initial value of the receiver's active period is stored in an internal variable. Each menu entry frame has a finite amount of time to arrive, equal to the value of the *waitMenuEntryTimeout* set by the call to FSLProfile_InitMenuDisplayerProcedure, otherwise the process times out. When the menu window reception process has been completed (either because all menu entries have been received or because the process has timed out) the application will be notified via a Display Menu Complete Indication message.

2.1.33 Display Menu Entry Indication

The Display Menu Entry Indication message informs the application of the arrival of display menu entry request from a menu owner.

2.1.33.1 Message Structure

The Display Menu Entry indication message has the following structure:

```
typedef struct fslProfileDisplayMenuEntryInd_tag
{
  uint8_t                 deviceId;
  uint8_t                 entryIndex;
  uint8_t                 entryType;
  uint8_t                 contentType;
  uint8_t                 entryValueLength;
  uint8_t                 entryTextLength;
  uint8_t*                pEntryValue;
  uint8_t*                pEntryText;
}fslProfileDisplayMenuEntryInd_t;
```

The following table specifies the fields available in the Display Menu Header Indication message.

Table 2-46. Display Menu Entry Indication Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Identifies the menu owner which has sent the menu entry.
entryIndex	uint8_t	-	Where in the menu window this entry should reside
entryType	uint8_t	-	The type of the menu entry
contentType	uint8_t	-	The type of content in the menu entry
entryValueLengh	uint8_t	-	The length of the entry value
entryTextLength	uint8_t	-	The length of text in the menu entry

Table 2-46. Display Menu Entry Indication Message Structure (continued)

pEntryValue	uint8_t*	-	The menu entry value
pEntryText	uint8_t*	-	The menu entry text

2.1.33.2 When Generated

The Display Menu Entry Indication message is generated by the profile layer when it receives a display menu entry request frame. The display menu entry request frame can arrive either as part of the transmission of a complete menu window (a menu header and all the menu entries in window) or on its own (e.g. when a single menu entry is updated).

2.1.33.3 Effect on Receipt

On receipt of the Display Menu Entry Indication message the application has been provided with the necessary information to display the menu entry described by the indication message.

2.1.34 Display Menu Complete Indication

The Display Menu Complete Indication message informs the application of the reception of complete menu window (one menu header and multiple menu entries).

2.1.34.1 Message Structure

The Display Menu Entry indication message has the following structure:

```
typedef struct fslProfileDisplayMenuCompleteInd_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
}fslProfileDisplayMenuCompleteInd_t;
```

The following table specifies the fields available in the Display Menu Header Indication message.

Table 2-47. Display Menu Complete Indication Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c gNWNotPermitted_c gNWNoResponse_c or the network layer error code	The status of the menu window reception process
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Identifies the menu owner which has sent the menu window.

2.1.34.2 When Generated

The Display Menu Complete Indication message is generated by the profile layer when the process to receive a menu window has been completed.

2.1.34.3 Effect on Receipt

When the Display Menu Header Indication message arrives the application is notified that a menu window has been received. The receiver's active period is restored to the initial state.

The possible values of the status field have the following significance:

- *gNWSuccess_c* – all display menu entry request frames have been received
- *gNWNoResponse_c* – the display menu entry request frames have not arrived in a timely fashion or have not been received in order
- *gNWNotPermitted_c* – the menu header specified a menu window greater than what the current device can display (i.e. greater than *gMaxMenuEntriesToDisplay*)

2.1.35 Display Menu Message Indication

The Display Menu Message Indication message informs the application of the reception of a display menu message request frame.

2.1.35.1 Message Structure

The Display Menu Message indication message has the following structure:

```
typedef struct fslProfileDisplayMenuMessageInd_tag
{
  uint8_t                 deviceId;
  menuMessageType_t       messageType;
  uint8_t                 messageLength;
  uint8_t*                pMessage;
}fslProfileDisplayMenuMessageInd_t;
```

The following table specifies the fields available in the Display Menu Header Indication message.

Table 2-48. Display Menu Message Indication Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Identifies the menu owner which has sent the menu message.
messageType	menuMessageType_t	-	The message type
messageLength	uint8_t	-	The message length
pMessage	uint8_t*	-	The message

2.1.35.2 When Generated

The Display Menu Message Indication message is generated by the profile layer when it receives a display menu message request frame.

2.1.35.3 Effect on Receipt

On receipt of the Display Menu Message Indication message the application has been provided with the type and content of the message and the device Id of the message originator.

2.1.36 Display Menu Exit Indication

The Display Menu Exit Indication message informs the application of the reception of a display menu exit request frame.

2.1.36.1 Message Structure

The Display Menu Message indication message has the following structure:

```
typedef struct fslProfileDisplayMenuExitInd_tag
{
  uint8_t                 deviceId;
}fslProfileDisplayMenuExitInd_t;
```

The following table specifies the fields available in the Display Menu Exit Indication message.

Table 2-49. Display Menu Exit Indication Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Identifies the menu owner which has sent the menu message.

2.1.36.2 When Generated

The Display Menu Exit Indication message is generated by the profile layer when it receives a display menu exit request frame.

2.1.36.3 Effect on Receipt

On receipt of the Display Menu Exit Indication message the application has been informed that the menu on the owner has become inactive.

2.1.37 FSLProfile_DisplayMenuHeaderRequest

FSLProfile_BrowseMenuRequest instructs the profile layer of a menu owner with a light menu-owner library to transmit a display menu header request frame to a menu displayer. To use this function, the application must link to the RF4CE_FSLProfile_MenuOwnerLight library.

