APPLICATION NOTE

AVR2052: BitCloud SDK Quick Start Guide

Atmel MCU Wireless

Introduction

This document is intended as a starting point for software engineers' prototyping, implementing, testing, and deploying ZigBee® Home Automation (ZHA), ZigBee Light Link (ZLL), and OEM ZigBee PRO devices based on the Atmel® BitCloud® software platform [1].

The BitCloud Software Development Kit (SDK) provides a complete set of tools – including BitCloud ZigBee PRO libraries, reference applications, API documentation, – required to build ZigBee-compliant end products running customized ZHA and ZLL applications.

This document describes how to quickly start with the BitCloud SDK by installing development environment, assembling hardware and programming devices with reference applications.

Chapter 1 provides an overview of the SDK, required development tools, and lists supported platforms.

Chapter 2 describes the documentation set available for BitCloud SDK. Chapter 3 gives instructions on SDK and tools development tools setup.

Chapter 4 describes how to build BitCloud application using supported IDEs.

Chapters 5, 6, and 7 provide description of ZLL, ZHA, and WSNDemo reference applications supplied with the SDK.

Starting from Appendix A the hardware specific part of the document begins. Each next appendix chapter describes the usage of a particular platform.

Features

- Introduction to BitCloud Software Development Kit (BitCloud SDK) [1]
- Description of SDK contents
- Development tools installation procedure
- Application build process
- Description of reference applications
  - ZHADevices – ZigBee Home Automation devices
  - ZLLDemo – ZigBee Light Link devices
  - WSNDemo – OEM ZigBee devices

Table of Contents

1. Overview

BitCloud is a full-featured, production grade, embedded software development platform from Atmel. It provides a framework for creating ZigBee Home Automation (ZHA), ZigBee Light Link (ZLL), and proprietary ZigBee devices running on supported Atmel microcontrollers and IEEE® 802.15.4-2006-compliant [5] radio transceivers. The BitCloud stack is fully compliant with the ZigBee PRO standards for wireless sensing and control.

1.1 BitCloud SDK

The main items provided as part of the BitCloud software development kit (SDK) [1] are:

- Atmel implementation of ZigBee PRO core stack protocol in form of libraries and API header files. The same core-stack library is used for all BitCloud applications.
- Source code and IDE projects for Atmel reference applications:
  - **HADevice** - ZigBee Home Automation Profile devices (see Chapter 6)
  - **ZLLDemo** - ZigBee Light Link Profile devices (see Chapter 5)
  - **WSNDemo** - OEM ZigBee PRO device implementation (see Chapter 7)
  - **Blink** - basic example that only does LED blinking
  - **ZAppSINP** - ZigBee PRO network processor application (see [28] for details)
- Source code of some of the BitCloud components, including:
  - ZigBee Cluster Library
  - Hardware Abstraction Layer
  - Board Support Package
  - System Task Manager

  …and some others.
- Set of precompiled firmware images for reference applications
- ZLL certification test suite (see Section 5.11)
- Documentation files (see Chapter 2)
- Etc…

Detailed structure of BitCloud SDK is given in Table 3-1.

1.2 Development Tools

A development tool chain for BitCloud applications consists of:

- BitCloud SDK
- A set of development or custom boards with supported Atmel MCU and RF transceivers as given in Table 1-1
- An integrated development environment (Atmel Studio [25] or IAR Embedded Workbench® [19], [20]), where sample applications may be modified, compiled, and debugged. IDE versions supported by BitCloud SDK are given for particular platforms in Table 1-1
- A corresponding compiler tool chain (AVRGCC, ARMGCC, or IAR™), which provides the necessary tools to compile application source code into binary images
- A programming device (for example, JTAGICE3 [22]), which may be used to program and debug the application on a target platform
- Optional: Atmel Serial/OTA Bootloader [12] if firmware programming over the serial interface or over-the-air is required
- Optional: ZigBee packet sniffer tool [23] for capturing over-the-air traffic

Setup instructions for the BitCloud SDK as well as supported IDEs are given in Chapter 2.

1.3 Supported Hardware Platforms and IDEs

The supported hardware platforms are shown in Table 1-1.

Table 1-1. Supported Hardware Platforms and IDEs

Name in this Document	Microcontroller	Supported RF Transceivers	Supported Evaluation Kits	Supported IDEs
megaRFR2	ATmega256RFR2 [9] ATmega2564RFR2 [10]	Built-in	AT256RFR2-EK [15] ATMEGA256RFR2-XPRO [17]	IAR Embedded Workbench for AVR® 6.40.3 (with C/C++ compiler 6.40.3) [20] Atmel Studio v6.2 (with AVR_8_bit_GNU_Toolchain_3.4.5_1522) [25]
SAMR21	ATSAMR21G18A [31] ATSAMR21E18A [32]	Built-in	ATSAMR21-XPRO [33]	IAR Embedded Workbench for ARM® 7.30 (with C/C++ compiler 7.30) [19] Atmel Studio v6.2 (with GCC version 4.8.3 213147, Atmel build 371) [25]

2. BitCloud Documentation

This chapter describes the documentation set available for BitCloud SDK. It is intended to help the user to understand where to find information required during application evaluation and development.

Table 2-1 lists all documents that compose BitCloud documentation set. The list of documents is the same for all platform-specific packages. The document files are available in the `/Documentation/` folder of the BitCloud SDK as well as from the Atmel website http://www.atmel.com/.

Table 2-1. BitCloud Documentation List

Title	Description
AVR2052: BitCloud Quick Start Guide [1]	This document. Contains following parts: • BitCloud SDK overview • SDK and IDE installing instructions • Description of reference ZHA, ZLL, and WSNDemo applications • Platform-specific details related to BitCloud SDK
AVR2050: BitCloud Developer Guide [4]	Focuses on user's application development and provides: • Architecture of the stack and that of a user's application • Application development concepts and rules • Stack features descriptions with reference to BitCloud API and code examples The document is organized around main tasks that a ZigBee application normally should accomplish. The tasks are grouped by the areas, to which they belong (such as network management, data exchange, security, etc.). Information contained in a developer guide is generally platform-independent unless stated otherwise.
BitCloud API Reference [3]	Provides full specification and description of functions and data types that compose BitCloud public APIs. API reference also describes the most common uses of the APIs illustrated with code samples mostly extracted from reference applications. API reference document is provided in CHM and HTML formats.
Application notes	
AVR2058: BitCloud OTAU User Guide [24]	Describes how to use the Over-the-Air upgrade feature in BitCloud applications.
AVR2057: ZAppSI User Guide [28]	Describes development of applications using ZAppSI serial protocol, protocol's implementation – ZAppSI host library, and scripting environment based on Python. Contains user's instruction for the WSNRunner development tool.
AT02597: ZigBee PRO Packet Analysis with Sniffer [23]	Describes how to configure and use various packet sniffing tools (along with Atmel MCU-based sniffer hardware) for analyzing ZigBee traffic.
AVR2054: Serial Bootloader User Guide [12]	Describes the standalone Serial Bootloader package, which is used to load firmware images to devices via serial connection. Not included into BitCloud SDK.
AT03663: Power Consumption of ZigBee End Devices [30]	Provides detailed description on the power consumption of Atmel ZigBee End Device node in various networking scenarios.
AT08550: ZigBee Attribute Reporting [36]	Provides in-depth details on the configuration and usage of Attribute Reporting in BitCloud applications.

