Atmel®

APPLICATION NOTE

AVR2058: BitCloud OTAU User Guide

Atmel MCU Wireless

Introduction

Designers and providers of embedded wireless systems continue to be challenged by the rapid evolution of the ZigBee® standard. The continuous evolution of the standard requires that these systems must be made “future-proof”, that is, the system engineers must design them to be easily upgraded systems to the next version even after the system has been deployed.

The ability to upgrade networks also depends on the individual devices having enough hardware resources to accommodate the next version of the specification. But even when these hardware resource requirements are met, there still must be a defined and interoperable mechanism for efficient over the air upgrade.

The over-the-air upgrade support in ZigBee networks is covered by the Over-the-Air Upgrade Cluster specification. The OTAU functionality relies on the use of standard ZigBee data transfer and network management facilities to transfer firmware images to any node on the network. The standard covers on air message exchange, but leaves the details of the architecture and implementation up to the vendor.

This document outlines the Atmel® architecture and implementation of over-the-air upgrade and describes how to add over-the-air upgrade support to embedded applications built on top of the Atmel BitCloud® C API. The guide also introduces PC-side tools for initiating OTAU and explores practical considerations in performing a network upgrade.

Table of Contents

1. Architecture

1.1 Architecture Building Blocks

Since ZigBee Over-the-Air Upgrade (OTAU) relies on existing ZigBee PRO services for service discovery and data transmission across the network, its building blocks fit nicely into the standard ZigBee architecture and Atmel BitCloud implementation thereof. The main components of the architecture are:

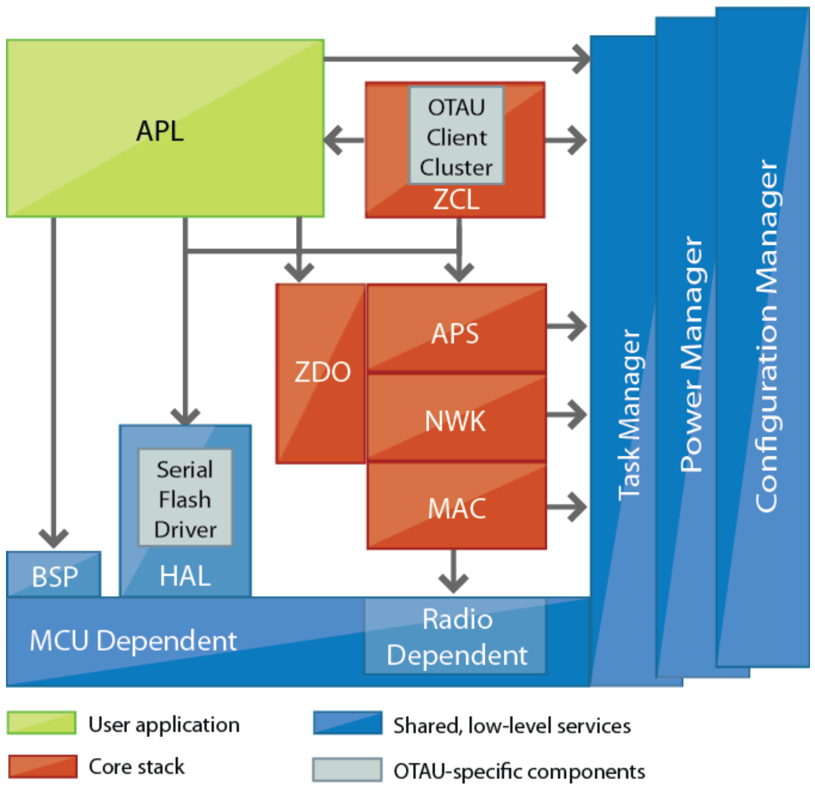
- the OTAU client, which resides on one of the end points on every upgradeable device;
- the OTAU server, which resides on whichever devices initiates the upgrade process;
- a HAL driver responsible for writing the transferred images to persistent storage on every upgradeable device;
- an OTAU-capable bootloader for transferring firmware images from persistent storage into microcontroller's flash, also present on every upgradeable device.

1.1.1 Client Side Architecture

The high-level software architecture of the OTAU client side (that is, the upgradeable device) is illustrated in Figure 1-1.

According to the OTAU specification [1], the OTAU client part of the architecture is realized as a client side of the OTAU cluster. This implies that the ZigBee Cluster Library (ZCL) framework must be present whenever OTAU is to be enabled on a device, which also restricts the use of OTAU to applications making use of ZCL.

Figure 1-1. Upgrade Client Architecture



1.1.2 Server Side Architecture

The server side of the OTAU cluster resides on what is commonly referred to as the upgrade access point (UAP). This may be a dedicated physical device which implements the server-side cluster or a multi-function in-network device which implements the OTAU service as an add-on piece of functionality. Regardless of how UAP is realized, there is always an implicit backchannel, usually in the form of another network or serial connection outside of the ZigBee network, which is used to transfer the firmware images and control commands to the UAP.

The document considers an application scenario where an in-network device is used as a permanent access point on the network through which over-the-air upgrades can be initiated at some point in a network's lifetime. In such case, UAP is a device joined to the network and supporting the server side of the OTAU cluster in addition to functionality of a common network device.

In order to start and control upgrade process an in-network UAP is connected serially to a PC, and the Bootloader PC tool [5] or a similar custom utility is used.

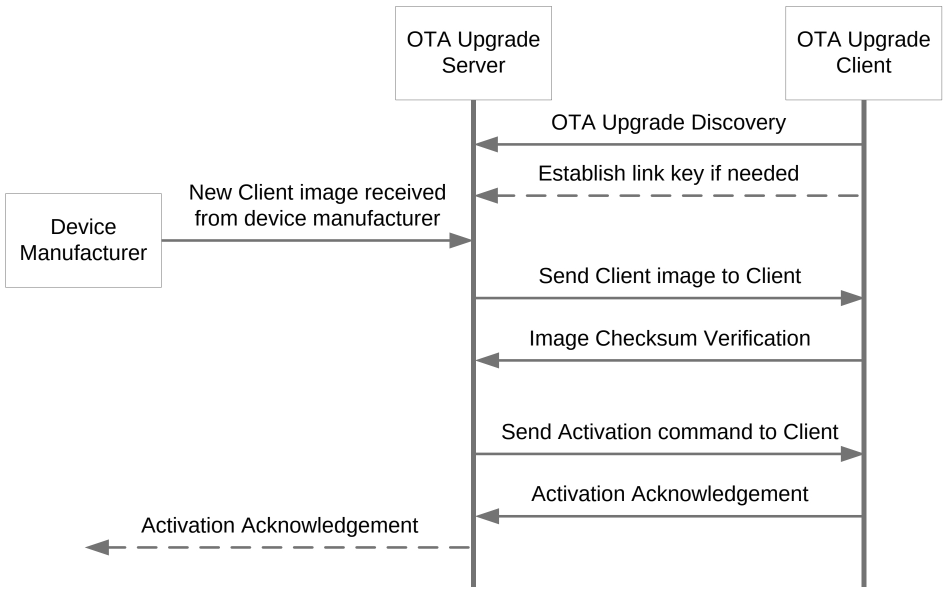
1.2 Basic Protocol and Control Flow 1` cx

Once UAP appears on the network (assuming it possesses the necessary security material to join it), it must be discovered by the OTAU clients (upgradeable nodes), that is, by those devices that implement the client side of the OTAU cluster. These client nodes issue periodic service discovery commands to discover the OTAU service cluster. In application scenarios where UAP is always present, this discovery will happen once when the client is powered on.

Before clients can proceed with any OTAU specific actions, they must secure the link to the server. The security settings applied to this link are the same as the security requirements for the specific ZigBee application profile of the network being upgraded. Thus for profiles that require standard link security, the client must negotiate with the trust center for an application link key with the OTAU server. OEM and private profiles can use their own security settings including forgoing security altogether.