2.1.37.1 Prototype

FSLProfile_DisplayMenuHeaderRequest has the following prototype:

```
uint8_t FSLProfile_DisplayMenuEntryRequest( uint8_t   deviceId,
                                            bool_t    bUseSecurity,
                                            uint8_t   entryIndex,
                                            uint8_t   entryType,
                                            uint8_t   contentType,
                                            uint8_t   entryValueLength,
                                            uint8_t   entryTextLength,
                                            uint8_t*  pEntryValue,
                                            uint8_t*  pEntryText);
```

The following table specifies the parameters for FSLProfile_DisplayMenuHeaderRequest.

Table 2-50. FSLProfile_DisplayMenuHeaderRequest Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the menu displayer
bUseSecurity	bool_t	{FALSE, TRUE}	Whether to use secured transmission or not
idxSelectedEntry	uint8_t	-	Identifies the menu entry that is selected
nrMenuItemsInWindow	uint8_t	-	The number of menu items in a window
nrMenuItemsInMenu	uint16_t	-	The total number of items in the menu
firstEntryNumber	uint16_t	-	The number of the first entry
contentType	uint8_t	-	The type of content in the menu
menuTextlength	uint8_t	-	The length of the text describing the menu
pMenuText	uint8_t*	-	Some text describing the menu

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

Table 2-51. FSLProfile_DisplayMenuHeaderRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDenied_c gNWInvalidParam_c gNWDeviceIdNotPaired_c gNWNoMemory_c gNWSuccess_c	All possible return values are fully described in Section 2.1.37.3, “Effect on Receipt”

2.1.37.2 Functionality

FSLProfile_DisplayMenuHeaderRequest is used to transmit a display menu header frame to a menu displayer. This is the initial request in the process of sending a full menu window.

2.1.37.3 Effect on Receipt

On receipt of FSLProfile_DisplayMenuHeaderRequest, the profile layer verifies if all the conditions to transmit a display menu header frame are met.

If the light menu owner functionality has not been initialized the function exits with *gNWNotPermitted_c*. If the profile layer is busy with another request the function exits with *gNWDenied_c*. If the menu displayer’s device Id is outside the bounds of the pair table the function exits with *gNWInvalidParams*_c. If the menu displayer’s device Id points to an empty pair table entry index the function exits with *gNWDeviceIdNotPaired*_c. If a secured transmission is requested but the pairing link is not secured the function exits with *gNWInvalidParams_c*. If no memory could be allocated to construct the display menu header frame payload the function exits with *gNWNoMemory_c*.

Otherwise the request is accepted for processing and the function returns *gNWSuccess_c*.

The profile layer now constructs a display menu header frame and sends it to the menu displayer using acknowledged transmission. Once the menu owner’s MAC layer replies with an ack the application will be notified via a display menu confirm message. The receiver is enabled indefinitely and the value of the receiver’s initial active period is stored in an internal variable.

2.1.38 FSLProfile_DisplayMenuEntryRequest

FSLProfile_DisplayMenuEntryRequest instructs the profile layer of a menu owner with a light menu-owner library to transmit a display menu header request frame to a menu displayer. To use this function, the application must link to the RF4CE_FSLProfile_MenuOwnerLight library.

2.1.38.1 Prototype

FSLProfile_DisplayMenuEntryRequest has the following prototype:

```
uint8_t FSLProfile_DisplayMenuEntryRequest( uint8_t   deviceId,
                                            bool_t    bUseSecurity,
                                            uint8_t   entryIndex,
                                            uint8_t   entryType,
                                            uint8_t   contentType,
                                            uint8_t   entryValueLength,
                                            uint8_t   entryTextLength,
                                            uint8_t*  pEntryValue,
                                            uint8_t*  pEntryText);
```

The following table specifies the parameters for FSLProfile_DisplayMenuEntryRequest.

Table 2-52. FSLProfile_DisplayMenuEntryRequest Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the menu displayer
bUseSecurity	bool_t	{FALSE, TRUE}	Whether to use secured transmission or not
entryIndex	uint8_t	-	Where in the menu window this entry should reside
entryType	uint8_t	-	The type of the menu entry
contentType	uint18_t	-	The type of content in the menu entry
entryValueLength	uint18_t	-	The length of the entry value
entryTextLength	uint8_t	-	The length of text in the menu entry
pEntryValue	uint8_t*	-	The menu entry value
pEntryText	uint8_t*	-	The menu entry text

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

Table 2-53. FSLProfile_DisplayMenuEntryRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDenied_c gNWInvalidParam_c gNWDeviceIdNotPaired_c gNWSuccess_c	All possible return values are fully described in Section 2.1.38.3, “Effect on Receipt”

2.1.38.2 Functionality

FSLProfile_DisplayMenuEntryRequest is used to transmit a display menu header frame to a menu displayer, either during the process of sending a full menu window or when a single menu entry needs to be updated.

2.1.38.3 Effect on Receipt

On receipt of FSLProfile_DisplayMenuEntryRequest, the profile layer verifies if all the conditions to transmit a display menu entry frame are met.

If the light menu owner functionality has not been initialized the function exits with *gNWNotPermitted_c*. If the profile layer is busy with another request the function exits with *gNWDenied_c*. If the menu displayer's device Id is outside the bounds of the pair table the function exits with *gNWInvalidParams*_c. If the menu displayer's device Id points to an empty pair table entry index the function exits with *gNWDeviceIdNotPaired*_c. If a secured transmission is requested but the pairing link is not secured the function exits with *gNWInvalidParams_c*. If no memory could be allocated to construct the display menu entry frame payload the function exits with *gNWNoMemory_c*.