2.1 Learning BitCloud

As evident from Table 2-1 BitCloud documents are divided into the following categories:

- Quick start guide
- Developer guide
- API reference
- Application notes

AVR2052: BitCloud SDK Quick Start Guide (this document) [2] is intended to be the starting point for a user learning BitCloud programming. Once the user understands the reference applications, a quick start guide may be used as a reference book for hardware-specific details.

Actual application development is described in the developer guide: *AVR2050: BitCloud Developer Guide* [4]. This document describes programming basics such as overall application's organization, task management, and other topics, as well as describes common ZigBee tasks in detail. This document is inevitable for the application development on top of BitCloud stack.

The user may find it convenient to start investigating the use of BitCloud APIs from *AVR2050: BitCloud Developer Guide* and then look for further information in other developer guides. Note that specification of all APIs available with a package can be found in the BitCloud API Reference [3].

Application notes focus on specific important topics. This kind of documents may contain instructions on installation of specific features, and the usage of related development tools and APIs.

3. Development Environment Setup

This chapter provides instructions on how to set up BitCloud SDK as well as supported IDEs. It also describes the structure of the BitCloud SDK and includes references to hardware setup of the supported platforms.

3.1 Installing the BitCloud SDK

Install the BitCloud SDK by unzipping BitCloud `.zip` archive into an empty folder with no blank spaces in the path to it (that is, avoid having folder names such as `/Program Files/`, `/My Documents/` and similar in the installation path).

Note: If the SDK installation path contains any blank spaces in its directory names, errors indicating path issues will occur when compiling reference and custom applications with the SDK.

Table 3-1 lists the location and purpose of key folders provided with the SDK.

Table 3-1. BitCloud SDK Directory Structure

Directory/File	Description	Notes
`./Applications/`	Folder containing reference applications	
`./Applications/ZLLDemo/`	ZLL reference application	Full application description is given in Chapter 5
`./Applications/HADevice/`	ZHA reference application	Full application description is given in Chapter 6
`./Applications/WSNDemo/`	Proprietary application for OEM-devices based purely on ZigBee PRO stack. Doesn't use ZigBee Clusters	Full application description is given in Chapter 7
`./Applications/Blink/`	Basic application that only blinks LEDs on the board. No over-the-air frames are exchanged	
`./Applications/ZAppSINP/`	ZAppSINP reference application for network processor	Full application description is given in [28]
`./BitCloud/`		
`./BitCloud/lib/`	Makerules and library files for BitCloud PRO stack and HAL	Doesn't require any modifications by a user
`./BitCloud/Components/`	ZigBee PRO stack header files organized by stack component and included by reference applications	For more details on BitCloud stack's components and their usage see [4]
`./BitCloud/Components/ZCL/`	ZCL header files included by reference applications	
`./BitCloud/Components/HAL/`	HAL component source code and header files for application access to available hardware interfaces such as UART and SPI	The HAL component is compiled separately from the application and from the core stack components
`./BitCloud/Components/BSP/`	BSP component source code and header files for application access to external peripherals (for example, LEDs, buttons, LCD available on development boards)	

Directory/File	Description	Notes
./Evaluation Tools/		
./Evaluation Tools/ZLLDemo/	Precompiled firmware images of ZLL reference devices for supported platforms	Full application description is given in Chapter 5
./Evaluation Tools/HADevice/	Precompiled firmware images of ZHA reference devices for supported platforms	Full application description is given in Chapter 6
./Evaluation Tools/WSNDemo (Embedded)/	Precompiled firmware images of WSNDemo devices for supported platforms	Full application description is given in Chapter 7
./Evaluation Tools/WSNDemo (WSN Monitor)/	Installer of WSNMonitor PC application required for WSNDemo	WSN Monitor is described in Section 7.4
./Evaluation Tools/ZAppSINP/	ZigBee PRO network processor application.	See [28] for details
./Evaluation Tools/Runner/	Installation file for the WSNRunner application, which is used to run the ZLL test scripts	Described in Section 5.11
./Evaluation Tools/SLRemote/	ZLL Bridge GUI application's installation files	See Section 5.7.2
./Evaluation Tools/ZLL_Scripts/	ZLL certification test scripts	See Section 5.11
./Documentation/	BitCloud documentation, including this Quick Start Guide	See Chapter 2
./ThirdPartySoftware/	Third party software such as drivers, libraries, etc.	

3.1.2 Serial and OTAU Bootloader Setup

For users who intend to use Serial Bootloader or Over-the-Air Upgrade (OTAU) features, find detailed description of the Serial Bootloader package, the list of supported platforms, instructions on generating SREC images in *AVR2054: Serial Bootloader User Guide* [12]. OTAU use is fully described in *AVR2058: OTAU User Guide* [24].

3.2 IDE Installation

3.2.1 Atmel Studio

Atmel Studio can be used to develop and debug applications for AVR- and ARM®-based platforms. Atmel Studio is equipped with the GCC compiler and does not require any additional external tools to compile and debug BitCloud applications. In order to compile the BitCloud applications using command line few of the utilities which are required are vailable only as part of MYSIS [35].

Installation procedure:

a. Download and install Atmel Studio [25] of the supported version given in Section 1.3, if not already installed on your PC.
b. Add path to the folder containing the AVRGCC compiler to the `Path` Windows® environment variable. The compiler is located in the `\extensions\Atmel\AVRGCC\3.3.1.27\AVRToolchain\bin` directory of the Atmel Studio installation directory. This step is necessary for command line compilation (with makefiles).
c. For detailed instructions on how to compile applications using Atmel Studio, refer to Chapter 4.

3.2.2 IAR Embedded Workbench

IAR Embedded Workbench for Atmel AVR [20] can be used to develop and debug applications for AVR-based platforms. IAR Embedded Workbench for ARM [19] can be used to develop and debug applications on ARM-based platforms. IAR IDEs support editing of application source code, compiling source files, linking object modules with libraries, and application debugging.

Installation procedure:

- IAR Embedded Workbench for AVR:
  a. Download and install IAR Embedded Workbench for Atmel AVR [20], if not already installed on your PC.
  b. Add a Windows environment variable named `IAR_AVR_HOME`, and set its value to the IAR Embedded Workbench installation directory (for a default installation, it is `C:\Program Files\IAR Systems\Embedded Workbench 6.40`). To do this, go to `Control Panel > System > Advanced > Environment Variables`, click `New` below the `System variables` list, and enter Variable Name and Variable Value. This step is required if you plan to build embedded images using IAR Embedded Workbench from the command line.
  c. For detailed instructions on how to compile applications using IAR Workbench, refer to Chapter 4.
- IAR Embedded Workbench for ARM:
  a. Download and install IAR Embedded Workbench for ARM [19], if not already installed on your PC.
  b. Add a Windows environment variable called `IAR_ARM_HOME`, and set its value to the IAR Embedded Workbench installation directory (for a default installation, it is `C:\Program Files\IAR Systems\Embedded Workbench 7.20`). To do this, go to `Control Panel > System > Advanced > Environment Variables`, click `New` below the `System variables` list and enter Variable Name and Variable Value. This step is required if you plan to build embedded images using IAR Embedded Workbench from the command line.
  c. For detailed instructions on how to compile applications using IAR Workbench, refer to Chapter 4.