Once the link is secured, the clients can begin querying the server for the next image. If the server indicates that a new image is available, then a client starts requesting individual firmware image blocks or image pages consisting of multiple blocks from the server, eventually completing the download. When the download is complete, the server can tell the client when to actually begin running the new firmware image. The whole sequence of steps is illustrated in Figure 1-2.

Figure 1-2. Network Protocol for Transferring Firmware Images


OTA Upgrade Server
OTA Upgrade Client
OTA Upgrade Discovery
Establish link key if needed
New Client image received from device manufacturer
Device Manufacturer
Send Client image to Client
Image Checksum Verification
Send Activation command to Client
Activation Acknowledgement
Activation Acknowledgement

If a client loses connection to the server, while it is loading an image, it will try to restore connection and continue downloading the image. The OTAU client cluster will try to get response from the server. If the server replies with information about the image of the same version as the image which downloading has been broken, the downloading will continue.

1.2.2 Image Page/block Request Modes

There are two possible modes for an OTAU client to load an application image from the OTAU server: using image block requests or image page request. In the first case an OTAU client sends a separate data request for each individual image block and confirms back reception of each block.

While a block fits into a single data frame, a page is a greater piece of an image. If the client uses image page requests it does not send acknowledgements to the server until all blocks for a given page are sent by the server. If, while receiving a page, the client discovers that some blocks are lost, it marks missed pieces (which may consist of several blocks), using a special buffer, and retrieves the missing blocks via image block requests after the transmission of the current page finishes.

Using image page requests speeds up image upload process, because the client does not have to acknowledge delivery of each individual block of the image. This mode is used by default. The user can switch to using image block requests or configure some aspects of image page requests usage through Configuration Server parameters (see Section 2.3.3).

1.3 Application/OTAU Interaction

An application that wishes to use OTAU functionality must initialize the OTAU client service to run alongside it and to participate in the interactions with the server as shown in Figure 1-2. This is done with a simple call to `ZCL_StartOtauService()` described in Section 2.4.1.

Although the OTAU client service is completely self-contained, that is, no further action needs to be taken by the application during the upgrade process; the application may choose to get OTAU-specific indications from the cluster. These include indications for a new available image, download completion, and various types of error conditions. To receive indications the application simply registers a callback with the OTAU cluster when the OTAU client service is initialized.

Once the application initiates the OTAU client and server, both will run in parallel. The application will continue to execute as usual, but it may experience degradation in performance consistent with the amount of traffic generated by the OTAU upgrade. Well-behaved applications should not see any other adverse effects besides proportionally decreased data throughput. The OTAU client cluster runs in the ZCL task handler and at the priority of ZCL layer, so the OTAU tasks will have slightly higher priority than the application tasks.

1.4 Embedded Bootloader and External Image Store

Due to unpredictable and dynamic nature of network links in multi-hop wireless networks like ZigBee, the firmware image can only be transferred to the client using the best effort facilities for data transfer. At any given point in time the client which has downloaded only a part of the image may become disconnected from the network. Since in the general case it is impossible to avoid such a scenario, it helps to ensure that the entire image is received before any part of it irreversibly overwrites any part of the currently running image.

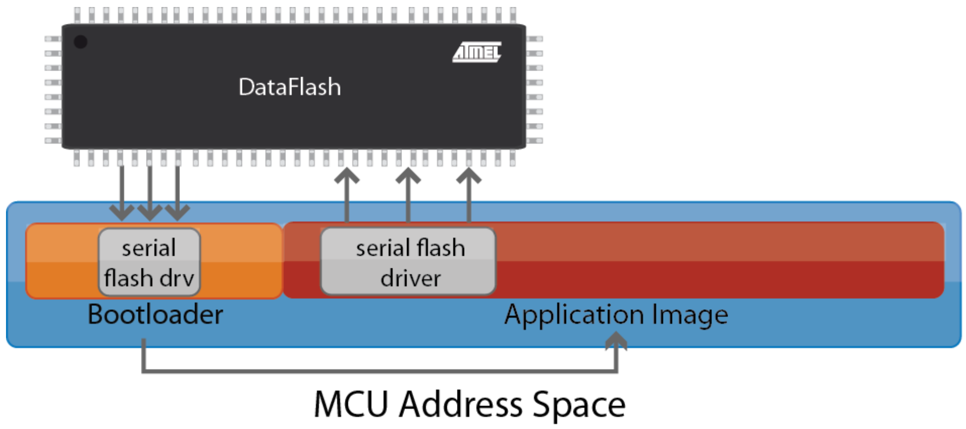
This requires an architecture where whole and partially downloaded firmware images can be stored in a dedicated, nonvolatile firmware image store decoupled from the flash memory holding the currently executing application image. The simplest version of such a system is the one that stores firmware images in a serial flash connected to an upgradable client node. Once the OTAU client cluster receives parts of the image it verifies its integrity (security is already provided by the profile security applied to all in-network communications) and writes it to serial flash. Once the whole image is received, it can be swapped from serial flash into microcontroller's internal flash memory.

Embedded bootloader typically refers to some code that runs on the embedded device after the device is reset, but before the control is passed to the application image. One piece of functionality shared by most bootloaders is the ability to modify parts or the entire application image residing in other parts of flash. Thus bootloaders are commonly used to upgrade embedded devices over a serial connection.

In context of OTAU, the embedded bootloader is extended to also cover interactions with the serial port as well as with the external firmware image store. Thus an OTAU-enabled bootloader may receive images either over serial input or from an external non-volatile memory. Figure 1-3 illustrates this bootloader architecture.

The ability of both the OTAU client cluster and the OTAU bootloader to interact with the external serial flash is critical, so both must be made aware of the external storage interface. Note that since bootloader and application image (including the OTAU client cluster) run in different address spaces, parts of the interface driver must be duplicated. Section 2.2 explores the serial flash driver in more detail.

Figure 1-3. Embedded Bootloader Architecture



1.4.1 Secure Bootloading

The upgrade image can optionally be encrypted to prevent storing of raw firmware image on external flash that can be easily accessed. Using encrypted image also improves protection for confidentiality and integrity.

Secure image bootloading is out of scope of ZigBee OTAU specification [1] that requires only standard encryption of individual over-the-air frames but allows overall image encryption as vendor-specific implementation. Atmel provides an example implementation of such secure bootloading. It can be used as is with just vendor-specific configuration for security material or can be used as a reference when implementing own algorithms.

Atmel embedded bootloader with security support has capability to read the metadata in the upgrade image to know if the image is encrypted. AES-128 CBC (cipher block chaining) is used to encrypt/decrypt the image.

To ensure image integrity, message integrity code (MIC), calculated for the encrypted content along with the file header, can optionally be added to the OTAU file. The message integrity can either be AES-128 CBC-MAC or 8-bit CRC. The integrity check can optionally be encrypted but it needs to use the same encryption key and initialization vector (IV) as the image encryption. The difference between the image encryption and integrity encryption is the integrity encryption derives the code both from the meta-data and from the encrypted payload. The advantage of encrypting the integrity code is that, key/IV mismatch at decryption side can be found out before even honoring the actual image. If the integrity check does not match, be it due to wrong key/IV or due to content corruption in the file, the OTAU image can be safely ignored.

The header part of OTAU file cluster carries the information of whether the image is encrypted, whether the file carries message integrity check and the type of it.

Additional details on implementation of secure bootloading are described in Section 2.5.

2. Implementation

This chapter describes how OTAU is implemented in Atmel BitCloud and provides instructions on making devices participate in OTAU as OTAU clients or OTAU servers. BitCloud packages provide reference applications which demonstrate OTAU functionality and can be used as templates for adding OTAU support to other reference applications or to the user's own custom applications.

To enable OTAU the user shall consider:

- Hardware setup
- OTAU parameters configuration
- Use of OTAU cluster API
- Use of PC tools to initiate and control OTAU process

The following sections address all these considerations in more detail.