Otherwise the request is accepted for processing and the function returns *gNWSuccess_c*.

The profile layer now constructs a display menu message frame and sends it to the menu displayer using acknowledged transmission. Once the menu owner's MAC layer replies with an ack the application will be notified via a display menu confirm message. The receiver is enabled indefinitely and the value of the receiver's initial active period is stored in an internal variable.

2.1.39 FSLProfile_DisplayMenuMessageRequest

FSLProfile_DisplayMenuMessageRequest instructs the profile layer of a menu owner with a light menu-owner library to transmit a display menu message request frame to a menu displayer. To use this function, the application must link to the RF4CE_FSLProfile_MenuOwnerLight library.

2.1.39.1 Prototype

FSLProfile_DisplayMenuMessageRequest has the following prototype:

```
uint8_t FSLProfile_DisplayMenuMessageRequest(uint8_t           deviceId,
                                             bool_t            bUseSecurity,
                                             menuMessageType_t messageType,
                                             uint8_t           messageLength,
                                             uint8_t*          pMessage);
```

The following table specifies the parameters for FSLProfile_DisplayMenuMessageRequest.

Table 2-54. FSLProfile_DisplayMenuMessageRequest Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the menu displayer

Table 2-54. FSLProfile_DisplayMenuMessageRequest Parameters (continued)

bUseSecurity	bool_t	{FALSE, TRUE}	Whether to use secured transmission or not
messageType	menuMessageType_t	-	The type of the message
messageLength	uint8_t	-	The length of the message
pMessage	uint8_t*	-	The message to send

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

Table 2-55. FSLProfile_DisplayMenuEntryRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDenied_c gNWInvalidParam_c gNWDeviceIdNotPaired_c gNWSuccess_c	All possible return values are fully described in Section 2.1.39.3, “Effect on Receipt”.

2.1.39.2 Functionality

FSLProfile_DisplayMenuMessageRequest is used to tramsmit a display menu message frame to a menu displayer.

2.1.39.3 Effect on Receipt

On receipt of FSLProfile_DisplayMenuMessageRequest, the profile layer verifies if all the conditions to transmit a display menu message frame are met.

If the light menu owner functionality has not been initialized the function exits with *gNWNotPermitted_c*. If the profile layer is busy with another request the function exits with *gNWDenied_c*. If the menu displayer’s device Id is outside the bounds of the pair table the function exits with *gNWInvalidParams*_c. If the menu displayer’s device Id points to an empty pair table entry index the function exits with *gNWDeviceIdNotPaired*_c. If a secured transmission is requested but the pairing link is not secured the function exits with *gNWInvalidParams_c*. If no memory could be allocated to construct the display menu message frame payload the function exits with *gNWNoMemory_c*.

Otherwise the request is accepted for processing and the function returns *gNWSuccess_c*.

The profile layer now constructs a display menu message frame and sends it to the menu displayer using acknowledged transmission. Once the menu owner’s MAC layer replies with an ack the application will be notified via a display menu confirm message. The receiver is enabled indefinitely and the value of the receiver’s initial active period is stored in an internal variable.

2.1.40 FSLProfile_DisplayMenuExitRequest

FSLProfile_DisplayMenuExitRequest instructs the profile layer of a menu owner with a light menu-owner library to inform the menu displayer that the menu has become inactive. To use this function, the application must link to the RF4CE_FSLProfile_MenuOwnerLight library.

2.1.40.1 Prototype

FSLProfile_DisplayMenuExitRequest has the following prototype:

```
uint8_t FSLProfile_DisplayMenuExitRequest(uint8_t deviceId,
                                          bool_t  bUseSecurity);
```

The following table specifies the parameters for FSLProfile_DisplayMenuExitRequest.

Table 2-56. FSLProfile_DisplayMenuExitRequest Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the menu displayer
bUseSecurity	bool_t	{FALSE, TRUE}	Whether to use secured transmission or not

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

Table 2-57. FSLProfile_DisplayMenuExitRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDenied_c gNWInvalidParam_c gNWDeviceIdNotPaired_c gNWSuccess_c	All possible return values are fully described in Section 2.1.40.3, “Effect on Receipt”.

2.1.40.2 Functionality

FSLProfile_DisplayMenuExitRequest is used to inform the menu displayer that the menu on the owner has become idle.

2.1.40.3 Effect on Receipt

On receipt of FSLProfile_DisplayMenuExitRequest, the profile layer verifies if all the conditions to transmit a display menu exit frame are met.

If the light menu owner functionality has not been initialized the function exits with *gNWNotPermitted_c*. If the profile layer is busy with another request the function exits with *gNWDenied_c*. If the menu displayer’s device Id is outside the bounds of the pair table the function exits with *gNWInvalidParams*_c. If the menu displayer’s device Id points to an empty pair table entry index the function exits with *gNWDeviceIdNotPaired*_c. If a secured transmission is requested but the pairing link is not secured the

function exits with *gNWInvalidParams_c*. If no memory could be allocated to construct the menu browse complete frame payload the function exits with *gNWNoMemory_c*.

Otherwise the request is accepted for processing and the function returns *gNWSuccess_c*.