3.3 Hardware Configuration

Hardware configuration instructions depend on the particular hardware platform used to evaluate and develop with the BitCloud SDK.

To get started, proceed to the platform-specific sections listed in Table 3-2.

Table 3-2. Hardware-specific getting Started Sections

Section Name	SAMR21-specific Sections	ATmegaRFR2-specific Sections
Hardware-specific Appendix	Appendix A	Appendix B
Hardware setup	A.1	B.1
Precompiled images	A.2	B.2
Programming devices	A.3	B.3
Reserved resources	A.4	B.4

4. Building BitCloud Applications

This chapter provides an overview on how to use Atmel Studio and IAR IDEs to work with reference BitCloud applications. IDE versions that are used during verification and guarantee to work are given in Table 1-1.

As mentioned in Section 1.1, a part of the stack components and hardware drivers are provided in source code, and are not part of the stack library. For convenience reasons, source files for these components are included in the IDE projects and can be accessed from the IDE.

4.1 Building Applications in Atmel Studio

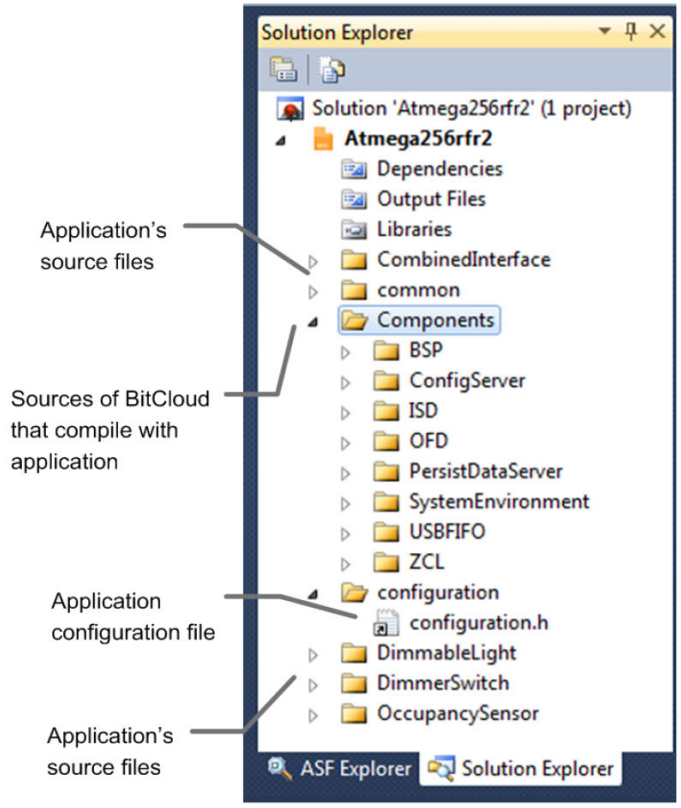
Atmel Studio can be used to develop and build Atmel BitCloud applications. Reference applications include Atmel Studio project files located in the `\atmelStudio_projects` subdirectory of the application root directory. These projects rely on the configurations given by external low-level makefiles (see Section 4.3).

Atmel Studio GUI allows the user to select an appropriate configuration from the list of available configurations and to modify any given configuration. For details on compilation and editing of configurations, refer to Atmel Studio documentation [27].

- **Building application from IDE:**
  - Open an appropriate `.atsln` project file from the `<appName>\atmelStudio_projects` directory with Atmel Studio. Solution Explorer tab as shown on Figure 4-1 provides access to the application source files as well as stack components that compile together with the application.

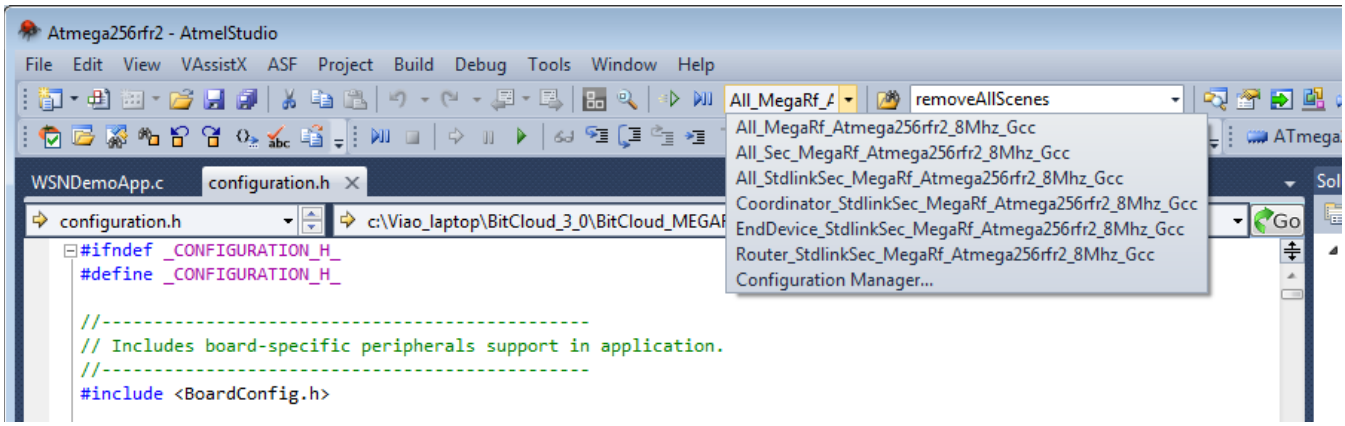
Note: Source files are included virtually to provide access to them. The set of files to be actually compiled is defined in corresponding external Makefile as described in Section 4.3.

Figure 4-1. Example Structure of Atmel Studio Application Project



- Select a target configuration in the dropdown list on the toolbar, as shown on Figure 4-2

Figure 4-2. Selecting Project's Configuration in Atmel Studio



- From the main menu execute *Build => Rebuild All*

Once the build process is completed, some of the `.hex`, `.srec`, `.bin`, and `.elf` image files will be generated, depending on the platform configuration that has been chosen. Use the `.hex` file for programming devices via JTAG and the `.srec` file for programming via Serial Bootloader. The `.elf` file is used for debugging.

- **Building application from command line:**
  - After selecting the target configuration in the application `Makefile` (see Section 4.3), compile the application by running the make utility, executing

```
make clean all
```

It is possible to run the make utility from Atmel Studio by selecting `Tools > Command Prompt`. This will guarantee that the make utility provided with Atmel Studio is used. Otherwise, the path to the folder containing the make utility can be added to the `Path` environment variable. In this case, run the make utility in the command line from the application's root directory.

