802.15.4 Media Access Controller (MAC) Over-The-Air (OTA) Programmer

Demonstration Application
User's Guide

Document Number: 802154MACOTAPUG
Rev. 2.0
2/2012

ARM POWERED®

freescale™
semiconductor

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

Audience v
Organization v
Revision History v
Definitions, Acronyms, and Abbreviations vi
References vi

Chapter 1
System Architecture

1.1 System Overview 1-1
1.2 Source File Hierarchy 1-2
1.2.1 OTA Programmer Applications 1-2
1.3 Creating the Network 1-3
1.4 Transferring Data 1-3
1.5 Monitoring Network Activity 1-3
1.5.1 Monitoring Messages Using the Serial Port 1-4

Chapter 2
Data Structures and Commands

2.1 Understanding OTA Header Format 2-1
2.1.1 OTA Upgrade File Identifier 2-2
2.1.2 OTA Header Version 2-2
2.1.3 OTA Header Length 2-2
2.1.4 OTA Header Field Control 2-3
2.1.5 Manufacturer Code 2-3
2.1.6 Image Type 2-3
2.1.7 File Version 2-4
2.1.8 ZigBee Stack Version 2-5
2.1.9 OTA Header String 2-5
2.1.10 Total Image Size 2-5
2.1.11 Minimum Hardware Version 2-6
2.1.12 Maximum Hardware Version 2-6
2.2 Understanding Sub-element Format 2-6
2.2.1 Tag ID 2-6
2.2.2 Length Field 2-6
2.2.3 Data 2-7
2.2.4 Tag Identifiers 2-7
2.3 OTA Programmer Commands 2-7
2.3.1 OTA Status Code 2-8
2.3.2 Image Notify Command 2-8
2.3.3 Query Next Image Request Command 2-10
2.3.4 Query Next Image Response Command 2-11
2.3.5 Image Block Request Command 2-12

2.3.6 Image Block Response Command 2-13
2.3.7 Upgrade End Request Command 2-15
2.3.8 Upgrade End Response Command 2-15
2.4 OTA Programmer Image Format 2-17

Chapter 3
Software Implementation

3.1 PC MAC OTA Programmer Application 3-1
3.2 OTA Programmer Server (PAN Coordinator) 3-2
3.2.1 Server ZTC 3-2
3.2.2 Server AppTask 3-3
3.2.3 OTA Transfer Diagram 3-5
3.3 Client AppTask 3-5

About This Book

This guide provides information about the 802.15.4 MAC Over-the-Air Programmer (MAC OTA Programmer) based on the Freescale 802.15.4 Media Access Controller (MAC) implementation. The demonstration application presented in this document can be used with the MC1320x, MC1321x and the MC1323x IEEE 802.15.4 compliant wireless platforms.

Audience

This document is intended for application developers building 802.15.4/ZigBee applications.

Organization

This document is organized into the following chapters.

Chapter 1 **System Architecture** — Describes the hardware architecture of the 802.15.4 MAC Over-the-Air Programmer (OTA Programmer) demonstration and guides users through the required steps for starting the network and performing various operations.

Chapter 2 **Data Structures and Commands** — Details the OTA file format which is composed of a header followed by a number of sub-elements.

Chapter 3 **Software Implementation** — Describes the PC, server and client software.

Revision History

The following table summarizes revisions to this document since the previous release (Rev. 1.0).

Revision History

Location	Revision
Entire Document	Updates for MC1323x 128K and added support for MC1320x and MC1321x

Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document.

BDM debugger	A debugger using the BDM interface for communication with the MCU. An example is the P&E BDM Multilink debugger for HCS08.
BDM	Background Debug Module
EVB	Evaluation Board
EVK	Evaluation Kit
FFD	Full Function Device (Coordinator)
GUI	Graphical User Interface
MAC	Medium Access Control
MCU	Micro Controller Unit
NVM	None-Volatile Memory
PC	Personal Computer
PCB	Printed Circuit Board
RFD	Reduced Function Device (End Device)

References

The following sources were referenced to produce this book:

[1] IEEE Computer Society, IEEE Std 802.15.4™-2003, May 12th 2003

[2] 802.15.4 MAC/PHY Software Reference Manual, 802154MPSRM, Freescale Semiconductor, 2004, 2005, 2006

[3] 802.15.4 MAC MyWirelessApp User's Guide, 802154MWAUG, revision 1.1, Freescale Semiconductor, September 2007

[4] BeeKit Wireless Connectivity Toolkit User’s Guide (BKWCTKUG), revision 1.0, March 2007

Chapter 1
System Architecture

This chapter describes the hardware architecture of the 802.15.4 MAC over-the-air programmer demonstration and guides users through the required steps for starting the network and performing various operations.

The examples as shown in this guide use one 1323x-Remote Extender Motherboard (1323x-REM) as the PAN Coordinator (Server application) and other 1323x-REMs as End Devices (Client application).

The demonstration applications are included in MAC HCS08 Codebase, starting with the BeeKit MAC Codebase version 2.3.0.

Updates for the Codebase, applications and examples can be downloaded from the Freescale ZigBee site at: www.freescale.com/802154.

1.1 System Overview

System setup consists of one PAN Coordinator and one to four End Devices as shown in Figure 1-1. The serial connections with PCs are for displaying different status messages and for sending text messages to the devices in the network. The OTA Programmer Client serial connections are optional because the network can function in autonomous mode.

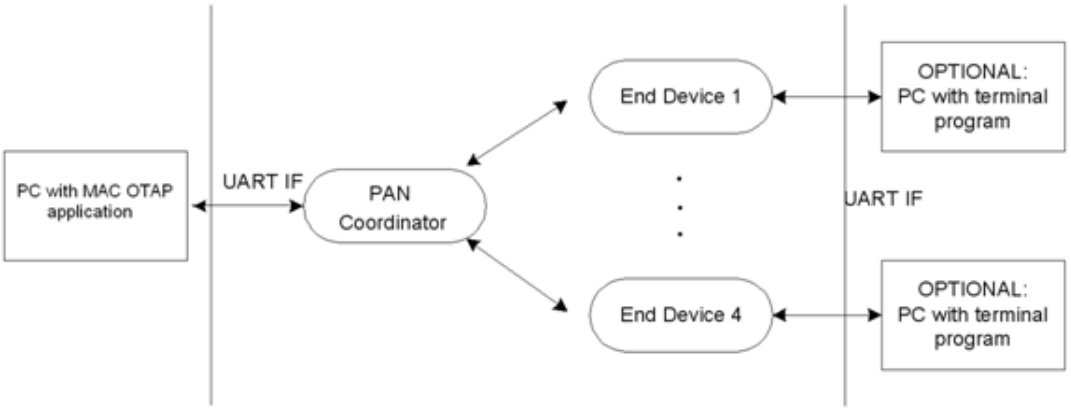

Figure 1-1. Block Diagram

1.2 Source File Hierarchy

Each OTA Programmer demonstration application represents a separate project in the MAC HCS08 Codebase. Source files for each OTA Programmer demonstration application are found after exporting each project from BeeKit at the following path:

```
..\Project name\Application\Source
```

Where `Project name` is the name of the BeeKit project (e.g. OTA Programming Demo (Server)).