The OTAU has the classical client-server architecture. Both a client and a server shall support the OTAU cluster, but in different modes: an OTAU client shall register the OTAU cluster as an output cluster, while a server shall support an input OTAU cluster to be able to process incoming requests. The network may contain multiple servers, although typically there is only one OTAU server in the network. A server can be either a common in-network device with OTAU cluster support or a device temporally added to the network, which in the reference implementation is a Runner device.

The Serial Bootloader software package [5] shall also be downloaded. The package contains embedded bootloader's firmware images and sources, and the Bootloader PC tool, which is used to control the OTAU server device. All OTAU clients should be programmed with the embedded bootloader for OTAU firmware, and the application should be loader via the serial connection with the help of the Bootloader PC tool. For details see [5].

2.1 Supported Platforms

BitCloud is the Atmel professional-grade implementation of ZigBee PRO standard for wireless monitoring and control [2]. OTAU support with regard to reference applications can be found in [3]. The list of OTAU-enabled hardware platforms is shown in Table 2-1.

Table 2-1. Supported Packages

Package name	Microcontroller	Radio frequency transceiver	Supported external Flash
BitCloud SDK for megaRF	ATmega256RFR2 ATmega2564RFR2	Built-in	AT25DF041A
BitCloud SDK for SAMR21	ATSAMR21G18A ATSAMR21E18A SAMR21B18_MZ210PA_MODULE SAMR21G18_MR210UA_MODULE ATSAMR21E19A	Built-in	M25P40VMN6PB M25P40VMN6PB MX25L2006E AT45DB041E MX25L4006E(OnChip Flash)

2.2 Hardware Setup

Since the embedded bootloader and the hardware abstraction layer (HAL) of the Atmel BitCloud stack are provided in source code, the user may extend the range of supported external flash devices by modifying the driver to interface it with any unsupported chipset. The driver maintains a consistent API for the OTAU cluster to rely upon, which makes it possible to integrate core stack libraries with custom drivers. Of course, the serial flash driver must then also be replicated in the bootloader section.

Note that to switch from one external Flash device to another, the user shall recompile the embedded bootloader and the application itself. In the application the type of external Flash is specified in the `configuration.h` file. For example if Atmel AT25DF041A is used, it will contain with the following line:

```
#define EXTERNAL_MEMORY AT25DF041A
```

2.3 OTAU Configuration

2.3.1 Enabling OTAU

In BitCloud reference applications OTAU support is enabled via `APP_USE_OTAU` define in applications' `configuration.h` file. If it is set to `1` then OTAU cluster will be registered on the application endpoint, OTAU cluster code in ZCL component and external flash driver in HAL will be compiled together with the application.

Note that for SAMR21 platforms additionally IDE project configuration shall have an OTAU support.

The resulting application image includes all of the components illustrated in Figure 1-1. Detailed description on how application code controls OTAU is given in Section 2.4.

2.3.2 Setting OTAU Parameters

The `configuration.h` file of the application includes a set of parameters enabled when `APP_USE_OTAU` equals `1` and disabled otherwise:

- ConfigServer parameters (see Section 2.3.3), mainly configuring OTAU-cluster functionality
- Parameters related to secure booloading (see Section 2.3.4)
- Application-specific parameters:
  - `APP_USE_OTAU` defines whether OTAU is enabled or not
  - `APP_USE_FAKE_OFD_DRIVER`: when set to `1`, the stack will substitute the real Flash driver with the fake one that implements all driver operations as stub functions that do not store received image anywhere. This feature may be useful to test OTAU operation on evaluation boards without external flash present.
  - `APP_USE_ISD_CONSOLE_TUNNELING` that indicates support for simultaneous usage of the same serial interface for receiving commands from command console and commands exchanged by the ISD driver and the bootloader PC tool. This parameter is valid for the OTAU server.
  - `APP_SUPPORT_OTAU_PAGE_REQUEST` which enables use of page requests in OTAU cluster
  - `APP_SUPPORT_OTAU_RECOVERY`: when set to `1`, OTAU will recover the upgrade, after power failure, from where it was interuppted.
  - `EXTERNAL_MEMORY` which defines what external Flash device should be used
  - `OTAU_CLIENT/OTAU_SERVER` which specifies whether the device operates as an OTAU client or an OTAU server. The user shall uncomment the required option and comment out the other
  - For ATmega256RFR2 platform, the following line needs to be commented in `configuration.h` of the application when using bootloader:
    `#define PDS_NO_BOOTLOADER_SUPPORT`

2.3.3 OTAU ConfigServer Parameters

Parameters of the Configuration Server component (ConfigServer) of Atmel BitCloud that customize the OTAU operation are presented in Table 2-2. Note that parameters are applied either on the client or the server side. The role of a device is determined by whether `OTAU_CLIENT` or `OTAU_SERVER` in uncommented in the configuration.h file of the application.

Table 2-2. ConfigServer Parameters for OTAU

Parameter	Used on	Description
`CS_ZCL_OTAU_DISCOVERED_SERVER_AMOUNT`	Client	The maximum number of OTAU servers in the network whose responses the client can process
`CS_ZCL_OTAU_CLIENT_SESSION_AMOUNT`	Server	The maximum number of clients served by the server simultaneously. If equals `1`, the server will upgrade one client at a time
`CS_ZCL_OTAU_SERVER_DISCOVERY_PERIOD`	Client	The duration between two attempts to find an OTAU server
`CS_ZCL_OTAU_DEFAULT_UPGRADE_SERVER_IEEE_ADDRESS`	Client	The default OTAU server address. If the user specifies a valid extended address of a device in the network, the client will send server discovery requests to this particular address. If a broadcast address is specified (`0x0000000000000000` or `0xFFFFFFFFFFFFFFFF`), the client will broadcast server discovery requests.
`CS_ZCL_OTAU_IMAGE_PAGE_REQUEST_ENABLE`	Client	Specifies whether to use image page requests (if set to `1`) or image block requests (if set to `0`). Note that `APP_SUPPORT_OTAU_PAGE_REQUEST` shall be set to `1` for this parameter to work.
`CS_ZCL_OTAU_IMAGE_PAGE_REQUEST_RESPONSE_SPACING`	Client	The minimum duration between sending two blocks. The server receives this value from the client. According to the OTAU specification, the minimum value should be 200ms.
`CS_ZCL_OTAU_IMAGE_PAGE_REQUEST_PAGE_SIZE`	Client	The number of bytes transferred from the server to the client for a single image page request.
`CS_ZCL_OTAU_QUERY_INTERVAL`	Client	The interval in milliseconds between two successful attempts to query the server.
`CS_ZCL_OTAU_MAX_RETRY_COUNT`	Client	Maximum numbers of retries for commands (OTAU cluster, ZDO and APS) used for OTAU process. Retry happens at ZCL level in case of failure to receive a successful response.

To find out more details about ConfigServer parameters refer to [4].

As it can be observed from Table 2-2, the network can include several OTAU servers, although the user rarely needs more than one server. A client device periodically attempts to discover upgrade servers by sending service discovery requests either to all devices in the network (broadcast) or to a particular address if it is predefined. An upgrade server can also specify the number of simultaneous client sessions, which the server serves in parallel.

2.3.4 Secure Bootloader Parameters

Table 2-3. Security Parameters

Configuration parameter	Application/ Bootloader	Possible values	Description
`USE_IMAGE_SECURITY`	Both	0	client supports plain image
		1	client supports secured image
`IMAGE_KEY`	Application	128-bit value	Key value that should be used for image decryption
`IMAGE_IV`	Application	128-bit value	Initialization vector that should be used for AES decryption

More information on operation of secure bootloading is given in Section 2.5.