4.2 Building Applications in IAR Embedded Workbench

IAR Embedded Workbench can be used to develop and build Atmel BitCloud applications. All reference applications include IAR project files located in the `\iar_projects` subdirectory of the application root directory. IAR projects come complete with a set of configurations, which correspond to the configurations given by low-level makefiles.

IAR Embedded Workbench GUI allows the user to select an appropriate configuration from the list of available configurations and to modify any given configuration. For details on compilation and editing of configurations, refer to the IAR Embedded Workbench documentation [21].

As mentioned above, a part of stack components and drivers are compiled with the application. For convenience reasons, source files for these components are included in the IAR projects, so they are effectively a part of the application.

For compilation from the command line with the IAR compiler, makefiles are used in exactly the same way as described in Chapter 4.3.

- ***IDE build procedure*:** Open the `.eww` file in the iar_projects subdirectory of the appropriate application directory (for WSNDemo, the `WSNDemo.eww` file from the `<SDK-Root>\Applications\WSNDemo\iar_projects` subdirectory) with IAR Embedded Workbench, select appropriate configuration (as shown in Figure 4-3) and execute the `Rebuild All` item from the `Project` menu.

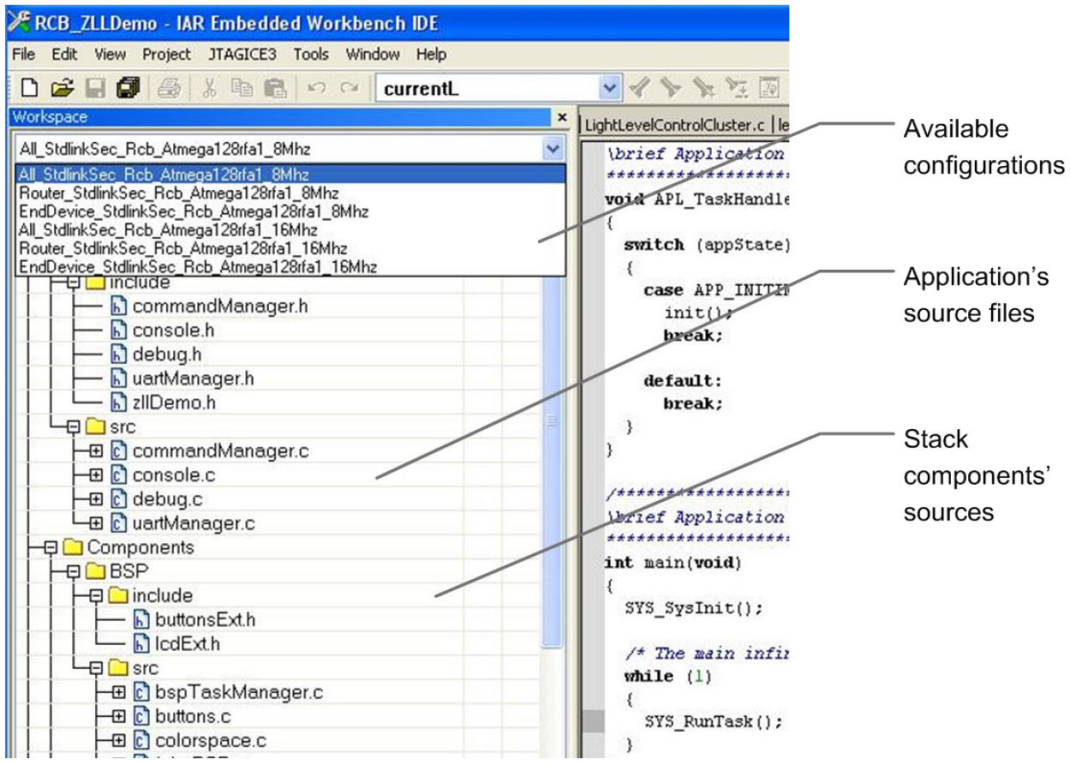
By default, the `.a90` file (for WSNDemo, `WSNDemo.a90`) will be generated in the `\iar_projects\Debug\exe` subdirectory (for WSNDemo, in the `<SDK-Root>\Applications\WSNDemo\iar_projects\Debug\exe` directory) with format as specified in Linker Output Options of the IAR project.

- ***Command line build procedure:*** Compile the application by running the make utility, executing

```
make clean all
```

Some of the `.hex`, `.srec`, `.bin`, and `.elf` image files will then be generated, depending on the platform configuration that has been chosen.

Figure 4-3. Project's Structure and Configuration's Selection in IAR Embedded Workbench



4.3 Makefiles Organization

Each sample application is provided with makefiles for the most typical application configurations. Makefiles are located in the `\makefiles` directory inside subdirectories corresponding to different supported boards. In addition to these low-level makefiles each application includes high-level makefile located in the application root folder.

The high-level makefile is used to specify the low-level makefile that will be used to build the application. The choice depends on the values assigned to special variables inside the high-level makefile:

- `PROJECT_NAME`: specifies the subdirectory name of the `\makefiles` directory where the target file is located
- `CONFIG_NAME`: used to obtain the target makefile name by adding `CONFIG_NAME` to `Makefile_`

For example, if makefile contains the following lines:

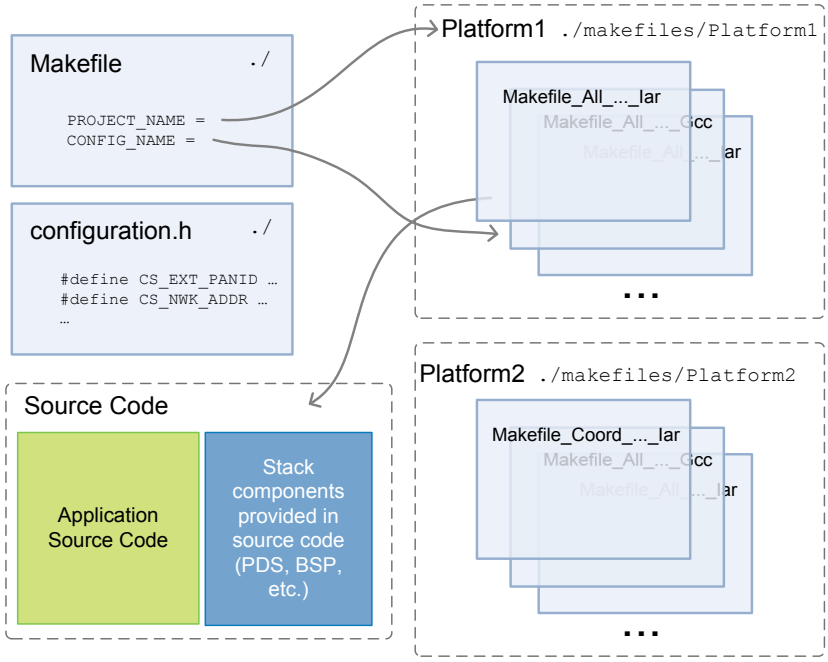
```
PROJECT_NAME = Atmega256rfr2
CONFIG_NAME = All_StdlinkSec_MegaRf_Atmega256fr2_8Mhz_Gcc
```

then the compilation instructions will be extracted from the makefile located at

```
\makefiles\Atmega256rfr2\Makefile_All_StdlinkSec_MegaRf_Atmega256fr2_8Mhz_Gcc
```

The application structure is illustrated in Figure 4-4. A high-level makefile for sample applications already contains commented lines for all configurations provided, so the user just has to uncomment appropriate lines.