Clear references are made to the appropriate source file. This guide focuses on the 802.15.4 Standard and Freescale specific issues only. It does not explain in detail the state machine that the code contains.

OTA Programmer demonstration source files have the same name inside the projects (MApp.c). The project name is the one which differentiates each application.

The following projects are available in BeeKit. Each new project adds additional functionality until a fully functional application has been developed for both a server and a client. Project names ending in '(Server)' demonstrate coordinator behavior and project names with ending in '(Client)' demonstrate device behavior.

Proper initialization of the MAC is handled and the MLME main function is called in the main loop.

1.2.1 OTA Programmer Applications

1.2.1.1 OTA ProgrammingDemoApp (Server)

This application is based on the MyWirelessApp demonstration Non Beacon (Coordinator) with some portion of the ZTC module enabled. In this application, the server receives from a PC application a new image, which will be transferred Over The Air (OTA) to a Client device.

If `gOtapExternalMemorySupported_d` is set to TRUE, the server will first store the image into the External Memory.

1.2.1.2 OTA ProgrammingDemoApp (Client)

This application is based on the MyWirelessApp demonstration Non Beacon (EndDevice). In this application, the Client Device receives an image notify when a new image is available or it can query the server any time. The image is transferred OTA, and the Client will store it into the External Memory. After the transfer is finished, the MCU resets giving control to the bootloader. The bootloader writes the image from the External Memory into the Internal FLASH and resets the MCU again (giving control to the application).

NOTE

For information about the Task Scheduler, Setup and Initialization of the MAC layer, Starting and Joining a PAN and Indirect Data Transfer, see the *MyWirelessApp User's Guide*.

1.3 Creating the Network

Create the network by powering on the boards. One of the boards must be running the PAN Coordinator firmware. After power-up, a switch (SW1 to SW4 must be pressed to start the PAN Coordinator. When the starting phase is finished, the PAN Coordinator LED's turn off.

The PAN Coordinator does not print a message on the terminal. The Serial session is used by the ZTC module to communicate with the PC application, so no terminal console should be opened.

The End Device prints the following messages when it successfully associates to the PAN Coordinator:

```
PAN Coordinator address
PAND ID
RF channel
Beacon information
Link Quality Indicator (LQI)
Short address received
```

Data can be exchanged after the End Device is associated to the PAN Coordinator.

1.4 Transferring Data

When a new image is available, the PAN Coordinator (OTA Programmer Server) informs the associated device that a new image is available. A device wanting to download the new image sends a series of requests to the server. The image is transferred OTA in blocks and the End Device (OTA Programming Client) saves every block in the External Memory. When the transfer is finished, the End Device resets and the bootloader writes the new image from the External Memory to the FLASH memory. After this process is finished, the new image runs.

NOTE

This OTAP Demo Server Application can transfer an image to only one Client Device at a time.

1.5 Monitoring Network Activity

User interaction with the OTA Programming demonstration application takes place through a terminal console connected to the End Device UART ports and through the PC application connected to the Coordinator's UART port as shown in Figure 1-1.

NOTE

Do not open a terminal console for the Coordinator (OTA Programmer Server).

Query the Server for a new image at anytime by pressing any switch on the Client Device board.

1.5.1 Monitoring Messages Using the Serial Port

The End Devices output messages to the serial port. To monitor these messages, the boards must be connected to a PC through the serial interface.

Before attempting to monitor messages, ensure that a Hyper Terminal style program is properly configured as follows:

1. Launch Hyper Terminal (or a Hyper Terminal style program). Select the Com Port to which the board is connected. Configure the Port Settings communication options as shown in Figure 1-2.

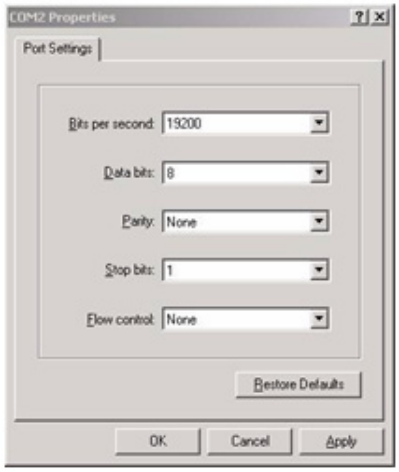

Figure 1-2. Properties Window

2. Click the OK button and the Hyper Terminal main window appears as shown in Figure 1-3.

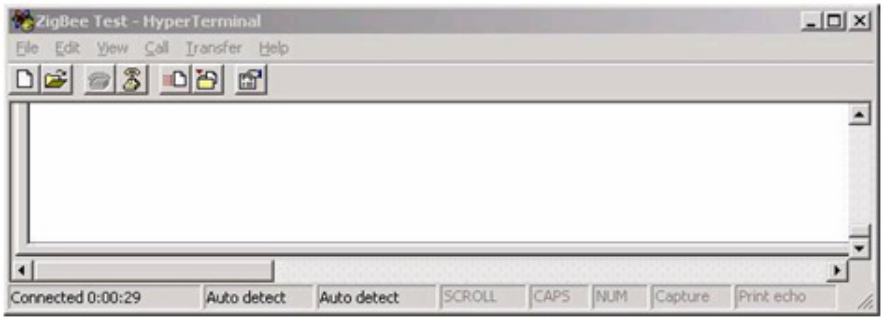

Figure 1-3. Hyper Terminal Main Window

3. From the Hyper Terminal Main window click on File -> Properties.
4. Click on the Settings tab, then click on the ASCII button.
5. Select the following options as shown in Figure 1-4:
   a) Send line ends with line feeds
   b) Echo typed characters locally
   c) Leave the other options set as shown in Figure 1-4.

Figure 1-4. ASCII Setup Window

For example, the Coordinator is linked to COM3 and the End Device is linked to COM4. Perform the following steps:

1. Start a Hyper Terminal instance for COM4.
2. Turn on the Coordinator and press any switch.
3. Turn on the End Device. The appropriate Hyper Terminal window outputs the following message:

```
Press any switch on board to start running the application.
```

After pressing a switch on the End Device board, and after it has connected to the PAN (the short address 1 has been assigned by the Coordinator), the appropriate Hyper Terminal window shows a message similar to the following (some values may be different):

```
Found a coordinator with the following properties:
Address . . . . . 0xCAFE
PAN ID. . . . . 0xBEEF
Logical Channel. . .0x0B
Beacon Spec. . . . .0xCFFF
Link Quality. . . . 0x52

Associating to PAN Coordinator on channel 0x0B
Sending MLME-Associate Request message to the MAC…Done
Association successful.
We were assigned the short address 0x0001

Press any switch to query for a new image.
```

4. Press one of the four buttons on the End Device. The Coordinator receives a query for a new image.

Chapter 2
Data Structures and Commands

The OTA file format is composed of a header followed by a number of sub-elements. The header describes general information about the file such as version, the manufacturer that created it, and the device it is intended for. Sub-elements in the file may contain upgrade data for the embedded device, certificates, configuration data, log messages, or other manufacturer-specific pieces.