2.4 Application Operation with OTAU Support

If a correct build configuration is chosen and parameters are specified, the application can enable OTAU functionality through several simple steps described in Section 2.4.2. After the OTAU service has been launched, the application can control the OTAU operation by processing notifications about various events in the callback function specified at OTAU service start and by calling other OTAU cluster API functions described in Section 2.4.1.

A custom application can safely reuse code from reference applications as described in Section 2.4.3.

2.4.1 OTAU Cluster API Overview

Table 2-4 lists OTAU cluster API functions. More details, arguments specifications, etc. can be found in [4].

Table 2-4. OTAU Cluster API Functions

Function	Valid for	Description
`ZCL_GetOtauClientCluster()`	Client	Retrieves the OTAU cluster information on an OTAU client, which should be passed to the endpoint registration function while registering the endpoint for the OTAU service
`ZCL_GetOtauServerCluster()`	Server	Retrieves the OTAU cluster information on an OTAU server, which should be passed to the endpoint registration function while registering the endpoint for the OTAU service
`ZCL_StartOtauService()`	Client and server	Starts the OTAU service. The function shall be called after a network start. See Section 2.4.3.5 for use example
`ZCL_StopOtauService()`	Server	Stops the OTAU service; is implemented for a server only
`zclIsOtauBusy()`	Client and server	Checks whether the OTAU cluster is busy or not
`ZCL_UnsolicitedUpgradeEndResp()`	Server	Sends an upgrade end response to a client specifying the duration to wait before swapping firmware images
`ZCL_ImageNotifyReq()`	Server	Sends an image notify command to a client or set of clients indicating them, the availability of new image to upgrade

2.4.1.1 Processing OTAU Notifications

The OTAU cluster informs the application of various events that occur during its operation via the callback function specified in the `ZCL_StartOtauService()` call (see Section 2.4.3.5). A callback function has the following signature:

```
typedef void (* ZCL_OtauStatInd_t)(ZCL_OtauAction_t action);
```

The action variable of the `ZCL_OtauAction_t` enumeration type indicates the type of the event. The enumeration is defined in the `zclOTAUCluster.h` file located in the `\BitCloud\Components\ZCL\include` directory.

The application on a server is not obliged to process any events, while a client shall process at least the `OTAU_DEVICE_SHALL_CHANGE_IMAGE` notification, which is called when the time received with the upgrade end response from the server elapses. Thus this notification indicates that the device is ready to swap image, and the application shall respond by invoking hardware reset. Note that the device is not reset automatically after loading the image. This allows the application to make necessary preparations before switching the application image.

2.4.2 Running the OTAU Service on a Client/Server

The OTAU cluster is implemented as a service, such that the user would only need to properly configure the service and start it with a single function call. Since the process is almost identical for both a client and a server, a general procedure is given below highlighting differences between the server and the client side whenever necessary.

To enable OTAU functionality the user shall do the following:

1. Specify whether the device is a client or a server by enabling `OTAU_CLIENT` or `OTAU_SERVER` in the `configuration.h` file of the application.
2. Get the OTAU cluster structure of the `ZCL_OtauCluster_t` type by calling:
    a. `ZCL_GetOtauClientCluster()` for a client.
    b. `ZCL_GetOtauServerCluster()` for a server.
3. Configure and register the endpoint where the OTAU service will reside via the `ZCL_RegisterEndpoint()` function. Note the differences in endpoint configuration for a client and a server:
    a. For a client specify the OTAU cluster in the out clusters list and assign the pointer to the structure received in Step 2 to the clientCluster field of the endpoint.
    b. For a server specify the OTAU cluster in the in clusters list and assign the pointer to the structure received in Step 2 to the serverCluster field of the endpoint.
4. After a network start, run the OTAU service by calling:

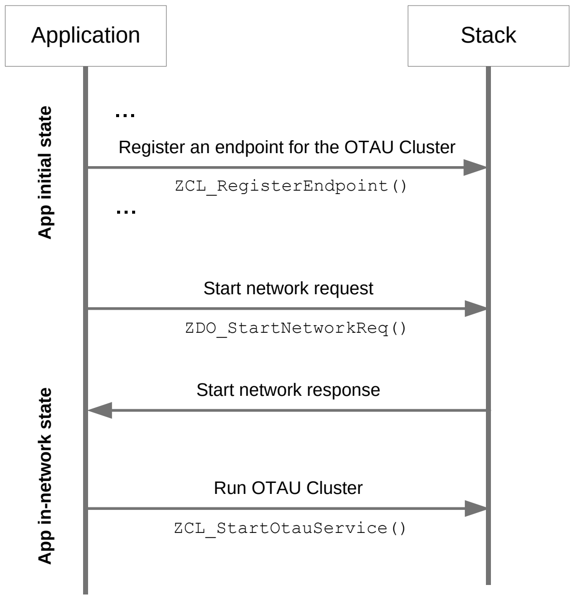
    ```
    ZCL_StartOtauService(&otauInitParams, otauClusterIndication);
    ```

    a. In `otauInitParams` set clusterSide to `ZCL_CLIENT_CLUSTER_TYPE` for a client and `ZCL_SERVER_CLUSTER_TYPE` for a server.
    b. `otauClusterIndication` is a callback function executed upon certain events related to the OTAU operation.

The first three steps are performed before the device entered a network. `ZCL_StartOtauService()` is called after the network start. Interaction between the application and the stack is illustrated in Figure 2-1. For source code examples refer to Section 2.4.3.

After the OTAU service has been launched, no more actions are required for a server. However, a client must process the `OTAU_DEVICE_SHALL_CHANGE_IMAGE` notification, which indicates that the device is ready to swap images. Refer to Section 2.4.1.1 for details.

Figure 2-1. Application/stack Interaction to Support the OTAU Cluster



2.4.3 Example Code for the OTAU Service Usage

The user can use reference implementation of OTAU support (for example, in WSNDemo) as a template. In the WSNDemo application invocation of the OTAU cluster API is localized in the `WSNZclManager.c` file. Parameters for API functions are configured in the `appZclManagerInit()` function, which is called from the application state machine during application initialization. The OTAU service is started by the `runOtauService()` function called from the application state machine after the network start.

The following code examples illustrate the procedure described in Section 2.4.2.

2.4.3.1 Define Variables and Constants

Variables holding parameters for API functions are defined in the file scope as follows:

```
static ZCL_Cluster_t otauCluster;
static ClusterId_t otauClusterId = OTAU_CLUSTER_ID;
static ZCL_OtauInitParams_t otauInitParams;
static ZCL_DeviceEndpoint_t otauClusterEndpoint;
```

In addition, define constants for the number of in and out clusters, which differ for a client and a server:

```
#if defined(OTAU_CLIENT)
#define OUT_CLUSTERS_COUNT 1
#define IN_CLUSTERS_COUNT 0
#elif defined(OTAU_SERVER)
#define OUT_CLUSTERS_COUNT 0
#define IN_CLUSTERS_COUNT 1
#endif
```

2.4.3.2 Get the OTAU Cluster

The following code gets the OTAU cluster information for both a client and a server:

```
#if defined(OTAU_CLIENT)
  otauCluster = ZCL_GetOtauClientCluster();
#elif defined(OTAU_SERVER)
  otauCluster = ZCL_GetOtauServerCluster();
#endif
```