The profile layer now constructs a display menu exit frame and sends it to the menu displayer using acknowledged transmission. Once the menu displayer's MAC layer replies with an ack the application will be notified via a display menu confirm message. The receiver is enabled indefinitely and the value of the receiver's initial active period is stored in an internal variable. The original value for the active period will be re-instated once the display confirm message is received.

2.1.41 Display Menu Confirm

The Display Menu Confirm message informs the application about the transmission status of a display menu frame (menu header, menu entry or menu message).

2.1.41.1 Message Structure

The Display Menu Confirm message has the following structure:

```
typedef struct fslProfileDisplayMenuCnf_tag
{
  uint8_t                 status;
  uint8_t                 deviceId;
}fslProfileDisplayMenuCnf_t;
```

The following table specifies the fields available in the Display Menu Confirm message.

Table 2-58. Display Menu Confirm Message Structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c or the network layer error code	Indicates either the successful transmission of the display menu frame or identifies the error that has occured.
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Identifies the menu displayer to which the display menu frame was transmitted.

2.1.41.2 When Generated

The Display Menu Confirm message is generated by the profile layer when the previously requested transmission of a display menu frame has been completed. This can be either a display menu header frame or a display menu entry frame or a display menu message frame

2.1.41.3 Effect on Receipt

When the Display Menu Confirm message arrives the application is notified of the completion status of a request to transmit a display menu frame. The status field is copied from the NLDE Data Confirm message. The profile layer can accept new requests and receiver's active period is restored to the value it had before transmission began.

2.1.42 Menu Browse Indication

The Menu Browse Indication message informs the application of a light menu owner about the arrival of a menu browse request frame from a menu browser.

2.1.42.1 Message Structure

The Menu Browse Indication message has the following structure:

```
typedef struct fslProfileMenuBrowseInd_tag
{
  uint8_t                 deviceId;
  menuBrowseDirection_t   direction;
  bool_t                  bUseSecurity;
}fslProfileMenuBrowseInd_t;
```

The following table specifies the fields available in the Menu Browse Indication message.

Table 2-59. Menu Browse Indication Message Structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the menu browser
direction	menuBrowseDirection_t	menuBrowseUp_c menuBrowseDown_c menuBrowseLeft_c menuBrowseRight_c menuBrowseOk_c menuBrowseExit_c menuBrowseRefresh_c menuBrowseMax_c	The menu browsing direction
bUseSecurity	bool_t	{FALSE;TRUE}	Indicates whether the menu browse request was received encrypted or not

2.1.42.2 When Generated

The Menu Browse Indication message is generated by the profile layer of a light menu owner when a menu browse request frame is received from a menu browser.

2.1.42.3 Effect on Receipt

When the Menu Browse Indication message arrives the application is notified that the menu browser device has made a menu browsing request.

2.1.43 FSLProfile_DisplayCompleteIndToBrowserRequest

FSLProfile_DisplayCompleteIndToBrowserRequest instructs the profile layer of a menu owner with a light menu-owner library to inform the menu browser whether the menu displayer has acknowledged the display menu frame requests. To use this function, the application must link to the RF4CE_FSLProfile_MenuOwnerLight library.

2.1.43.1 Prototype

FSLProfile_DisplayCompleteIndToBrowserRequest has the following prototype:

```
uint8_t FSLProfile_DisplayCompleteIndToBrowserRequest(
                    uint8_t browserDeviceId,
                    uint8_t displayerDeviceId,
                    bool_t  bUseSecurity,
                    uint8_t status
);
```

The following table specifies the parameters for FSLProfile_DisplayCompleteIndToBrowserRequest.

Table 2-60. FSLProfile_DisplayCompleteIndToBrowserRequest Parameters

Name	Type	Valid range	Description
browserDeviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the menu displayer
displayerDeviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the menu displayer
bUseSecurity	bool_t	{FALSE, TRUE}	Whether to use secured transmission or not
status	uint8_t	gNWSuccess_cgNWNoResponse_c	The status of the browsing request

The possible return values for the FSLProfile_DisplayMenuMessageRequest API call are shown in the following table:

Table 2-61. FSLProfile_DisplayCompleteIndToBrowserRequest API Call Return Values

Type	Possible Values	Description
uint8_t	gNWNotPermitted_c gNWDenied_c gNWInvalidParam_c gNWDeviceIdNotPaired_c gNWSuccess_c	All possible return values are fully described in Section 2.1.43.3, “Effect on Receipt”.

2.1.43.2 Functionality

FSLProfile_DisplayMenuMessageRequest is used to inform the menu browser whether the menu displayer has received the display menu requests generated by the owner in response to the browser’s menu navigation requests.

2.1.43.3 Effect on Receipt

On receipt of FSLProfile_DisplayMenuMessageRequest, the profile layer verifies if all the conditions to transmit a menu browse complete frame are met.

If the light menu owner functionality has not been initialized the function exits with *gNWNotPermitted_c*. If the profile layer is busy with another request the function exits with *gNWDenied_c*. If the menu displayer's device Id is outside the bounds of the pair table the function exits with *gNWInvalidParams*_c. If the menu displayer's device Id points to an empty pair table entry index the function exits with *gNWDeviceIdNotPaired*_c. If a secured transmission is requested but the pairing link is not secured the function exits with *gNWInvalidParams_c*. If no memory could be allocated to construct the menu browse complete frame payload the function exits with *gNWNoMemory_c*.

Otherwise the request is accepted for processing and the function returns *gNWSuccess_c*.