Table 2-1. Sample OTA File

Octets	Variable	Variable	Variable	Variable
Data	OTA Header	Upgrade Image	Signer Certificate	Signature

NOTE

Signer Certificate and Signature sub-elements are not used by the described application.

The OTA header does not describe details of the particular sub-elements. Each sub-element is self-describing. With exception of a few sub-elements, the interpretation of the data contained is up to the manufacturer of the device.

2.1 Understanding OTA Header Format

Table 2-2. OTA Header Fields

Octets	Data Types	Field Names	Mandatory/Optional
4	Unsigned 32-bit integer	OTA upgrade file identifier	M
2	Unsigned 16-bit integer	OTA Header version	M
2	Unsigned 16-bit integer	OTA Header length	M
2	Unsigned 16-bit integer	OTA Header field control	M
2	Unsigned 16-bit integer	Manufacturer code	M
2	Unsigned 16-bit integer	Image type	M
4	Unsigned 32-bit integer	File version	M

Table 2-2. OTA Header Fields

Octets	Data Types	Field Names	Mandatory/Optional
2	Unsigned 16-bit integer	ZigBee Stack version	M
32	Character string	OTA Header string	M
4	Unsigned 32-bit integer	Total Image size (including header)	M
0/1*	Unsigned 8-bit integer	Security credential version	O
0/8*	IEEE Address	Upgrade file destination	O
0/2	Unsigned 16-bit integer	Minimum hardware version	O
0/2	Unsigned 16-bit integer	Maximum hardware version	O

NOTE

Not implemented in the described applications.

2.1.1 OTA Upgrade File Identifier

The value is a unique 4-byte value that is included at the beginning of all ZigBee OTA upgrade image files to quickly identify and distinguish the file as being a ZigBee OTA cluster upgrade file, without having to examine the whole file content. This helps distinguishing the file from other file types on disk. The value is defined to be “0x0BEEF11E”.

2.1.2 OTA Header Version

The value enumerates the version of the header and provides compatibility information. The value is composed of a major and minor version number (one byte each). The high byte (or the most significant byte) represents the major version and the low byte (or the least significant byte) represents the minor version number. A change to the minor version means the OTA upgrade file format is still backward compatible, while a change to the major version suggests incompatibility.

The current OTA header version is 0x0100 with major version of “01” and minor version of “00”.

2.1.3 OTA Header Length

This value indicates full length of the OTA header in bytes, including the OTA upgrade file identifier, OTA header length itself to any optional fields. The value insulates existing software against new fields that may be added to the header. If new header fields added are not compatible with current running software, the implementations should process all fields they understand and then skip over any remaining bytes in the header to process the image or signing certificate. The value of the header length depends on the value of the OTA header field control, which dictates which optional OTA header fields are included.

The length of the OTA header implemented by the demonstration apps is 60 bytes.

2.1.4 OTA Header Field Control

The bit mask indicates whether additional information such as Image Signature or Signing Certificate are included as part of the OTA Upgrade Image.

Table 2-3. OTA Header Field Control Bitmask

Bits	Name
0	Security Credential Version Present
1	Device Specific File
2	Hardware Versions Present
3–15	Reserved

Security credential version present bit indicates whether security credential version field is present or not in the OTA header.

Device specific file bit in the field control indicates that this particular OTA upgrade file is specific to a single device.

Hardware version present bit indicates whether minimum and maximum hardware version fields are present in the OTA header or not.

NOTE

Only Hardware Versions are present in the header (Field Control = 0x04).

2.1.5 Manufacturer Code

This is the ZigBee assigned identifier for each member company. When used during the OTA upgrade process, manufacturer code value of 0xffff has a special meaning of a wild card. The value has a ‘match all’ effect. OTA server may send a command with wild card value for manufacturer code to match all client devices from all manufacturers.

NOTE

The manufacturer code used by the demonstration apps is 0x1004.

2.1.6 Image Type

The manufacturer should assign an appropriate and unique image type value to each of its devices to distinguish the products. This is a manufacturer specific value. However, the OTA Upgrade cluster has reserved the last 64 values of image type value to indicate specific file types such as security credential, log, and configuration. When a client wants to request one of these specific file types, it uses one of the reserved image type values instead of its own (manufacturer specific) value when requesting the image via Query Next Image Request command.

Table 2-4. Image Type Values

File Type Values	File Type Description
0x0000 – 0xffbf	Manufacturer Specific
0xffc0	Security credential
0xffc1	Configuration
0xffc2	Log
0xffc3 – 0xfffe	Reserved (unassigned)
0xffff	Reserved: wild card

Image type value of 0xffff has a special meaning of a wild card. The value has a ‘match all’ effect. For example, the OTA server may send Image Notify command with image type value of 0xffff to indicate to a group of client devices that it has all types of images for the clients. Additionally, the OTA server may send Upgrade End Response command with image type value of 0xffff to indicate a group of clients, with disregard to their image types, to upgrade.

NOTE

The image type used in this demonstration is 0x0000.

2.1.7 File Version

For firmware image, the file version represents the release and build number of the image’s application and stack. The application release and build numbers are manufacturer specific, however, each manufacturer should obtain stack release and build numbers from their stack vendor. OTA Upgrade cluster makes the recommendation regarding how the file version should be defined, in an attempt to make it easy for humans and upgrade servers to determine which versions are newer than others. The upgrade server should use this version value to compare with the one received from the client.

The server may implement more sophisticated policies to determine whether to upgrade the client based on the file version. A higher file version number indicates a newer file.

Table 2-5. Recommended File Version Definition

Application Release	Application Build	Stack Release	Stack Build
1 byte	1 byte	1 byte	1 byte
8-bit integer	8-bit integer	8-bit integer	8-bit integer

For example:

- File version A: 0x10053519 represents application release 1.0 build 05 with stack release 3.5 b19.
- File version B: 0x10103519 represents application release 1.0 build 10 with stack release 3.5 b19.
- File version C: 0x10103701 represents application release 1.0 build 10 with stack release 3.7 b01.
- File version B is newer than File version A because its application version is higher while File version C is newer than File version B because its stack version is higher.

For device-specific files, the file version value may be defined differently than that for firmware image to represent version scheme of different image types. For example, version scheme for security credential data may be different than that of log or configuration file. The specific implementation is manufacturer specific.