2.4.3.3 Register the Endpoint

The OTAU service requires an endpoint, which shall be registered by a call to the `ZCL_RegisterEndpoint()` function, rather than via the APS component function used to register endpoints for data transfer. The following code prepares and registers the endpoint:

```
  otauClusterEndpoint.simpleDescriptor.endpoint = APP_OTAU_CLUSTER_ENDPOINT;
  otauClusterEndpoint.simpleDescriptor.AppProfileId = PROFILE_ID_SMART_ENERGY;
  otauClusterEndpoint.simpleDescriptor.AppDeviceId = WSNDEMO_DEVICE_ID;
  otauClusterEndpoint.simpleDescriptor.AppInClustersCount = IN_CLUSTERS_COUNT;
  otauClusterEndpoint.simpleDescriptor.AppOutClustersCount = OUT_CLUSTERS_COUNT;
#if defined(OTAU_CLIENT)
  otauClusterEndpoint.simpleDescriptor.AppInClustersList = NULL;
  otauClusterEndpoint.simpleDescriptor.AppOutClustersList = &otauClusterId;
  otauClusterEndpoint.serverCluster = NULL;
  otauClusterEndpoint.clientCluster = &otauCluster;
#elif defined(OTAU_SERVER)
  otauClusterEndpoint.simpleDescriptor.AppInClustersList = &otauClusterId;
  otauClusterEndpoint.simpleDescriptor.AppOutClustersList = NULL;
  otauClusterEndpoint.serverCluster = &otauCluster;
  otauClusterEndpoint.clientCluster = NULL;
#endif
  ZCL_RegisterEndpoint(&otauClusterEndpoint);
```

Again, the code above is valid for both a client and a server. A custom application may change the value assigned to AppDeviceId to whatever value needed. It is also assumed that the application defines all constants with the APP prefix. Here it is the endpoint identifier, which can take any value from 1 to 240 not occupied by other endpoints.

2.4.3.4 Prepare Parameters for OTAU Initialization

Parameters for the `ZCL_StartOtauService()` function, which is called after a network start to run the OTAU service, may be configured immediately after the endpoint registration.

```
#if defined(OTAU_CLIENT)
  otauInitParams.clusterSide = ZCL_CLIENT_CLUSTER_TYPE;
#elif defined(OTAU_SERVER)
  otauInitParams.clusterSide = ZCL_SERVER_CLUSTER_TYPE;
#endif
  otauInitParams.firmwareVersion.memAlloc = APP_OTAU_SOFTWARE_VERSION;
  otauInitParams.otauEndpoint = APP_OTAU_CLUSTER_ENDPOINT;
  otauInitParams.profileId = PROFILE_ID_SMART_ENERGY;
```

Note: `otauEndpoint` shall be set to the identifier of the endpoint registered for the OTAU service.

2.4.3.5 Run the OTAU Service

As a second argument `ZCL_StartOtauService()` requires a callback function which is called upon OTAU-related events. The simplest callback implementation may look like this:

```
static void otauClusterIndication(ZCL_OtauAction_t action)
{
  if (OTAU_DEVICE_SHALL_CHANGE_IMAGE == action)
  {
    // Device has finished uploading image and can be reset. The
    // application can perform additional actions here before the
    // reset.
    HAL_WarmReset();
  }
}
```

Provided the callback function is declared as stated above, the following line starts the OTAU service:

```
  ZCL_StartOtauService(&otauInitParams, otauClusterIndication);
```

2.5 Secure Bootloader Operation

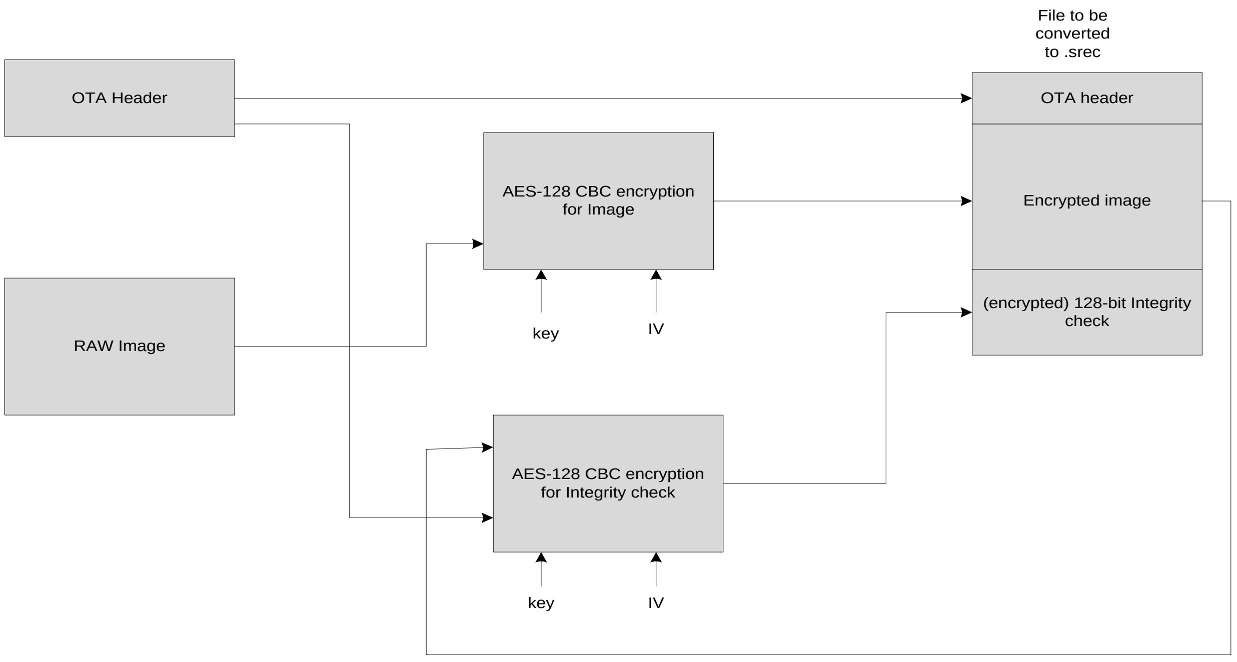
2.5.1 Server Side (encryption)

Figure 2-2 illustrates the procedure required to form an encrypted image that can be sent over the air. This procedure requires following steps:

1. Extract the raw payload from .srec container.
2. Encrypt (AES128 CBC) the raw content with the configured key/IV.
3. Add OTAU cluster file header with appropriate parameters on top of the encrypted payload as required by OTAU specification. Note Atmel-specific use of Image Type as described in Section 2.5.1.2.
4. Add encrypted(optional) integrity code(optional) if configured.
5. Update the appropriate fields in the OTAU header to reflect step 4.

Note: BitCloud SDK provides an encryption tool (see Section 2.5.1.1) that automates the steps described above.

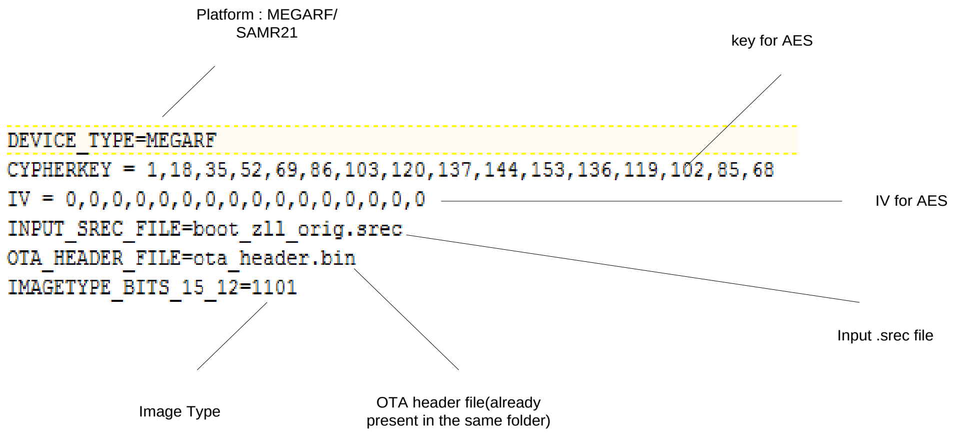
Figure 2-2. Image Encryption Flow


File to be converted to .srec
OTA Header
OTA header
AES-128 CBC encryption for Image
Encrypted image
RAW Image
key
IV
(encrypted) 128-bit Integrity check
AES-128 CBC encryption for Integrity check
key
IV