The profile layer now constructs a menu browse complete message frame and sends it to the menu browser using acknowledged transmission. Once the menu owner's MAC layer replies with an ack the application will be notified via a display menu confirm message. The receiver is enabled indefinitely and the value of the receiver's initial active period is stored in an internal variable.

2.1.44 FSLProfile_InitOtapServerProcedure

FSLProfile_InitOtapServerProcedure initializes the over the air programming server functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_OtapServer library.

2.1.44.1 Function prototype

FSLProfile_InitOtapServerProcedure has the following prototype:

```
uint8_t FSLProfile_InitOtapServerProcedure(void);
```

FSLProfile_InitOtapServerProcedure has no parameters.

The possible return values for the FSLProfile_InitOtapServerProcedure API call are shown in the following table:

Table 2-62. FSLProfile_InitOtapServerProcedure API Call Return Values

Type	Possible Values	Description
uint8_t	gNWSuccess_c	The function call always returns *gNwSuccess_c*

2.1.44.2 Functionality

FSLProfile_InitOtapServerProcedure is used to enable the over the air programming functionality in the profile layer on the server side.

2.1.44.3 Effect on Receipt

On receipt of the FSLProfile_InitOtapServerProcedure function call the profile layer initializes the over the air programming functionality.

2.1.45 FSLProfile_InitOtapClientProcedure

FSLProfile_InitOtapClientProcedure initializes the over the air programming client functionality in the profile layer. To use this function, the application must link to the RF4CE_FSLProfile_OtapClient library.

2.1.45.1 Function prototype

FSLProfile_InitOtapClientProcedure has the following prototype:

```
uint8_t FSLProfile_InitOtapClientProcedure(void);
```

FSLProfile_InitOtapClientProcedure has no parameters.

The possible return values for the FSLProfile_InitOtapClientProcedure API call are shown in the following table:

Table 2-63. FSLProfile_InitOtapClientProcedure API Call Return Values

Type	Possible Values	Description
uint8_t	gNWSuccess_c gNWNoTimers_c	All possible return values are fully described in Section 2.1.45.3

2.1.45.2 Functionality

FSLProfile_InitOtapClientProcedure is used to enable the client over the air programming functionality in the profile layer.

2.1.45.3 Effect on Receipt

On receipt of the FSLProfile_InitOtapClientProcedure function call the profile layer configures itself to be able to handle the client over the air programming functionality. Timers are required for the profile layer for the OTAP client functionality and the the function call tries to allocate them (taking care to avoid double allocation in case of repeated calls). If no timers are available the function exits with *gNWNoTimers_c*. Otherwise the function call returns *gNWSuccess_c* and the client over the air programming functionality is available for use.

2.1.46 OTAP Server Query Next Image Indication

The OTAP Server Query Next Image Indication message informs the server application layer of the arrival of a query next image command frame from a client, requesting a new image.

2.1.46.1 Message Structure

The OTAP Server Query Next Image Indication message has the following structure:

```
typedef struct fslProfileOtapServerQueryNextImageInd_tag
{
  uint8_t                 deviceId;
  uint8_t                 fileVersion[4];
  uint8_t                 hardwareVersion[2];
}fslProfileOtapServerQueryNextImageInd_t;
```

The following table specifies the fields available in the OTAP Server Query Next Image Indication message.

Table 2-64. OTAP Server Query Next Image Indication message structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the device requesting a new flash image
fileVersion	uint8_t[4]	-	The client's current firmware version
hardwareVersion	uint8_t[2]	-	The client's hardware version

2.1.46.2 When Generated

The OTAP Server Query Next Image Indication message is generated by the arrival of a query next image request frame from a paired device at the profile layer. The message contains the requester's deviceId, current firmware version and hardware version.

2.1.46.3 Effect on Receipt

On receipt of the OTAP Server Query Next Image Indication message the application is informed that a paired device is looking for an upgraded firmware image. The application is also informed of the requester's current firmware and hardware version.

2.1.47 OTAP Server Query Next Block Request Indication

The OTAP Server Query Next Block Request Indication message informs the server application layer of the arrival of query next block frame from a paired client device. The client is requesting a new block (fragment) from the firmware image. The message specifies the firmware image version and block offset the client is requesting, plus the maximum block size it can accept.

2.1.47.1 Message Structure

The OTAP Server Query Next Block Request Indication message has the following structure:

```
typedef struct fslProfileOtapServerNextBlockReqInd_tag
{
  uint8_t                 deviceId;
```

```
  uint8_t                 fileVersion[4];
  uint8_t                 fileOffset[4];
  uint8_t                 maxDataSize;
}fslProfileOtapOrigNextBlockReqInd_t;
```

The following table specifies the fields available in the OTAP Server Query Next Block Request Indication message.

Table 2-65. OTAP Server Query Next Block Request Indication message structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the device requesting a new block
fileVersion	uint8_t[4]	-	The requested image's version
fileOffset	uint8_t[4]	-	The offset from the start of the requested block
maxDataSize	uint8_t	0x01-0xff	The maximum data size the client can accept

2.1.47.2 When Generated

The OTAP Server Query Next Block Request Indication message is generated by the arrival of a query next block request frame from a paired device at the profile layer.

2.1.47.3 Effect of Receipt

On receipt of the OTAP Server Query Next Block Request Indication message the application is informed that a paired client is requesting the next fragment of the upgraded firmware image. The application should check if it can provide the requested block, and, if so, respond by calling FSLProfile_OtapServerSend with a message type of *gFslProfileOtapNextBlockRespCmd_c*. The client times out waiting for the response in 10 seconds after sending the Query Next Block frame over the air.