NOTE

A binary-coded decimal convention (BCD) concept is used here for version number. This is to allow easy conversion to decimal digits for printing or display, and allows faster decimal calculations.

2.1.8 ZigBee Stack Version

This information indicates the ZigBee stack version that is used by the application. This provides the upgrade server the ability to coordinate the distribution of images to devices when the upgrades may cause a major jump that usually breaks the over-the-air compatibility, for example, from ZigBee Pro to ZigBee IP. The values as shown in the following table represent all currently available ZigBee stack versions

Table 2-6. ZigBee Stack Version Values

ZigBee Stack Version Values	Stack Name
0x0000	ZigBee 2006
0x0001	ZigBee 2007
0x0002	ZigBee Pro
0x0003	ZigBee IP
0x0004 – 0xffff	Reserved

NOTE

The ZigBee Stack Version parameter is ignored by the applications.

2.1.9 OTA Header String

This is a manufacturer specific string that may be used to store other necessary information as seen fit by each manufacturer. The idea is to have a human readable string that can prove helpful during development cycle. The string is defined to occupy 32 bytes of space in the OTA header.

NOTE

The string used by this demonstration is “BeeStack Image File”.

2.1.10 Total Image Size

The value represents the total image size in bytes. This is the total of data in bytes that is transferred over-the-air from the server to the client. In most cases, the total image size of an OTA upgrade image file is the sum of the OTA header and the actual file data (along with its tag) lengths. If the image is a signed

image and contains a certificate of the signer, then the Total image size also includes the signer certificate and the signature (along with their tags) in bytes.

This value is crucial in the OTA upgrade process. It allows the client to determine how many image request commands to send to the server to complete the upgrade process.

2.1.11 Minimum Hardware Version

The value represents the earliest hardware platform version this image should be used on. This field is defined as follows:

Table 2-7. Hardware Version Format

Version	Revision
1 byte	1 byte
8-bit integer	8-bit integer

The high byte represents the version and the low byte represents the revision.

2.1.12 Maximum Hardware Version

The value represents the latest hardware platform this image should be used on. The field is defined the same as the Minimum Hardware Version (above).

The hardware version of the device should not be earlier than the minimum (hardware) version and should not be later than the maximum (hardware) version to run the OTA upgrade file.

2.2 Understanding Sub-element Format

Sub-elements in the file are composed of an identifier followed by a length field, followed by the data. The identifier provides for forward and backward compatibility as new sub-elements are introduced. Existing devices that do not understand newer sub-elements may ignore the data.

Table 2-8. Sub-element format

Octets	2-bytes	4-bytes	Variable
Data	Tag ID	Length Field	Data

2.2.1 Tag ID

The tag identifier denotes the type and format of the data contained within the sub-element. The identifier is one of the values from the following table.

2.2.2 Length Field

This value dictates the length of the rest of the data within the sub-element in bytes. It does not include the size of the Tag ID or the Length Fields.

2.2.3 Data

The length of the data in the sub-element must be equal to the value of the Length Field in bytes. The type and format of the data contained in the sub-element is specific to the Tag.

2.2.4 Tag Identifiers

Sub-elements are generally specific to the manufacturer and the implementation. However this specification has defined a number of common identifiers that may be used across multiple manufacturers.

Table 2-9. Tag Identifiers

Tag Identifiers	Description
0x0000	Upgrade Image
0x0001	ECDSA Signature
0x0002	ECDSA Signing Certificate
0x0003 – 0xefff	Reserved
0xf000 – 0xffff	Manufacturer Specific Use

Manufacturers may define tag identifiers for their own use and dictate the format and behavior of devices that receive images with that data.

NOTE

The TagId for the Sub Element containing the Sector Bitmap is 0xf000 and the TagId for the Sub Element containing the CRC is 0xf100.

2.3 OTA Programmer Commands

OTA Upgrade cluster commands, the frame control value is comprised of the following:

- Frame type is 0x01 — Commands are cluster specific (not a global command).
- Manufacturer specific is 0x00 — Commands are not manufacturer specific.
- Direction — Is either 0x00 (client->server) or 0x01 (server->client) depending on the commands.

Command identifier values are shown in the following table.

Table 2-10. OTA Upgrade cluster command frames

Command Identifier Field Value	Description	Direction	Disable Default Response	Mandatory /Optional
0x00	Image Notify	Server -> Client(s) (0x01)	Set if sent as broadcast or multicast; Not Set if sent as unicast	O

Table 2-10. OTA Upgrade cluster command frames

0x01	Query Next Image Request	Client -> Server (0x00)	Not Set	M
0x02	Query Next Image Response	Server -> Client (0x01)	Set	M
0x03	Image Block Request	Client -> Server (0x00)	Not Set	M
0x04	Image Page Request	Client -> Server (0x00)	Not Set	O
0x05	Image Block Response	Server -> Client (0x01)	Set	M
0x06	Upgrade End Request	Client -> Server (0x00)	Not Set	M
0x07	Upgrade End Response	Server -> Client (0x01)	Set	M
0x08	Query Specific File Request	Client -> Server (0x00)	Not Set	O
0x09	Query Specific File Response	Server -> Client (0x01)	Set	O

2.3.1 OTA Status Code

OTA Upgrade cluster uses ZCL defined status codes during the upgrade process. These status codes are included as values in status field in payload of OTA Upgrade cluster's response commands and in default response command.

Table 2-11. Status Code defined and used by OTA Upgrade Cluster

ZCL Status Code	Value	Description
SUCCESS	0x00	Success Operation
ABORT	0x95	Failed case when a client or a server decides to abort the upgrade process.
NOT_AUTHORIZED	0x7E	Server is not authorized to upgrade the client
INVALID_IMAGE	0x96	Invalid OTA upgrade image (ex. failed signature validation or signer information check or CRC check)
WAIT_FOR_DATA	0x97	Server does not have data block available yet
NO_IMAGE_AVAILABLE	0x98	No OTA upgrade image available for a particular client
MALFORMED_COMMAND	0x80	The command received is badly formatted. It usually means the command is missing certain fields or values included in the fields are invalid ex. invalid jitter value, invalid payload type value, invalid time value, invalid data size value, invalid image type value, invalid manufacturer code value and invalid file offset value
UNSUP_CLUSTER_COMMAND	0x81	Such command is not supported on the device
REQUIRE_MORE_IMAGE	0x99	The client still requires more OTA upgrade image files to successfully upgrade

2.3.2 Image Notify Command

The purpose of sending Image Notify command is so the server has a way to notify client devices of when a new upgrade image is available for them. It eliminates the need for client devices having to check with the server periodically of when the new images are available. However, all client devices still need to send in Query Next Image Request command to officially start the OTA upgrade process.

NOTE

This Demo Server sends an ImageNotify command only to the device with address 0x0001.