2.5.1.1 Using the Encryption Tool

The encryption tool is part of the BitCloud SDK package. The tool supports both MEGARF and SAMR21 platform outputs. To generate the encrypted files, the configuration parameters need to be set in "`boot_ldr.cfg`". The following are the fields present in "`boot_ldr.cfg`":

Figure 2-3. Configuration Parameters of Encryption Tool


Platform : MEGARF/ SAMR21
key for AES
IV for AES
Input .srec file
Image Type
OTA header file(already present in the same folder)

```
DEVICE_TYPE=MEGARF
CYPHERKEY = 1,18,35,52,69,86,103,120,137,144,153,136,119,102,85,68
IV = 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
INPUT_SREC_FILE=boot_zll_orig.srec
OTA_HEADER_FILE=ota_header.bin
IMAGETYPE_BITS_15_12=1101
```

After configuring the parameters, run the application to generate the .srec file. The command to run the application is:
`encrypt_bc.exe boot_ldr.cfg`

2.5.1.2 Usage of "Image Type" Field in OTAU Header

"Image Type" field in the OTAU file header is used to reflect upgrade file's security credentials. Once the image is honored, the same value is used in OTA upgrade request frames from the client. The following table provides the interpretation of bits of "image type":

Table 2-5. "Image Type" Bit Interpretation

Bit	Value	Description
15	0	Firmware image is plain, un-encrypted
	1	Firmware image is AES-128 CBC encrypted
14	0	No integrity code present
	1	Integrity code is appended at the end of the payload
13	0	Integrity code is unencrypted(if bit 14 is '1')
	1	Integrity code is encrypted(if bit 14 is '1')
12	0	Integrity code used is an 8-bit CRC (if bit 14 is '1'). The rest of 15 bytes can be anything
	1	Integrity code used is a 128-bit CBC-MAC(if bit 14 is '1')
Other bits/values until 0xffbf		Reserved

Based on the above table, the following are the valid combinations of "image type" values:

Table 2-6. Supported Values of "Image Type"

Value	Description
0x0000	Firmware image is plain, un-encrypted
0x8000	Image is encrypted. No integrity code present.
0xC000	Image is encrypted. Plain 8-bit CRC(aligned to 128-bits) is present at the end of the file
0xD000	Image is encrypted. Plain 128-bit CBC MAC is present at the end of the file
0xE000	Image is encrypted. AES-128 Encrypted 8-bit CRC(aligned to 128-bits) is present at the end of the file
0xF000	Image is encrypted. AES-128 Encrypted 128-bit CBC MAC is present at the end of the file

2.5.2 Client Side (decryption)

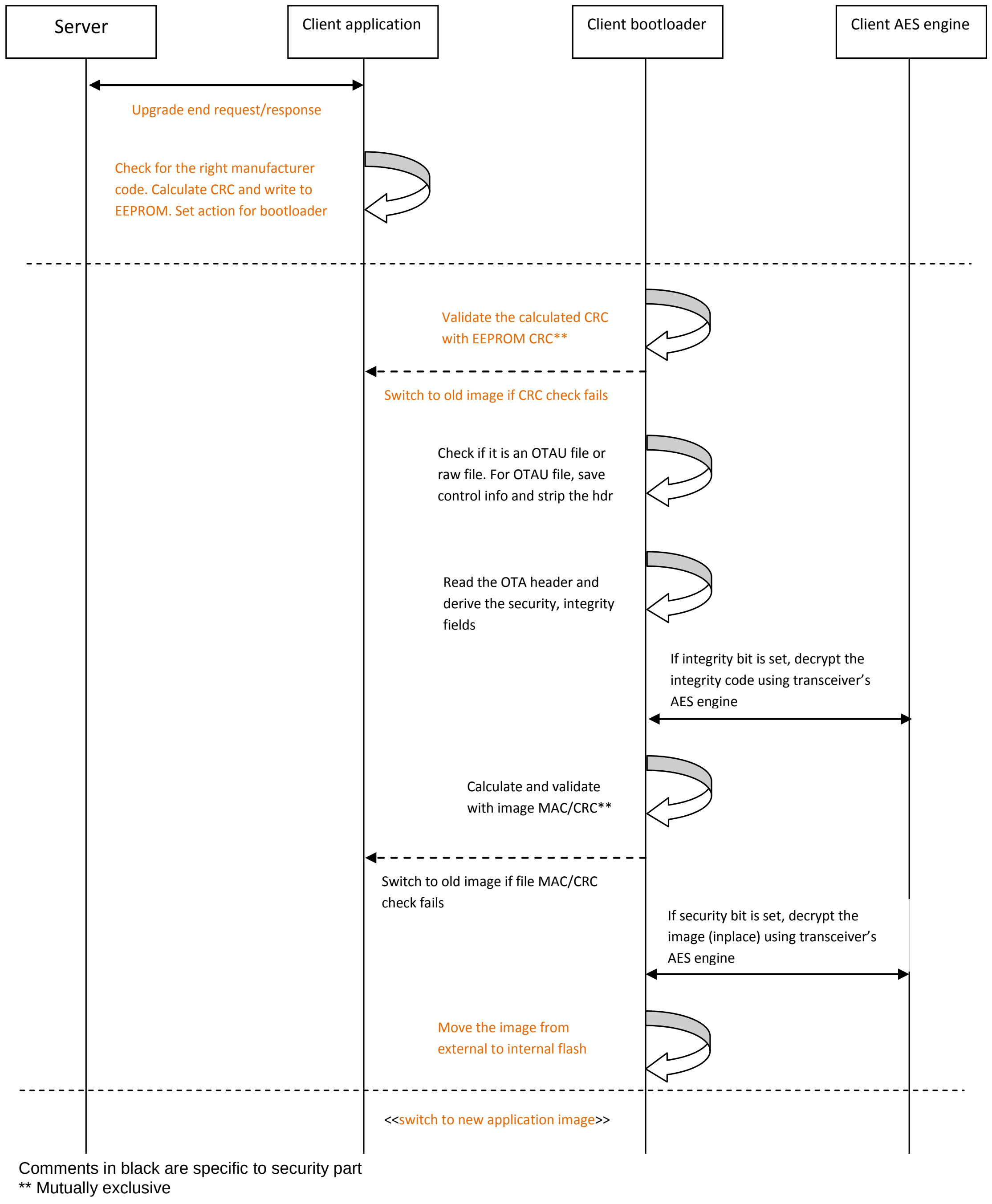
Secure bootloader shall be present on a device [5] if handling of encrypted firmware images is required. Additionally application shall be compiled with `USE_IMAGE_SECURITY` set to `1` in application `configuration.h` file.

If application wants to configure decryption material it can define it via `IMAGE_KEY` and `IMAGE_IV` parameters (see Section 2.3.4) and then call `ZCL_ConfigureOtauImageKey()` function in the application initialization code. This function reads `IMAGE_KEY` and `IMAGE_IV` macros configured by the application and shares for bootloader's use via EEPROM area (see Section 2.5.4).

Figure 2-4 illustrates the decryption sequence diagram that is performed on the client side by secure bootloader as follows:

1. Check if application has configured decryption key and initialization vector. If not, use the default key/IV.
2. Check for OTAU file cluster identifier to detect if it is an OTAU cluster file.
3. Read integrity code (if present) from the file, if OTAU file header notifies its presence.
4. Decrypt the integrity code (if present and encrypted).
5. If integrity check passes, decrypt the upgrade image and use it.

Figure 2-4. Decryption Flow


Server
Client application
Client bootloader
Client AES engine
Upgrade end request/response
Check for the right manufacturer code. Calculate CRC and write to EEPROM. Set action for bootloader
Validate the calculated CRC with EEPROM CRC**
Switch to old image if CRC check fails
Check if it is an OTAU file or raw file. For OTAU file, save control info and strip the hdr
Read the OTA header and derive the security, integrity fields
If integrity bit is set, decrypt the integrity code using transceiver's AES engine
Calculate and validate with image MAC/CRC**
Switch to old image if file MAC/CRC check fails
If security bit is set, decrypt the image (inplace) using transceiver's AES engine
Move the image from external to internal flash
<<switch to new application image>>
Comments in black are specific to security part
** Mutually exclusive

2.5.3 Use of Secured and Unsecured Image

If `USE_IMAGE_SECURITY` is enabled in the bootloader, the bootloader can upgrade only encrypted file sent in OTAU cluster file format. Plain images are not honored.