2.1.48 OTAP Server Upgrade End Request Indication

The OTAP Server Upgrade End Request Indication message informs the server application layer that a client has received a complete new firmware image.

2.1.48.1 Message Structure

The OTAP Server Upgrade End Request Indication message has the following structure:

```
typedef struct fslProfileOtapServerUpgradeEndReqInd_tag
{
  uint8_t                 deviceId;
  uint8_t                 status;
  uint8_t                 fileVersion[4];
}fslProfileOtapServerUpgradeEndReqInd_t;
```

The following table specifies the fields available in the OTAP Server Upgrade End Request Indication message:

Table 2-66. OTAP Server Upgrade End Request Indication message structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the device requesting a new flash image
status	uint8_t	gNWSuccess_c	Implemented for later use
fileVersion	uint8_t[4]	-	The new firmware version of the client

2.1.48.2 When Generated

The OTAP Server Upgrade End Request Indication message is generated by the arrival of an upgrade end request frame from a client, signaling that it has received the the entire new firmware image.

2.1.48.3 Effect of Receipt

On receipt of the OTAP Server UPgrade End Request Indication message the application is informed that a client has completed the image transfer process. The application should respond with a call to FSLProfile_OtapServerSend and a message of type *gProfileCmfIdOtapUpgradeEndReq*.

2.1.49 FSLProfile_OtapServerSend

The FSLProfile_OtapServerSend function call allows a server application to transmit a message to the client. This includes information about available firmware images and actual pieces of the firmware images.

2.1.49.1 Function prototype

FSLProfile_OtapServerSend has the following prototype:

```
uint8_t FSLProfile_OtapOrigSend(
                  uint8_t                   deviceId,
                  fslProfileApptoOtapCmd_t* otapCmdMsg
);
```

The following table specifies the parameters for FSLProfile_OtapServerSend

Table 2-67. FSLProfile_OtapServerSend Parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1) or 0xFF	Pair table entry index of the client, or 0xFF if the server is multicasting a new image notification to all paired devices
otapCmdMsg	fslProfileAppToOtapCmd_t*	-	Pointer to a structure describing the message to send

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

Table 2-68. FSLProfile_OtapServerSend API Call Return Values

Type	Possible Values	Description
uint8_t	gNWSuccess_c gNWNotPermitted_c gNWNoMemory_c gNWDenied_c	All possible return values are fully described in Section 2.1.49.3

2.1.49.2 Functionality

FSLProfile_OtapServerSend is used to by the server application to send a message to one or more OTAP clients. The following messages can be sent:

- Notification that a new firmware image is available for download
- Information about the firmware image available for download
- Firmware image block
- The time when the client must start using the new firmware, measured from the moment it reports that it has received the entire image.

The otapCmdMsg parameter specifies the type of message that will be transmitted. It is a pointer to a structure with the following format:

```
typedef struct fslProfileApptoOtapCmd_tag
{
  fslProfileOTAPOrigCommand_t               cmdType;
  union {
    imageNotify_t           imageNotify;
    queryNextImgResp_t      queryNextImgResp;
    imageBlockResp_t        imageBlockResp;
    upgradeEndResp_t        upgradeEndResp;
  } cmdData;
}fslProfileApptoOtapCmd_t;
```

The fields in the structure are described in the following table:

Table 2-69. fslProfileApptoOtapCmd_t

Name	Type	Possible values	Description
cmdType	enumeration	gFslProfileOtapImageNotifyCmd gFslProfileOtapQueryNextImageRespCmd gFlsProfileOtapNextBlockRespCmd gFlsProfileOtapUpgradeEndRespCmd	Message type, identifies the valid union member
cmdData	union	-	Each union member is described below

The *otapCmdMsg* parameter may point to a structure either on the stack, or in an allocated message buffer. The function will not free the message before exiting. If the structure is in an allocated buffer, it is the application's responsibility to free it.

To send a message of a particular type set the *cmdType* parameter to the appropriate message type, and fill in the fields of the corresponding *cmdData* union member. Each message type and union member are described below.

2.1.49.2.1 Image notify message

The image notify message is used to announce that a new firmware image is available for download.

The corresponding command type is *gFslProfileOtapImageNotifyCmd* and the *msgData* union member is *imageNotify* of type *imageNotify_t*, which has the following format:

```
typedef struct imageNotify_tag
{
  uint8_t                 jitterVal;
  uint8_t                 imageType[2];
  uint8_t                 fileVersion[4];
}imageNotify_t;
```

The fields of the *imageNotify_t* structure are described in the following table:

Table 2-70. imageNotifiy_t fields

Name	Type	Possible values	Description
jitterVal	uint8_t	0x01 - 0x64	Server selected value used to avoid flooding the server with image requests. Receiving clients generate a random value between 0x01 and 0x64 and if it is larger than the received jitter value it will ignore the message. Setting the jitter to 0x32 will cause approximately half the clients to request the new image, 0x16 one quarter, and setting it to 0x64 will remove any restriction
imageType	uint8_t[2]	-	Image type, application specific
fileVersion	uint8_t[2]	-	New image firmware version

2.1.49.2.2 Query next image response

The query next image response is used to send a block of a firmware image to a client, in response to its request for it (i.e. the reception by the server of a Query Next Image Indication message).