2.3.2.1 Payload Format

Table 2-12. Format of Image Notify Command Payload

Octets	1	1	0/2	0/2	0/4
Data Type	8-bit Enumeration	Unsigned 8-bit	Unsigned 16-bit	Unsigned 16-bit	Unsigned 32-bit
Field Name	Payload type	Query jitter	Manufacturer code	Image type	(new) File version

2.3.2.2 Payload Field Definitions

Image Notify Command Payload Type

Table 2-13. Image Notify Command Payload Type

Payload Type Values	Description
0x00	Query jitter
0x01	Query jitter and manufacturer code
0x02	Query jitter, manufacturer code, and image type
0x03	Query jitter, manufacturer code, image type, and new file version
0x04 – 0xff	Reserved

Query Jitter

To proceed, the device randomly chooses a number between 1 and 100 and compares it to the value of the QueryJitter value in the received message. If the generated value is less than or equal to the received value for QueryJitter, it queries the upgrade server. If not, then it discards the message and no further processing continues.

NOTE

This parameter is not used by the demonstration application (it's always 100).

Manufacturer Code

Manufacturer code when included in the command should contain the specific value that indicates certain manufacturer. If the server intends for the command to be applied to all manufacturers then the value should be omitted.

Image Type

Image type when included in the command should contain the specific value that indicates certain file type. If the server intends for the command to be applied to all image type values then the wild card value (0xffff) should be used.

(new) File Version

The value is the OTA upgrade file version that the server tries to upgrade client devices in the network to. If the server intends for the command to be applied to all file version values then the wild card value (0xffffffff) should be used.

NOTE

The application uses 0xffffffff value. In this case downgrades can also be done.

2.3.3 Query Next Image Request Command

2.3.3.1 Payload Format

Table 2-14. Format of Query Next Image Request Command Payload

Octets	1	2	2	4	0/2
Data Type	Unsigned 8-bit	Unsigned 16-bit	Unsigned 16-bit	Unsigned 32-bit	Unsigned 16-bit
Field Name	Field control	Manufacturer code	Image type	(Current) File version	Hardware version

2.3.3.2 Payload Field Definitions

Query Next Image Request Command Field Control

The field control indicates whether additional information such as device's current running hardware version is included as part of the Query Next Image Request command.

Table 2-15. Query Next Image Request Field Control Bitmask

Bits	Name
0	Hardware Version Present
1–7	Reserved

Manufacturer Code

The value is the device's assigned manufacturer code. Wild card value is not used in this case.

Image Type

The value is between 0x0000–0xffbf (manufacturer specific value range).

(current) File Version

The file version included in the payload represents the device's current running image version. Wild card value is not used in this case.

(optional) Hardware Version

The hardware version if included in the payload represents the device's current running hardware version. Wild card value is not used in this case.

2.3.4 Query Next Image Response Command

2.3.4.1 Payload Format

Table 2-16. Format of Query Next Image Response Command Payload

Octets	1	0/2	0/2	0/4	0/4
Data Type	Unsigned 8-bit	Unsigned 16-bit	Unsigned 16-bit	Unsigned 32-bit	Unsigned 32-bit
Field Name	Status	Manufacturer code	Image type	File version	Image size

2.3.4.2 Payload Field Definitions

Query Next Image Response Status

Only if the status is SUCCESS that other fields are included. For other (error) status values, only status field is present.

Manufacturer Code

The value is the one received by the server in the Query Next Image Request command.

Image Type

The value is the one received by the server in the Query Next Image Request command.

File Version

The file version indicates the image version that the client is required to install. The version value may be lower than the current image version on the client if the server decides to perform a downgrade. The version value may be the same as the client's current version if the server decides to perform a reinstall. However, in general, the version value should be higher than the current image version on the client to indicate an upgrade.

Image Size

The value represents the total size of the image (in bytes) including header and all sub-elements.

2.3.5 Image Block Request Command

2.3.5.1 Payload Format

Table 2-17. Format of Image Block Request Command Payload

Octets	1	2	2	4	4	1	0/8
Data Type	Unsigned 8-bit	Unsigned 16-bit	Unsigned 16-bit	Unsigned 32-bit	Unsigned 32-bit	Unsigned 8-bit	IEEE Address
Field Name	Field control	Manufacturer code	Image type	File version	File offset	Maximum data size	Request node address

2.3.5.2 Payload Field Definitions

Image Block Request Command Field Control

Field control value is used to indicate additional optional fields that may be included in the payload of Image Block Request command. Currently, the device is required to support field control value of only 0x00; support for other field control value is optional.

Field control value 0x00 (bit 0 not set) indicates that the client is requesting a generic OTA upgrade file; hence, there is no need to include additional fields. The value of Image Type included in this case is manufacturer specific.

Field control value of 0x01 (bit 0 set) means that the client's IEEE address is included in the payload. This indicates that the client is requesting a device specific file such as security credential, log, or configuration; hence, the need to include the device's IEEE address in the image request command. The value of Image type included in this case is one of the reserved values that are assigned to each specific file type.

Table 2-18. Image Block Request Field Control Bitmask

Bits	Name
0	Request node's IEEE address Present
1–7	Reserved

Manufacturer Code

The value is that of the client device assigned to each manufacturer by ZigBee.

Image Type

The value is between 0x0000–0xffbf (manufacturer specific value range).

File Version

The file version included in the payload represents the OTA upgrade image file version that is being requested.

File Offset

The value indicates number of bytes of data offset from the beginning of the file. It essentially points to the location in the OTA upgrade image file that the client is requesting the data from. The value reflects the amount of (OTA upgrade image file) data (in bytes) that the client has received so far.

Maximum Data Size

The value indicates the largest possible length of data (in bytes) that the client can receive at once. The server respects the value and doesn't send data that is larger than the maximum data size. The server may send data that is smaller than the maximum data size value, for example, to account for source routing payload overhead if the client is multiple hops away. By having the client send both file offset and maximum data size in every command, it eliminates the burden on the server for having to remember the information for each client.

NOTE

The data size requested by the Client Device must be between `gImageDataPacketMinSize_c` and `gImageDataPacketMaxSize_c`.

(optional) Request Node Address

This is the IEEE address of the client device sending the Image Block Request command.

NOTE

This parameter is not implemented in the demonstration applications.

2.3.6 Image Block Response Command

2.3.6.1 Payload Format

Table 2-19. Image Block Response Command Payload with SUCCESS status

Octets	1	2	2	4	4	1	Variable
Data Type	Unsigned 8-bit	Unsigned 16-bit	Unsigned 16-bit	Unsigned 32-bit	Unsigned 32-bit	Unsigned 8-bit	Octet
Field Name	Success status	Manufacturer code	Image type	File version	File offset	Data size	Image data

2.3.6.2 Payload Field Definitions

Image Block Response Status

The status in the Image Block Response command may be SUCCESS, ABORT, or WAIT_FOR_DATA.