Figure 4-4. Application Build Structure with Makefiles



After the desired configuration is chosen in the makefile, the application can be built by executing make clean all from the command line in the application root folder or by selecting the Build command in the context menu in Atmel Studio.

4.3.2 Low-level Makefile Name Structure

The name of a low-level makefile consists of parts showing which configuration the file specifies. These include specification of:

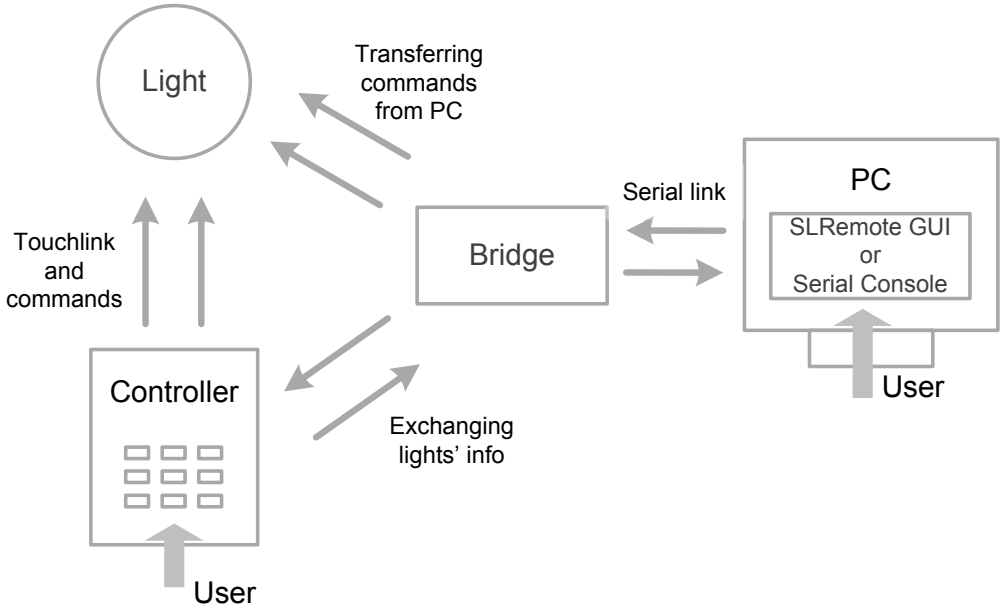
- ZigBee device type (`All`, `Coordinator`, `Router`, or `EndDevice`); All means that this configuration can be used for any device type
- Security supported (standard with link keys (`StdlinkSec`), standard (`Sec`), or none (empty))
- Platform (board or SoC family)
- MCU
- Radio chip, if applicable
- MCU frequency
- Compiler

Note: Not all combinations make sense for a given platform. Makefiles are provided only for supported configurations.

5. ZigBee Light Link Reference Application

ZLLDemo reference application implements standard device types defined in the ZigBee Light Link Profile specification [29] color scene controller, light devices, and bridge. Figure 5-1 shows a scheme of interactions between devices in a ZLL network. Several types of the light device with different sets of supported clusters and cluster commands are provided: on-off light, dimmable light, color light, temperature color light and extended color light.

Figure 5-1. Interactions Scheme in the ZLLDemo Application



The application demonstrates controlling lights by color scene controllers as well as their interactions with a bridge device. All main features of the ZigBee Light Link profile are implemented:

- Light control functionality, including On/Off, Level (brightness), Color (hue, saturation), Groups, and Scenes
- Touchlink commissioning – network parameters are transferred to a new light from a scene controller, while the scene controller is kept close to the light
- A ZigBee Light Link network has no ZigBee network coordinator, even a bridge device acts as a ZigBee router
- Lights are ZigBee routers and propagate messages across the network
- Multiple controllers may be used to control same or different sets of lights
- Application data and network parameters are saved to non-volatile memory to restore the state after reset, power off, etc.

5.2 Launching the Demo

To launch the demo, at least one light is needed and one color scene controller or a bridge. More lights and/or controllers may be added as required.

Follow the instruction below to launch a demo:

1. Assemble devices as instructed for target platform in Section 3.3.
2. Program devices with firmware images as described for corresponding platform in Section 3.3.
   The pre-built images are located in the `\Evaluation Tools\ZLLDemo` directory. Precompiled light's firmware is for an extended color light device. To program other types of the light device, the application must be recompiled for corresponding device type set in the `configuration.h` file (see Section 5.4.1).

Note: Prebuilt firmware images are built for *8MHz* MCU frequency.

3. Bring devices into the same network and perform light control.
    a. Using a bridge device:
        - Use SLRemote PC application or serial console commands to create a network via the bridge device. Open the network for joining. See Section 5.6.2 for bridge device functionality description.
        - Reset light device, on power up it will automatically join the network
        - Perform light discovery from the bridge using SLRemote or serial console commands
        - Control discovered lights using SLRemote GUI or using console commands

        See Section 5.6.2 for more details on the bridge device functionality including syntax of serial console commands and use of SLRemote GUI.
    b. Using a color scene controller device:
        - Bring color scene controller in close proximity to a light. Perform a touchlink procedure between the light and the color scene controller by using either corresponding on-board button on the controller device or using “touchlink” serial command.
        - Control touchlinked light from remote controller using its on-board buttons or serial console commands

        See Section 5.8 for detailed description of color scene controller functionality including syntax of serial console commands, reset to factory new, etc.
4. See Section 5.5 for details on the light functionality such as light status indication, reset to factory new, logging into serial interface, etc.
5. *Additionally:*
    - More light devices and controller devices may be added through touchlink with the controller device that is already in the network or via classical joining
6. For details in executing firmware Over-the-Air upgrade (OTAU) see Section 5.9.
7. For details on interoperability with Home Automation networks see Section 0.

5.3 Supported Clusters

Table 5-1 lists clusters supported by the demo applications for light and color scene controller. Note that most of the clusters used by Light Link applications duplicate common clusters from ZigBee Cluster Library, but may be slightly different, and so applications should employ clusters specially defined for the ZigBee Light Link profile (header files for these clusters include `zll` in their names).

Table 5-1. ZigBee Clusters Supported by Devices in the ZLLDemo Application

Device Type	Server Clusters	Client Clusters
On/Off light Dimmable light Color light Extended color light Color temperature light	Basic ZLL Commissioning OnOff Groups Identify Scenes Level Control Color Control (for color capable lights only)	Basic OTAU (if enabled)

Device Type	Server Clusters	Client Clusters
Color scene remote controller	Basic ZLL Commissioning	Basic ZLL Commissioning OnOff Level control Groups Identify Scenes Color Control Link Info (manufacture-specific) OTAU (if enabled)
Control bridge	Basic ZLL Commissioning Link Info (manufacture-specific) OTAU (if enabled)	ZLL Commissioning Identify OnOff Level Control Groups Scenes Color Control

5.4 Source Code Organization

Application projects and source code are located in the `\Applications\ZLLDemo` folder inside the SDK. The source code is divided into the common part and device-specific code. The entry `main()` function is located in the `light.c`, `bridge.c`, and `colorSceneRemote.c` files in the corresponding folders. An endpoint for communication between clusters is registered in the same file.