When `USE_IMAGE_SECURITY` is disabled in the bootloader, the bootloader honors the image as is in the upgrade file. It is expected that the image upgraded is a plain image in this case.

In either case, the bootloader updates the image Type value in EEPROM (see also Section 2.5.4) to reflect the new image's capability and for application's use in outgoing OTA upgrade requests.

2.5.4 EEPROM Usage for OTAU Parameters

EEPROM is used to share non-volatile parameters between application and bootloader. Even if upgrade image has EEPROM content, using it for sharing is not a problem. This is because, for application to bootloader sharing, bootloader reads EEPROM parameters before writing with image's content. For bootloader to application sharing, the non-volatile parameters are over-written after copying image's contents to EEPROM.

Table 2-7 shows the index usage in EEPROM.

Table 2-7. EEPROM Usage for Secure Bootloading

EEPROM location	Direction	Value	Description	Additional comments
0-3	Application to bootloader	Action & CRC	Bootloader action is notified. External flash contents' 8-bit CRC is stored	-
4	Application to bootloader	1	Use the key & IV from EEPROM	Specific to secured bootloader
		Any other value	Use default key	
5-20	Application to bootloader	128-bit hex value	If byte 4 is a '1', read 128-bit key from this offset	Specific to secured bootloader
21-36	Application to bootloader	128-bit hex value	If byte 4 is a '1', read 128-bit initialization vector from this offset	Specific to secured bootloader
37-38	Bootloader to Application	16-bit hex value	"Image Type" of the newly upgraded image	-
39-43	Application to bootloader	32-bit NVM start address	Bootloader skips writing to internal flash to not overwrite non-volatile parameters	-
44-47	Application to bootloader	32-bit NVM end address		-

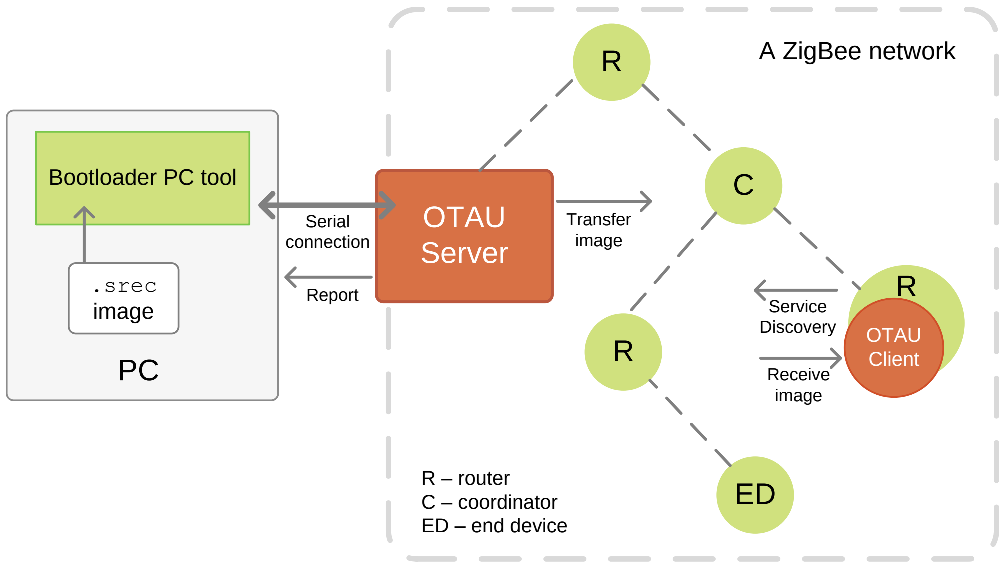
2.6 Upgrade Access Point Tools

Once the network is up and running, the user may proceed with starting the upgrade process with the help of PC tools provided with the BitCloud SDK. In case of a dedicated upgrade access point (UAP), the user shall first configure a Runner device, which will serve as an OTAU server, while in case of an in-network UAP it is assumed that the server device is already present in the network. In both cases the server device shall be connected over a serial link to the Bootloader PC tool installed on a PC.

The Bootloader PC tool is used to initiate and control the upgrade process. Another tool involved is the Image Converter utility used to convert `*.srec` images into compatible `*.zigbee` images, which can be uploaded to the devices. The Bootloader PC tool and the Image Converter utility are provided with the Serial Bootloader software package [5].

The overall upgrade process is illustrated in Figure 2-5. The OTAU Server depicted in the illustration is an in-network device supporting the OTAU server cluster. Proceed to Section 2.6.1 for details on connecting the Bootloader PC tool to an OTAU server. The procedure for updating the network after the server is connected is given in Section 2.6.2.

Figure 2-5. Over-the-Air Upgrade with the OTAU Bootloader Tool



2.6.1 Connecting the OTAU Bootloader Tool to an In-network Server

The following steps illustrate a typical sequence to connect the Bootloader PC tool to an in-network OTAU server and prepare to upgrade the target network:

1. Start the Bootloader PC tool.
2. Switch to the OTAU tab as shown in Figure 2-6.
3. Specify connection settings to match the port where the OTAU server device is connected.
4. Click the Start passive mode button. The program will listen to the specified port to detect the OTAU server device.

2.6.2 Updating the Network with the Bootloader PC Tool

The following steps illustrate a typical sequence to upgrade a single device on the network:

1. Connect the tool to a server as described in Section 2.6.1.
2. The utility will automatically populate the list of devices that support the OTAU functionality (that is, applications that include the OTAU client cluster). By default, only devices programmed with application images with OTAU support should be shown in the list. The operation may take up to one minute and more depending on end devices' sleep periods.
3. Start the Image Converter utility (you will also be able to convert images in the Bootloader PC tool although it is not possible to set metadata information there):
   a. Select `*.srec` image(s) you wish to upload to a remote OTAU-capable device over the air.
   b. Fill in image metadata information in fields below and click Convert. You can specify the firmware version and the stack version.

Note: An *.srec image may contain an EEPROM payload. This is configured in the Image Converter utility through the checkbox near the Erase label. If it is checked, the image generated by the utility will contain the EEPROM part and the device's EEPROM will be cleared during update with this image. If the Erase checkbox is left unchecked, the firmware will not contain the image for EEPROM so that after the update device's EEPROM data will stay unchanged.

4. Return to the Bootloader PC tool:
   a. Click on the Update button next to device information.
   b. In the window that has opened specify the folder containing firmware images either in the `*.zigbee` or the `*.srec` format. `*.srec` images can be converted in place by clicking on Convert.

c. Select a suitable image and click Upload and the upload process will begin.
d. Once the image is uploaded the progress bar next to the updated device will be replaced by a button. Click on this button to send an update end response to the device, informing it that it can swap application images. Note switching to a new firmware can take additional time.

Upload progress for each device being updated will be shown in the OTAU server tool window depicted on Figure 2-7. When a sleeping end device uploads a new firmware image, it suspends sleeping, but it still uses the polling mechanism to request data from its parent. Therefore for end devices with greater `CS_INDIRECT_POLL_RATE` parameter, which specifies the period of time between two poll requests, it will take longer to download the firmware image.