Manufacturer Code

The value is the same as the one included in Image Block/Page Request command.

Image Type

The value is the same as the one included in Image Block/Page Request command.

File Version

The file version indicates the image version that the client is required to install. The version value may be lower than the current image version on the client if the server decides to perform a downgrade. The version value may be the same as the client's current version if the server decides to perform a reinstall. However, in general, the version value should be higher than the current image version on the client to indicate an upgrade.

File Offset

The value represents the location of the data requested by the client. For most cases, the file offset value included in the (Image Block) response should be the same as the value requested by the client. For (unsolicited) Image Block responses generated as a result of Image Page Request, the file offset value increments to indicate the next data location.

Data Size

The value indicates the length of the image data (in bytes) that is being included in the command. The value may be equal or smaller than the maximum data size value requested by the client.

Image Data

The actual OTA upgrade image data with the length equals to data size value.

Current Time and Request Time

If status is WAIT_FOR_DATA, the payload then includes the server's current time and the request time that the client retries the request command. The client waits at least the request time value before trying again. In case of a sleepy device, it may choose to wait longer than the specified time to not disrupt its sleeping cycle. If the current time value is zero that means the server does not support UTC time and the client treats the request time value as offset time. If neither time value is zero, and the client supports UTC time, it treats the request time value as UTC time. If the client does not support UTC time, it calculates the offset time from the difference between the two time values. The offset indicates the minimum amount of time to wait in seconds. The UTC time indicates the actual time moment that needs to pass before the client should try again.

NOTE

These fields are ignored by this Demo Application (are set to zero)

2.3.7 Upgrade End Request Command

2.3.7.1 Payload Format

Table 2-20. Format of Upgrade End Request Command Payload

Octets	1	2	2	4
Data Type	Unsigned 8-bit	Unsigned 16-bit	Unsigned 16-bit	Unsigned 32-bit
Field Name	Status	Manufacturer code	Image type	File version

2.3.7.2 Payload Field Definitions

Upgrade End Request Command Status

The status value of the Upgrade End Request command is SUCCESS, INVALID_IMAGE, REQUIRE_MORE_IMAGE, or ABORT.

Manufacturer Code

The value is that of the client device assigned to each manufacturer by ZigBee.

Image Type

The value is between 0x0000–0xffbf (manufacturer specific value range).

File Version

The file version included in the payload represents the newly downloaded OTA upgrade image file version.

2.3.8 Upgrade End Response Command

2.3.8.1 Payload Format

Table 2-21. Table Format of Upgrade End Response Command Payload

Octets	2	2	4	4	4
Data Type	Unsigned 16-bit	Unsigned 16-bit	Unsigned 32-bit	Unsigned 32-bit	Unsigned 32-bit
Field Name	Manufacturer code	Image type	File version	Current time	Upgrade time

2.3.8.2 Payload Field Definitions

The ability to send the command with wild card values for manufacturer code, image type, and file version is useful in this case because it eliminates the need for the server having to send the command multiple times for each manufacturer as well as having to keep track of all devices' manufacturers in the network.

Manufacturer Code

Manufacturer code may be sent using wildcard value of 0xfffff to apply the command to all devices disregard of their manufacturers.

Image Type

Image type may be sent using wildcard value of 0xfffff to apply the command to all devices disregard of their manufacturers.

File Version

The file version included in the payload represents the newly downloaded OTA upgrade image file version. The value matches that included in the request. Alternatively, file version may be sent using wildcard value of 0xffffffff to apply the command to all devices regardless of their manufacturers.

Current Time and Upgrade Time

Current time and Upgrade time values are used by the client device to determine when to upgrade its running firmware image(s) with the newly downloaded one(s).

NOTE

These fields are ignored by this Demo Application (are set to zero).

2.4 OTA Programmer Image Format

The following figures represent the OTA and EEPROM image formats.

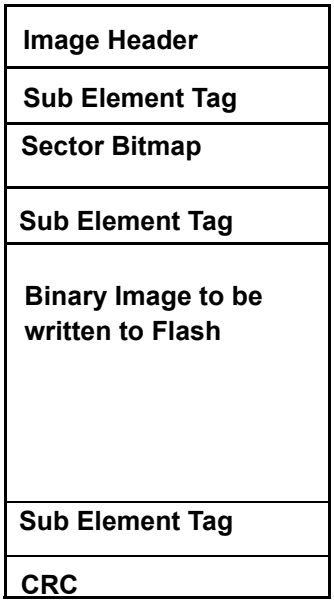

Figure 2-1. PC Application and OTA Image Format

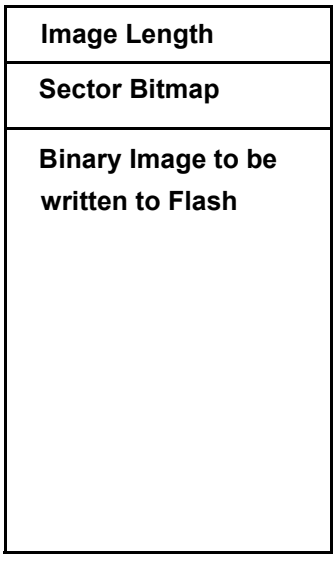

Figure 2-2. Client EEPROM Image Format

Chapter 3
Software Implementation

3.1 PC MAC OTA Programmer Application

The application used for uploading images (*.S19) to the OTAProgrammingDemoApp (Server) is integrated into the Freescale Test Tool, starting with Test Tool version 12.

Configure the *Image Type*, *File Version*, *Min* and *Max HW version*, and *Sector Bitmap* as show in Figure 3-1.