Supported clusters are configured in `<device>Clusters.c` files. A separate source code file is provided for each cluster supported by a specific device. Such file initializes structures needed for the cluster and implements callback functions that are called to indicate commands' responses. For example, see the `lightColorControlCluster.c` file, which initializes the color control cluster for the light device.

The application's configuration is set in the `configuration.h` file located in the `\Applications\ZLLDemo` folder. Serial interface used by the device to send information to a PC is also configured in this file. Additionally, in the application's source code UART is configured in the `\Applications\ZLLDemo\common\src\uartManager.c` file.

5.4.1 Application Configuration

Reference application's configuration parameters are set in the `configuration.h` file. Table 5-2 describes parameters of particular interest to the user. Note that parameters starting with `APP_` are application specific (defined only in reference applications), while those starting with `CS_` are stack-level parameters implemented in the ConfigServer component.

Table 5-2. Key Application Parameters and Their Meanings

Parameter	Description

`APP_ZLL_DEVICE_TYPE`	Selects ZLL device type; possible values are:
	`APP_DEVICE_TYPE_ON_OFF_LIGHT`
	`APP_DEVICE_TYPE_DIMMABLE_LIGHT`
	`APP_DEVICE_TYPE_COLOR_LIGHT`
	`APP_DEVICE_TYPE_TEMPERATURE_COLOR_LIGHT`
	`APP_DEVICE_TYPE_EXTENDED_COLOR_LIGHT`
	`APP_DEVICE_TYPE_COLOR_SCENE_REMOTE`
	`APP_DEVICE_TYPE_BRIDGE`

Parameter	Description
APP_ENABLE_CONSOLE	Specifies whether or not serial console commands are supported in the application. If set to 1 then reception of serial commands is enabled on defined APP_INTERFACE. Also in such case sleep on end devices is automatically disabled.
APP_DEVICE_EVENTS_LOGGING	Configures events logging. If set to 1 then application will print information on application's events to serial port defined via APP_INTERFACE.
BSP_SUPPORT	Specifies the board that will be used by the application. On-board peripherals such as buttons, LEDs, LCD, MAC address reading, etc. will be configured and compiled according to the made selection. User can extend BSP options to support its custom board. BOARD_FAKE option can be used for testing network communication as it substitutes BSP API with stub functions.
APP_INTERFACE	Configures the serial interface used to connect the device to a PC. Available options depend on the platform and are listed in the configuration.h file. Values are of APP_INTERFACE_<name> format. For some interfaces (UART) additional parameters should be configured such as APP_USART_CHANNEL.
APP_PRIMARY_CHANNELS_MASK	Bitmask of ZLL primary channels.
APP_SECONDARY_CHANNELS_MASK	Bitmask of ZLL secondary channels.
APP_SCAN_ON_STARTUP	If set to 1 for the light device, the application scans for networks on startup.
APP_USE_OTAU	Set to 1 to enable OTAU support in the application and set to 0 to disable it.
OTAU_CLIENT OTAU_SERVER	Defines whether device acts as OTAU client (device that will be upgraded) or OTAU server (device that will provide the new firmware). Only one role shall be selected. By default bridge device acts as OTAU server and other devices as OTAU clients. Applicable only if APP_USE_OTAU is set to 1.
APP_USE_ISD_CONSOLE_TUNNELING	Support simultaneous usage of the same serial interface for passing (1) commands from console and (2) commands exchanged by the ISD driver and the bootloader PC tool. This parameter is valid for the OTAU server (the bridge device).
APP_USE_FAKE_OFD_DRIVER	Enables fake implementation of the OFD driver on OTAU client devices. This may be useful for testing OTAU on boards without external flash memory.
APP_SUPPORT_OTAU_PAGE_REQUEST	Configures use of OTAU image page request feature on OTA client devices (refer to [24] for details).
EXTERNAL_MEMORY	Specifies the type of external memory (where the new firmware image will be stored).
APP_ENABLE_CERTIFICATION_EXTENSION	Set to 1 to compile application for running certification test scripts (see Section 5.11) and to 0 otherwise.

5.5 Serial Console Commands

ZLLDemo reference application includes implementation of a serial console interface that allows control and monitoring of the device over a serial connection using a terminal program (HyperTerminal, RealTerm, etc.) on a PC.

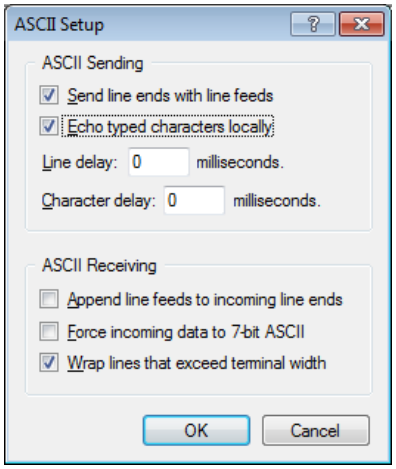
To enable use of such console in the application APP_DEVICE_EVENTS_LOGGING and APP_ENABLE_CONSOLE parameters shall be set to 1 in applications' configuration.h file. Additionally, APP_INTERFACE and, if applicable, APP_USART_CHANNEL shall be set correctly for the target board as described in Section Table 5-4. Configuration is the same independent on the device type used.

On the PC side virtual COM port connection that corresponds to the board shall have following settings:

```
BAUD RATE: 38400
PARITY: None
DATA BITS: 8
STOP BITS: 1
FLOW CONTROL: None (Hardware for the Xplained-PRO boards)
```

Additionally local echo and sending line ends with line feeds shall be enabled as shown in Figure 5-2 as an example for HyperTerminal. This configuration is available from the Connection Properties => Settings tab => ASCII Setup… button.

Figure 5-2. Additional HyperTerminal Configuration for Serial Console



Once connection settings and the COM port assigned to the bridge device are specified, type commands in the terminal window. Key supported commands are described in Table 5-3, while complete list is available in console implementation files: `colorSceneRemoteConsole.c`, `lightConsole.c` and `bridgeConsole.c`.

Table 5-3. Key Serial Console Commands

Command Syntax	Applies to Device Types	Description
help	All	Shows supported commands.
reset	All	Perform HW reset for the device. Upon reset the device will attempt to restore data from NVM and apply it instead of starting as factory new.
resetToFN	All	Resets the device to Factory New state by deleting network and application data from NVM. However this command doesn't reset the outgoing NWK security counter, and hence device is able to join its previous network.
resetToFD	All	Resets the device to Factory Default state by deleting network and application data from NVM. This command fully resets the outgoing NWK security counter and hence device might have problems joining to its previous network.
startNetwork	Bridge	Initiates scan and join to existing networks.