The corresponding command type is *gFslProfileOtapQueryNextImageRespCmd* and the *msgData* union member is *queryNextImageResp* of type *queryNextImageResp_t*, which has the following format:

```
typedef struct queryNextImgResp_tag
{
  queryCmdStatusCode_t    status;
  uint8_t                 fileVersion[4];
  uint8_t                 imageSize[4];
  uint8_t                 crc[4];
  uint8_t                 bitmapLength;
  #if (defined(PROCESSOR_QE128) || defined(PROCESSOR_MC1323X))
  uint8_t                 bitmap[32];
  #else
  uint8_t                 bitmap[15];
```

```
  #endif
}queryNextImgResp_t;
```

The fields of the *queryNextImageResp_t* structure are described in the following table:

Table 2-71. queryNextImageResp_t fields

Name	Type	Possible values	Description
status	enumeration	gNWSuccess_c gFSLProfile_NotAuthorized gFSLProfile_NotImageAvailable	Query status: either success, or the server is not authorized to transmit the image, or the request was invalid
fileVersion	uint8_t[2]	-	Available firmware version
imageSize	uint8_t[4]	-	Available firmware image size in little endian
crc	uint8_t[4]	0x00000000 - 0x0000ffff	The 16 bit CRC-CCITT (0x1021) of the image
bitmapLength	uint8_t	0x00-0xFF	Bitmap string length
bitmap	uint8_t[32]	-	Bitmap of the flash pages which must/must not be overwritten by the new image. If the bit is 1, the corresponding flash page must be overwritten from the new image, if 0 it must remain the same. The size of the flash page depends on the MCU.

2.1.49.2.3 Image Block Response

The Image Block Response message is used to send a block of the upgraded firmware image to the client, in response to the client's request for that particular block (i.e. the reception by the server of an Image Block Request message).

The corresponding command type is *gFslProfileOtapImageBlockRespCmd* and the *msgData* union member is *imageBlockResp* of type *imageBlockResp_t*, which has the following format:

```
typedef struct imageBlockResp_tag
{
  nextBlockStatusCode_t   status;
  uint8_t                 fileOffset[4];
  uint8_t                 dataSize;
  uint32_t                addressImage;
  uint8_t*                pData;
}imageBlockResp_t;
```

The fields of the *imageBlockResp_t* structure are described in the following table:

Table 2-72. imageBlockResp_t fields

Name	Type	Possible values	Description
status	enumeration	gNWSuccess_c gFSLProfile_BlockStateAbort	Request status: either success, or the server is aborting the process
fileOffset	uint8_t[2]	-	Image file offset of the block to be sent

Table 2-72. imageBlockResp_t fields

dataSize	uint8_t[4]	0x00-0x46	Size of the block to be sent, in bytes
addressImage	uint32_t	-	Reserved for future use
pData	uint8_t*	-	Pointer to the actual firmware image block

2.1.49.2.4 Upgrade End Response

This message is sent in response to a notification from the client that it has received the entire new firmware image (i.e. the reception by the server of an Upgrade End Request message). It instructs the client to switch to the new image after a specified delay.

The corresponding command type is *gFslProfileOtapUpgradeEndRespCmd* and the *msgData* union member is *upgradeEndResp* of type *upgradeEndResp_t*, which has the following format:

```
typedef struct upgradeEndResp_tag
{
  uint8_t                 delayUntilUpgrade[4];
}upgradeEndResp_t;
```

The fields of the *upgradeEndResp_t* structure are described in the following table:

Table 2-73. upgradeEndResp_t fields

Name	Type	Possible values	Description
delayUntilUpgrade	uint8_t[4]	-	The time in milliseconds and in little endian format, when the client must switch to using the new firmware.

2.1.49.3 Effect on receipt

On Receipt of the FSLProfile_OtapOrigSend, the profile layer checks if can transmit the requested message. If the OTAP server functionality has not been initialized the function exits with *gNWNotPermitted_c*. If the profile layer is currently busy the function exits with *gNWDenied_c*. If no message buffers could be allocated for transmission the function exits with *gNWNoMemory_c*. Otherwise, the message frame is constructed and passed to the network layer for over the air transmission and the function returns *gNWSuccess_c*. When transmission is complete, the application will be notified via an OTAP Server Confirm message.

2.1.50 OTAP Server Confirm

The OTAP Server confirm message informs the server application about the completion of an OTAP transmission initiated by a call to FSLProfile_OtapServerSend.

2.1.50.1 Message Structure

The OTAP Server Confirm message has the following structure:

```
typedef struct upgradeEndResp_tag
{
  uint8_t                 delayUntilUpgrade[4];
}upgradeEndResp_t;
```

The following table specifies the fields available in the OTAP Server Confirm message.

Table 2-74. OTAP Server Confirm message structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c or the network layer error code	Indicates either the successful transmission of the OTAP message or the error that has occured
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1) or 0xFF	OTAP message destination

2.1.50.2 When Generated

The OTAP Server Confirm message is generated by the profile layer at the conclusion of an OTAP message transmission process.

2.1.50.3 Effect on Receipt

On receipt of an OTAP Server Confirm message the application layer is notified of the result of the OTAP message transmission process.

2.1.51 OTAP Client Image Notify Indication

The OTAP Client Image Notify Indication message is generated by the arrival of an OTAP Image Notify message from a paired device.