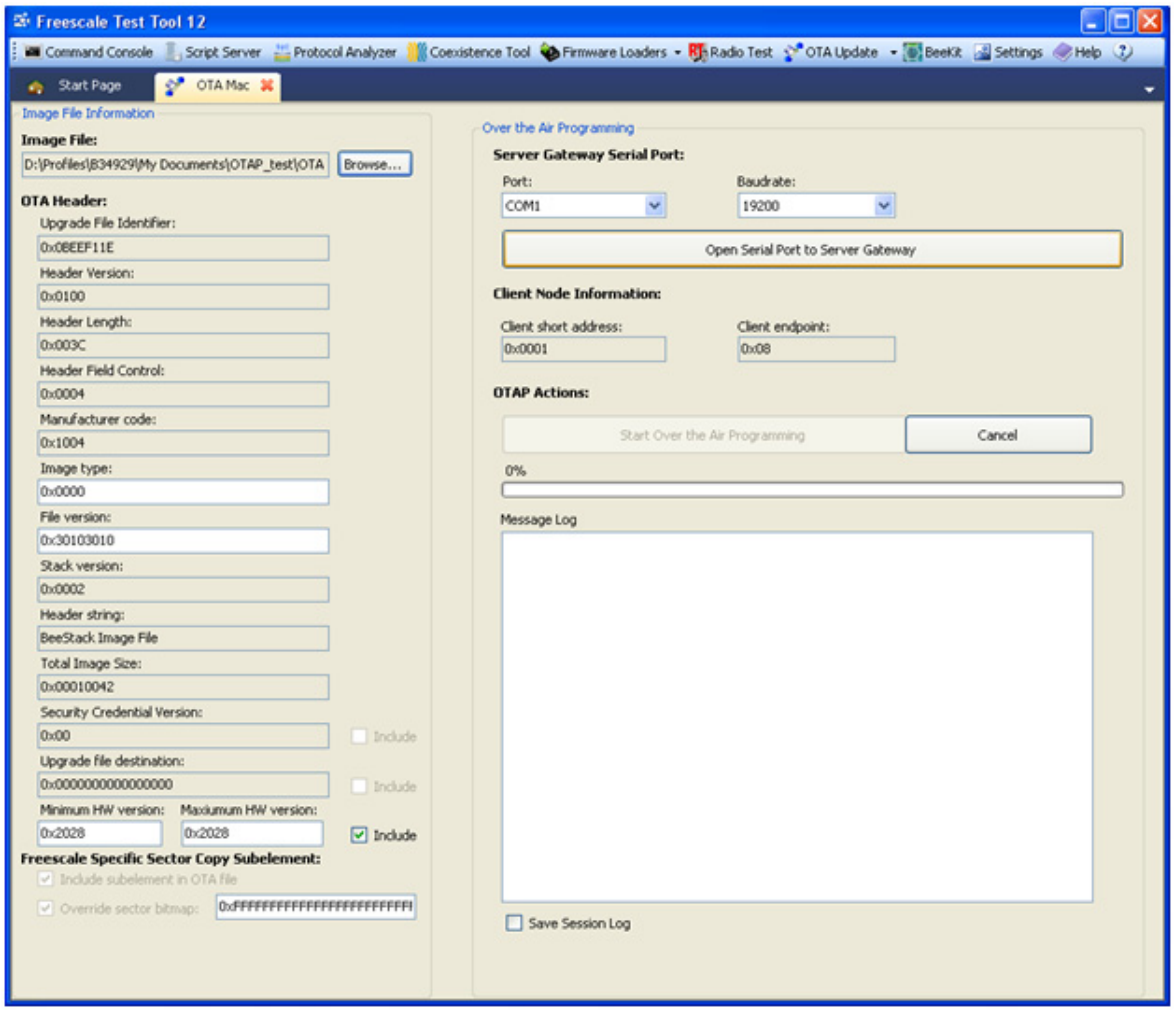

Figure 3-1. Parameter Configuration in Test Tool 12

To start downloading the new image to the Server, select the COM port on which the Coordinator board is connected, enter the baud rate (usually 19200bps) and then press the *Open Serial Port to Server Gateway.* After this the transfer can be started by pressing Start Over the Air Programming button.

NOTE

No terminal sessions should be opened on the COM port on which the Server is connected.

3.2 OTA Programmer Server (PAN Coordinator)

3.2.1 Server ZTC

To receive the image from the PC application, the ZTC module on the Server is enabled, but all SAP are disabled.

All data structures used by the OTA Programmer demonstration are declared in *ZtcInterface.h*.

The Message Types used by ZTC for OTA Programmer are:

```
mZtcOtapSupportSetModeReq_c         = 0x28,
mZtcOtapSupportStartImageReq_c      = 0x29,
mZtcOtapSupportPushImageChunkReq_c  = 0x2A,
mZtcOtapSupportCommitImageReq_c     = 0x2B,
mZtcOtapSupportCancelImageReq_c     = 0x2C,
mZtcOtapSupportImageChunkReq_c      = 0x2F,
mZtcOtapSupportQueryImageReq_c      = 0xC2,
mZtcOtapSupportQueryImageRsp_c      = 0xC3,
mZtcOtapSupportImageNotifyReq_c     = 0xC4,
```

Each one of these messages has a function associated:

```
ZtcMsgTbl(mZtcOtapSupportSetModeReq_c,         ZtcOtapSupportSetModeReqFunc)
ZtcMsgTbl(mZtcOtapSupportStartImageReq_c,      ZtcOtapSupportStartImageReqFunc)
ZtcMsgTbl(mZtcOtapSupportPushImageChunkReq_c, ZtcOtapSupportPushImageChunkReqFunc)
ZtcMsgTbl(mZtcOtapSupportCommitImageReq_c,     ZtcOtapSupportCommitImageReqFunc)
ZtcMsgTbl(mZtcOtapSupportCancelImageReq_c,     ZtcOtapSupportCancelImageReqFunc)
ZtcMsgTbl(mZtcOtapSupportQueryImageRsp_c,      ZtcOtapSupportQueryImageRspFunc)
ZtcMsgTbl(mZtcOtapSupportImageNotifyReq_c,     ZtcOtapSupportImageNotifyReqFunc)
```

The first command received from the PC app (`mZtcOtapSupportStartImageReq_c`) contains the image length without the length of the CRC Sub Element. The confirm to this command is used to notify the PC about the version of ZtcOtapSupport and the availability of an External Memory for the update process.

The second command received must be `mZtcOtapSupportSetModeReq_c`. If an External Memory is available, this command will specify if it will be used or not in the upgrade process. The AppTask will be informed when this command is received thorough a callback (`pfOtapSetModeCallback`).

The next step is to obtain a minimal set of information about the image by calling `ZtcOtapSupportQueryImageReqFunc` with the following parameters: device Id, manufacturer code, image type and file version. A value of 0xFFs has a special meaning of a wild card. The value has a ‘match all’ effect. When the response is received, the AppTask will be informed thorough a callback (`pfOtapQueryImageReqCallback`).

After this step, the Server will start requesting Image Chunks by calling `ZtcOtapSupportImageChunkReqFunc`, and specifying the image offset, the chunk length and the device ID.

The PC application will send a `mZtcOtapSupportPushImageChunkReq_c` command, and the AppTask will be informed thorough a callback (`pfOtapPushImageChunkCallback`). The chunks are handled by the AppTask.

If an error occurred during the image transfer (PC to Server device) the PC application sends `mZtcOtapSupportCancelImageReq_c`, and the transfer is stopped. An error message will be displayed by the MAC OTA Programmer PC application.

After all chunks have been transferred, the PC app sends a `mZtcOTAPSupportCommitImageReq_c` command that marks the end of the transfer.