Command Syntax	Applies to Device Types	Description
createNetwork *<ePanIdHigh> <ePanIdLow> <ch> <panId> <addr> <updateId>*	Bridge	Creates a network with the provided parameters.
sendPermitJoin *<permit>*	Bridge	Sends the permit joining ZDP command to all routers, permitting joining to the network by association permanently (*<permit>* is greater than zero) or permanently forbidding it (*<permit>* is 0).
startDiscovery	Bridge	Starts discovery of the light devices.
touchlink	Color Scene Controller	Initiates touchlink procedure.
resetDeviceToFN	Color Scene Controller	Resets remote device to factory new state. Target device shall be in close proximity and will be indicating to show that it is selected for reset.
identify *<addrMode> <addr> <endpoint>*	Bridge, Color Scene Controller	Sends the Identify command to the specified light node(s).
onOff *<addrMode> <addr> <endpoint> <"-on" – turn on; "-off" – turn off>*	Bridge, Color Scene Controller	Sends On or Off command to the specified node(s).
setBrightness *<addrMode> <addr> <endpoint> <level> <time>*	Bridge, Color Scene Controller	Sets brightness level on one or several light devices, depending on the address mode used.
stepBrightness *<addrMode> <addr> <endpoint> <mode> <size> <time>*	Bridge, Color Scene Controller	Sends the Step Brightness command to the specified node(s).
setColor *<addrMode> <addr> <endpoint> <hue> <sat> <time>*	Bridge, Color Scene Controller	Sets color via hue and saturation on the specified light node(s).
stepSaturation *<addrMode> <addr> <endpoint> <mode> <size> <time>*	Bridge, Color Scene Controller	Sends the Step Saturation command to the specified node(s).
addGroup *<addrMode> <addr> <endpoint> <group>*	Bridge, Color Scene Controller	Sends the Add Group command to the specified node(s).
removeGroup *<addrMode> <addr> <endpoint> <group>*	Bridge, Color Scene Controller	Sends the Remove Group command to the specified node(s).
scene *<addrMode> <addr> <endpoint> <cmd: "-store" or "-remove"> <group> <scene>*	Bridge, Color Scene Controller	Sends Store Scene or Remove Scene command to the specified node(s).
startOtau	Light, Color Scene Controller	Start OTAU process if stopped.
stopOtau	Light, Color Scene Controller	Stop OTAU process.

5.6 Light's Functions

Light devices in ZLLDemo reference application are implemented following the requirements for corresponding device types defined in ZigBee Light Link specification [29]. In order to compile ZLLDemo for a light device `APP_ZLL_DEVICE_TYPE` parameter in application `configuration.h` file shall be set to one of the following values:

- `APP_DEVICE_TYPE_ON_OFF_LIGHT`
- `APP_DEVICE_TYPE_DIMMABLE_LIGHT`
- `APP_DEVICE_TYPE_COLOR_LIGHT`

- APP_DEVICE_TYPE_TEMPERATURE_COLOR_LIGHT
- APP_DEVICE_TYPE_EXTENDED_COLOR_LIGHT

That defines functional capabilities of the ZLL light according to ZLL spec.

On power up or reset a factory new (FN) light performs scanning for existing ZigBee networks (can be controlled by APP_SCAN_ON_STARTUP parameter) on primary and secondary ZLL channels. If any open networks are found, the light will try to join them using classical ZigBee MAC association mechanism. If such network discovery or joining fails the light simply stays in a listening mode on one of the primary channels and is ready to be joined via a touchlink procedure.

FN light as well as non factory new (NFN) light can be joined into the target network using a touchlink procedure. For this the light shall be brought in close proximity (10 - 20cm range) to a controller device and then touchlink procedure shall be initiated from the controller (see sections 5.8 and 5.6.2 for description of controller functionalities).

After the light is joined to a network it can be controlled remotely. On Atmel development boards a ZLLDemo light device application will indicate its light status (On/Off, Level, Color, etc.) via on-board LEDs, LCD screen, or just by printing text information to the serial interface, depending on the board and configuration. Section 5.6.2 provides details on such indications for particular development board.

Section 5.6.1 explains how a light device can be reset to factory new state. This includes a common way using a controller device, as well as mechanisms specific for particular development boards.

It is also possible to perform firmware Over-the-Air upgrade (OTAU) on light devices. For details, see Section 5.9.

5.6.1 Reset Light to the Factory New State

There are two ways to reset a light device to the factory new state:

1. Remotely, by sending a special command from a controller device (bridge or color scene remote):
   a. Bring the light into close proximity to the color scene controller device.
   b. From the color scene controller device initiate reset of a remote node to factory new state. For example:
      - using a resetDeviceToFN console command
        or
      - if using color scene controller device with 256RFR2 RCB mounted on a Key Remote Control board, hold R1, R2, and PWR buttons, altogether, on the color scene controller for three seconds
   c. Observe target light identifying. After that the light will be reset to FN new state.
2. Locally, using an on-board button:
   a. See Section 5.6.2 for the button functionality description.

5.6.2 Use of On-board Peripherals

Table 5-4 summarizes functionality of the board peripherals for a ZLLDemo light device. Additional information on HW setup is given in corresponding platform-specific sections as referenced in Section 3.3.

Table 5-4. Functionality of Board Peripherals for ZLLDemo Light Devices

Development Board	LEDs Functionality	Buttons Functionality	Indication to Serial Interface[1]
ATSAMR21-XPRO [34] (BSP_SUPPORT set to BOARD_SAMR21_XPRO)	Light status is indicated on LED 0	For reset to factory new - hold the SW0 button within one second after reset is released and it must be held pressed for atleast 1 second	Via EDBG USB connection [2]. Set APP_INTERFACE to APP_INTERFACE_USART and APP_USART_CHANNEL to USART_CHANNEL_1

Development Board	LEDs Functionality	Buttons Functionality	Indication to Serial Interface[1]
256RFR2 RCB on Key Remote Control board [16] (BSP_SUPPORT set to BOARD_RCB_KEY_REMOTE)	N/A For LCD indication see Figure 5-3	For reset to factory new - hold the PWR button on Key Remote pressed when RCB is reset via red switch	Via RS-232 connection on EXT header. Set APP_INTERFACE to APP_INTERFACE_USART and APP_USART_CHANNEL to USART_CHANNEL_0
256RFR2 RCB on STB [16] (BSP_SUPPORT set to BOARD_RCB)	N/A	N/A	Via USB connection. Set APP_INTERFACE to APP_INTERFACE_USBFIFO
AT256RFR2-XPRO [18] (BSP_SUPPORT set to BOARD_ATMEGA256RFR2_XPRO)	Light status is indicated on LED 0	For reset to factory new - hold the SW0 button pressed when XPRO board is reset	Via DEBUG USB connection [2]. Set APP_INTERFACE to APP_INTERFACE_USART and APP_USART_CHANNEL to USART_CHANNEL_1
Standalone 256RFR2 RCB [16] (BSP_SUPPORT set to BOARD_RCB)	Light status is indicated on LED D4	For reset to factory new, hold the black push-button pressed when RCB is reset via red switch	N/A

Notes: [1] – See Section 5.5 for configuration instructions.