Figure 2-6. The OTAU Server Tool Main Screen

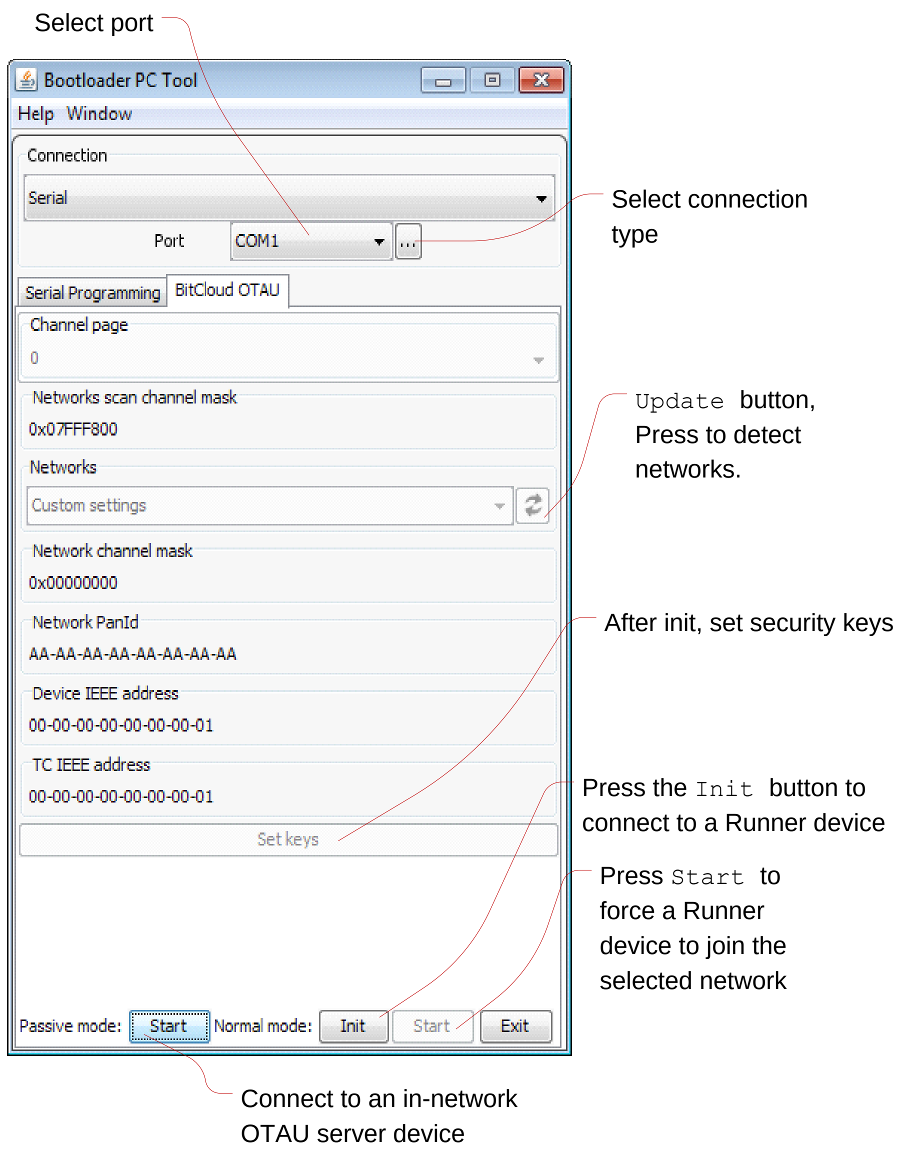
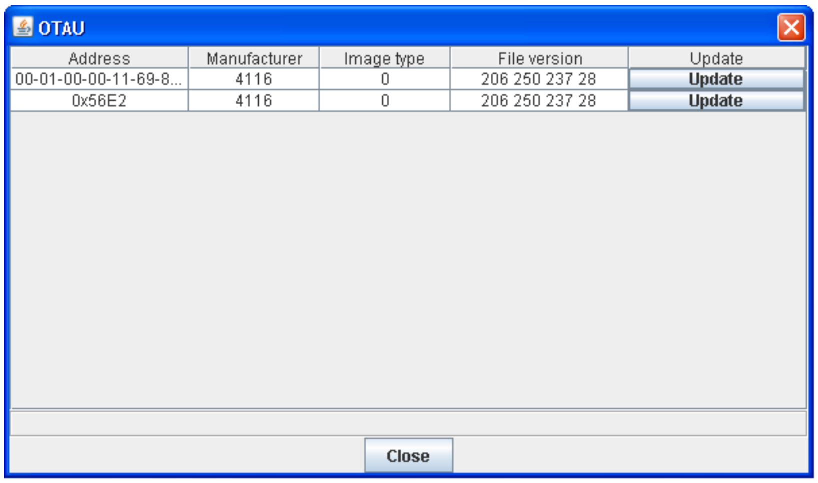


Figure 2-7. The OTAU Server Tool Devices' Screen



Address	Manufacturer	Image type	File version	Update
00-01-00-00-11-69-8...	4116	0	206 250 237 28	Update
0x56E2	4116	0	206 250 237 28	Update

Appropriate "`Image type`" as described in Section 2.5.1.2 will be displayed here.

2.7 Custom Use of ISD

The ISD component may be used to implement custom OTAU mechanism. In this case the server (that is, a device that distributes firmware images) should use the ISD API to pass messages from the client device to the image storage system.

ISD communicates with the storage system through the serial interface. The `ISD_Open()` and `ISD_Close()` functions are used to open and close the serial interface, respectively. While the serial interface is open the following functions may be called to address the storage system with a request from the client:

- `ISD_QueryNextImageReq()` to inform the storage system about a client requesting a new image and the client's firmware version: the success status in the response should indicate that a new image is available for the client
- `ISD_ImageBlockReq()` to request the specified block of data of the image from the storage system
- `ISD_UpgradeEndReq()` to notify the storage system that the client has uploaded the whole image and is only waiting for the update end response command to switch the current image with the new one

All ISD functions are executed asynchronously, with the callback's being called when the storage system responds to the message sent by ISD. For API specification refer to [4].

The storage system sends three types of commands each of them having the structure shown in Table 2-8. The storage system may add any sense to the meaning of the fields. ISD uses only command ID to identify the type of the message and to raise the right callback function. Besides, the payload field is cast to the type corresponding to the message type. The way other fields are processed in on behalf of the application and the callback functions it specifies, calling ISD API functions.

Table 2-8. ISD Protocol Command Structure

Field	Size [bytes]	Comment
Address mode	1	Standard addressing information that is used to identify the client to which the response is addressed
Short address	2	
Extended address	8	

Field	Size [bytes]	Comment
Profile ID	2	
Endpoint	1	
Destination endpoint	1	
Cluster ID	2	
Default Response	1	Being set to 1, indicates that the default response should be sent
Command's options	1	Bit 1 – direction, bit 2 – is general command; remaining bits are not used

Command ID	1	May have one of the following values:
		`QUERY_NEXT_IMAGE_REQUEST_ID`
		`IMAGE_BLOCK_REQUEST_ID`
		`UPGRADE_END_REQUEST_ID`
		(defined in the `zclOTAUCluster.h` file)

Payload	Depends on command	May be cast to one of the following types depending on the command ID:
		`ZCL_OtauQueryNextImageResp_t`
		`ZCL_OtauImageBlockResp_t`
		`ZCL_OtauUpgradeEndResp_t`

3. Reference

[1] 095264r21 ZigBee Over-the-Air Upgrading Cluster Specification

[2] ZigBee PRO specification (053474r20)

[3] AVR2052: BitCloud Quick Start Guide

[4] BitCloud API Reference (available in BitCloud SDK)

[5] AVR2054: Serial Bootloader User Guide

Document Revision History

Doc. Rev.	Date	Comment
E	09/2015	Modified for BitCloud 3.3 release.
D	01/2015	Modified for BitCloud 3.2 release.
C	08/2014	Modified for BitCloud 3.1 release. Added SAMR21 and security support.
B	03/2014	Updated list of supported platforms to RFR2 family. Document clean up.
A	08/2011	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Bldg.
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370