3.2.2 Server AppTask

At startup, the function pointers used by ZTC to interact with the AppTask are initialized with the address of the functions that will process the specific event.

```
pfOtapSetModeCallback = SetOtapMode;
pfOtapQueryImageReqCallback = QueryImageReqConfirm;
pfOtapPushImageChunkCallback = ProcessImageChunk;
```

To start a PAN, press any switch on the Server's board (the LED's are turned off when finished). For more information about the association procedure and the functions used please refer to *MyWirelessApp User's Guide*.

If an upgrade process was started, the `SetOtapMode()` function will be called when a `mZtcOtapSupportSetModeReq_c` command is received. If the mode is `gUseExternalMemoryForOtaUpdate_c` then the External Memory will be initialized.

Next a `mZtcOtapSupportQueryImageReq_c` command is sent to the PC to obtain some information about the image.

a) External Memory is available

When the `mZtcOtapSupportQueryImageRsp_c` command is received, the Server will start requesting Image Chunks. Every time an image chunk is received, it will be stored into the External Memory.

After the last packet is received, the Server device will send OTA an ImageNotify. When a Client device request an image block, the block is read from the External Memory.

b) External Memory is not available

When the `mZtcOtapSupportQueryImageRsp_c` command is received, the Server will send OTA an ImageNotify.

The Image chunks are requested from the PC only when an *ImageBlockRequest* command is received OTA from a Client device.

NOTE

No processing will be done by the Server over the received image chunks.

The Client device associated receives an *ImageNotify* command. If the device is interested on the image, a *QueryNextImageRequest* command will be sent to the Server. The Server responds with a *QueryNextImageResponse*, containing the number of bytes to be transferred OTA.

At this point the Client sends an *ImageBlockRequest*, specifying the required file offset and max data size it can handle (in this case `gImageDataPacketMaxSize_c`).

All messages received by the server are processed by the *ProcessMessage()* function, which executes when a data indication is received.

To end the transfer an *UpgradeEndRequest* command must be sent by the Client. The Server will respond with an *UpgradeEndResponse command* as shown in Figure 3-2.

If an error occurs during the transfer, the Client also sends an *UpgradeEndRequest*.

3.2.3 OTA Transfer Diagram

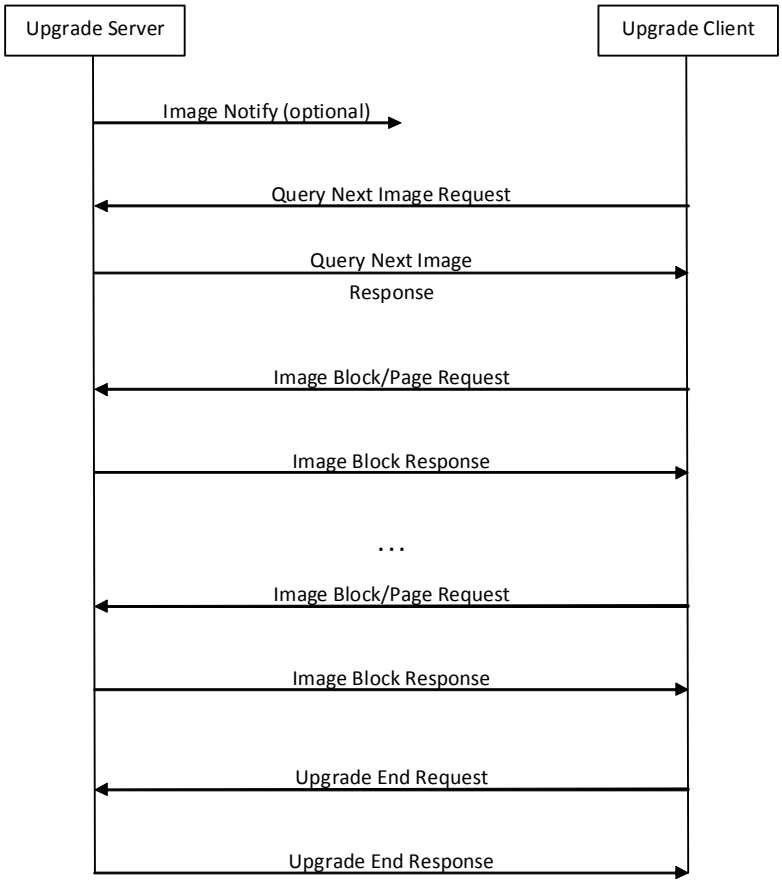

Figure 3-2. OTA Transfer Diagram

3.3 Client AppTask

Before sending/receiving any data, the device must join a PAN. This can be accomplished by pressing any switch on the Client's board. For more information about the association procedure and the functions used at this process please refer to *MyWirelessApp User's Guide*.

After this operation the Client device can query the server for a new image (by pressing any switch on the board) or it can wait a notification from the server.

```
Press any switch to query for a new image.

->Sent Query Next Image req
No image available!
```

From the *ImageNotify* and *QueryNextImageResponse* commands the Client can determine whether it is necessary to download the image. The Client device must send an *ImageBlockRequest* with the desired file offset and payload size. The requested size must between `gImageDataPacketMinSize_c` and `gImageDataPacketMaxSize_c`.

When the Client has received at least `sizeof(ImageHeader_t)` bytes, it will decode the Image Header into a RAM structure.

After the Header a variable number of Sub Elements can be received in a random order. The Client will interpret Sub Elements that have one of the following TagId:

```
gUpgradeImageTadId              0x0000
gSectorBitmapTagId              0xf000
gCRCTagId                       0xf100
```

Other Sub Elements will be ignored but they will still be requested, to compute their CRC. The CRC is computed using the `OTAP_CrcCompute()` function, over the whole image except the entire CRC Sub Element (CRC TagId, CRC TagLength and CRC field).

The Image Sub Element is saved into the External Memory using `OTAP_StartImage_NoCRC()` to initiate the process and the `OTAP_PushImageChunk_NoCRC()` to store the received chunk.

A progress bar and a percentage are displayed on the Client's terminal console using the *PrintProgress* function:

```
->Received an image Notify
->Sent Query Next Image req
OTA Transfer Progress:
100% [=========================]
```

If the computed CRC and the received CRC are the same the Client commits the image using the *OTAP_CommitImage_NoCRC* function. This operation actually writes the real image size and the sector bitmap into the EEPROM memory.

Next an *UpgradeEndRequest* must be sent to the server with the status of the upgrade. If a response (*UpgradeEndResponse*) with the success status is received from the Server, a Flag will be written to FLASH (see *OTAP_WriteNewImageFlashFlags* function) to inform the Bootloader that a new image is available, and *OTAP_ResetMcu* executes an illegal opcode to reset the MCU:

```
Transfer Successful!
Resetting MCU…
```

The Bootloader sees the flag and the image from the External Memory will be transferred to FLASH. After this step the MCU resets again and the new image is executed.

During the OTA transfer if the Server doesn't respond in *gOtaTimeout_c* three times, the process is aborted.

NOTE

The Server and the Client can store only one image in the EEPROM memory.