2.1.51.1 Message Structure

The OTAP Client Image Notify Indication message has the following structure:

```
typedef struct fslProfileOtapClientImageNotifyInd_tag
{
  uint8_t                 deviceId;
  uint8_t                 fileVersion[4];
} fslProfileOtapClientImageNotifyInd_t;
```

The following table specifies the fields in the OTAP Client Image Notify message:

Table 2-75. OTAP Server Confirm message structure

Field	Type	Possible Values	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	device Id of the server
fileVersion	uint8_t[4]	-	The available firmware version

2.1.51.2 When Generated

The OTAP Client Image Notify Indication message is generated by the arrival of and OTAP Client Image frame from a paired OTAP server. The profile layer first generates a random value in the [0x01, 0x64] interval and compares it to the jitter value in the OTAP Client Image frame. The indication message is only sent to the application if the generated value is less than the jitter; otherwise the the received frame is silently discarded.

2.1.51.3 Effect on Receipt

On receipt of the OTAP Client Image Notify Indication message the client application is informed that a new firmware image with the specified version is available from the server with the specified device Id.

2.1.52 FSLProfile_OtapQueryNextImageRequest

FSLProfile_OtapQueryNextImageRequest is used by clients to initiate an upgrade image download process.

2.1.52.1 Function Prototype

FSLProfileOtapQueryNextImageRequest has the following prototype:

```
uint8_t FSLProfile_OtapQueryNextImageRequest(
                    uint8_t deviceId,
                    const uint8_t fileVersion[4],
                    const uint8_t hardwareVersion[2]
);
```

The following table specifies the parameters for FSLProfile_OtapQueryNextImageRequest:

Table 2-76. FSLProfile_OtapQueryNextImageRequest parameters

Name	Type	Valid range	Description
deviceId	uint8_t	0 – (gMaxPairTableEntries_c - 1)	Pair table entry index of the OTAP server
fileVersion	const uint8_t[4]	-	the client's current firmware version
hardwareVersion	const uint8_t[4]	-	the client's hardware version

The possible return values for the FSLProfile_OtapQueryNextImageRequest API call are shown in the following table:

Table 2-77. FSLProfile_OtapQueryNextImageRequest API call return Values

Type	Possible Values	Description
uint8_t	gNWDenied_c gNWNotPermitted_c gNWInvalidParam_c gNWNoMemory_c gNWSuccess_c	All possible return values are fully described in Section 2.1.52.3.

2.1.52.2 Functionality

FSLProfile_OtapQueryNextImageRequest is used on an OTAP client to start the process of downloading a new firmware image from an OTAP server.

First a query next image frame will be sent to the server to inquire about available upgrade images. If none are available, the application will receive an OTAP Query Next Image Confirm message with a status of *gFSLProfile_NotImageAvailable*. If the server replies that the client is not authorized to receive an upgrade, the application will receive and OTAP Query Next Image Confirm message with a status of *gFSLProfile_NotAuthorized*.

If there is an upgrade firmware image available the actual download process will begin. The profile layer will ask for the firmware image, block by block and write it into the external EEPROM. Once download is complete the CRC will be checked. If the CRC is invalid, the process is aborted and the application will.receive an OTAP Query Next Image Confirm message with a status of *gNWAborted_c*. Otherwise the server will be notified that download is complete and it will reply with the time when the client must switch to the new firmware. The application will receive an OTAP Query Next Image Confirm message with a status of *gNWSuccess_c* and the *delayUntilUpgrade* field will contain the time, in milliseconds, when the client must switch to using the new firmware.

If there is a lower layer error at any point during the process, it will be aborted and the application will receive an OTAP Query Next Image Confirm message with a status containing the lower layer error code.

2.1.52.3 Effect on receipt

When the FSLProfile_OtapQueryNextImageRequest function is called, the profile layer checks whether it can transmit the query next image frame to the specified OTAP server. If the OTAP client functionality has not been initialized the function exits with *gNWNotPermitted_c*. If the profile layer is currently busy the function exits with *gNWDenied_c*. If needed message buffers could not be allocated the function exits with *gNWNoMemory_c*. Otherwise, the query next image frame is constructed and passed to the network layer for transmission, and the function exits with *gNWSuccess_c*.

2.1.53 Over the Air Programming (OTAP) Client Query Next Image Confirm

The OTAP Client Query Next Image Confirm message informs the application about the completion of an upgrade firmware image download process initiated by a FSLProfile_QueryNextImageRequest function call.

2.1.53.1 Message structure

The OTAP Client Query Next Image Confirm message has the following structure:

```
typedef struct fslProfileOtapClientQueryNextImgCnf_tag
{
  uint8_t                 status;
  uint8_t                 delayUntilUpgrade[4];
}fslProfileOtapClientQueryNextImgCnf_t;
```

The following table describes the fields in the OTAP Client Query Next Image Confirm message:

Table 2-78. OTAP Server Confirm message structure

Field	Type	Possible Values	Description
status	uint8_t	gNWSuccess_c gNWAborted_c gFSLProfile_NotAuthorized gFSLProfile_NotImageAvailable or the network layer error code	All status codes are fully described in Section 2.1.52.2
fileVersion	uint8_t[4]	-	The time when the client needs to switch to using the new firmware. Unused when the status is different than gNWSuccess_c

2.1.53.2 When Generated

The OTAP Client Query Next Image Confirm message is generated when an upgrade image download is finished (successfully or not). See Section 2.1.52.2 for a description of the download process.

2.1.53.3 Effect on Receipt

On receipt of the OTAP Client Query Next Image Confirm message the application is notified about the completion of an upgrade image download process. If the process was successful, the application is also informed about the time when it must re-boot with the new firmware image.