[2] – Corresponding virtual COM port connection shall be used on PC side. Note that Flow Control shall be enabled on the PC when working with XPRO boards.

Figure 5-3 describes functions of a light's LCD screen for 256RFR2 RCB mount on a Key Remote Control board. For the on-off light only On/Off status is shown. A dimmable light also displays color level. For a color light color information as X-Y and hue-saturation is added. A temperature color light allows setting the color via– temperature value is shown instead of X-Y color coordinates. Extended color light's screen shows all these values.

Figure 5-3. Light's LCD Screen on a Key Remote Control Board. Parameters shown on the Screen depends on Light Device Type

5.7 Bridge's Functions

In ZigBee Light Link the bridge device acts as a gateway between backend (Ethernet, Wi-Fi, 3G, etc.) and ZigBee network. In order to compile ZLLDemo for a bridge device APP_ZLL_DEVICE_TYPE parameter in application configuration.h file shall be set to APP_DEVICE_TYPE_BRIDGE.

In the Atmel ZLLDemo reference implementation the bridge is connected to a PC via serial interface. On the PC, the SLRemote desktop application should be installed, and configured to use corresponding virtual COM port (details are found in Section 5.7.2). Another option is to send commands to the bridge device connected to a PC manually, through a terminal program (see Section 5.5). The bridge device also serves as the OTAU server.

5.7.1 Network Joining

There are three ways to add a bridge device to a ZLL network:

1. *Form own network.* The bridge device may create its own network with given parameters. This can be accomplished by sending the `createNetwork` command to the bridge from console (see Section 5.5) or by using *Create network* option in the SLRemote GUI as described in Section 5.7.2.
2. *Automatically.* On startup, the bridge automatically starts searching for open ZLL networks and tries to join them.
3. *Through touchlink with a color scene controller.* In this case, bring the color scene controller close to the bridge, and initiate touch link procedure from the controller device. Once the touchlink is over, the color scene controller and the bridge will communicate using the manufacture-specific *Link Info* cluster. The color scene controller will send a command informing the bridge about the number of lights in the network. On receipt of this command, the bridge reads the attributes of the Link Info cluster containing information about the lights.

Once the bridge gets information about the lights in the network, it passes it to the connected PC, and the user can start using the GUI to control the lights.

5.7.2 Light Discovery and Control

When bridge and lights are successfully joined to the same network (as described in Section 5.7.1), the bridge shall discover the light devices in the network, in particular their network addresses and application endpoint IDs, as well as capabilities to be able to control them.

In the SLRemote GUI this can be accomplished by using *Discover Lights* button on the toolbar (see Section 5.7.2). Detected light devices will appear in the GUI. Click on the bulb icon corresponding to a light and send control commands by using the graphical interface from the pop-up menu as shown in Figure 5-4.

If serial console menu is used the light discovery shall be performed with `startDiscovery` command entered into terminal window. Network addresses and endpoint IDs for detected lights will be printed into the serial console and can be used to send light controlling commands as described in Section 5.5.

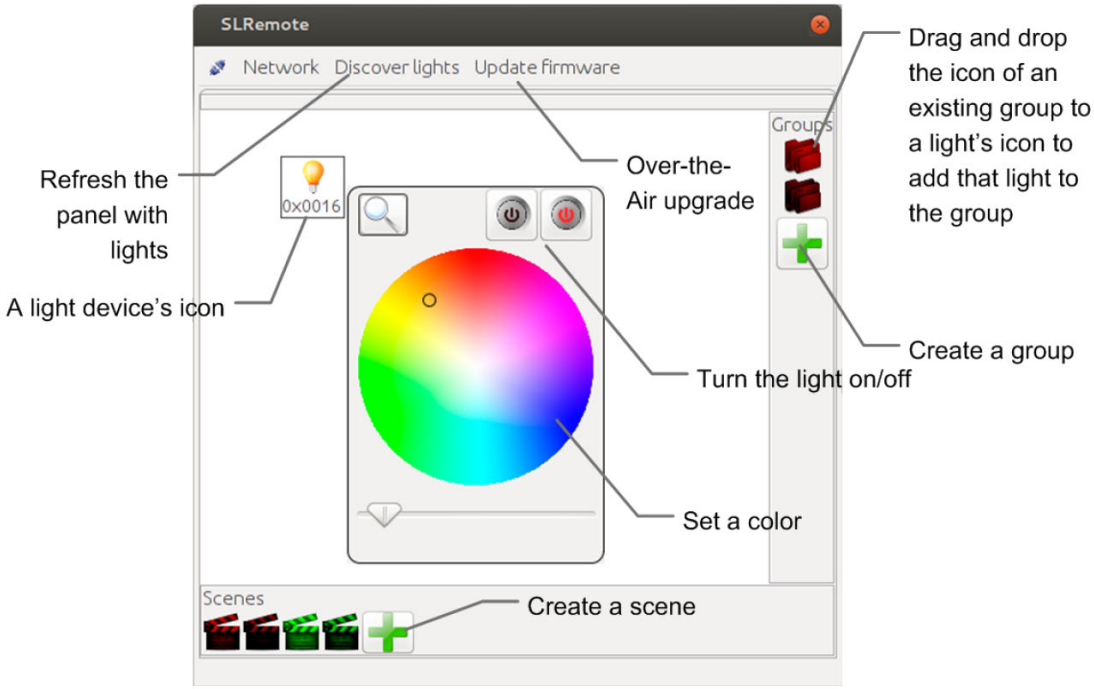
5.7.3 SLRemote GUI

BitCloud SDK includes the SLRemote GUI application for PC for communication with the bridge device connected serially to that PC. Through the GUI a user can organize nodes in groups, create scenes, and send commands to individual nodes to change brightness level or color, or turn a node on or off.

Basic usage of the SLRemote application is described below (see SLRemote's window on Figure 5-4):

1. Download and install Java® Runtime Environment [13], if not already installed on the PC.
2. Install the SLRemote application by launching the installation file provided with the SDK and following the instructions on the screen.
3. Launch the SLRemote's GUI once it is installed.
4. Connect the bridge device to the PC via serial cable.
5. In SLRemote, click the *Connect* button on the toolbar and set connection settings for the bridge (same as described in Section 5.5), specifying the COM port corresponding to it.
6. Associate the bridge device with a network by clicking *Network* button on the toolbar and then select either:
    a. *Create network* to make the bridge create a new network. To allow light devices to join it use *Permit Join* button.
    or
    b. *Discover network* to make the bridge device join an existing ZLL network.
7. Discover the lights present in the network by pressing *Discover lights* button on the SLRemote toolbar.
8. Light devices will appear in the GUI. Click the icon corresponding to a light and send commands by using the controls from the pop-up menu as shown on Figure 5-4.

Figure 5-4. SLRemote Application's Main Window and the Context Menu of a Light Device with Controls allowing sending Commands to that Light Device



5.8 Color Scene Controller's Functions

Color scene controller device in ZLLDemo reference application is implemented according the requirements for corresponding device type defined in ZigBee Light Link specification [29]. In order to compile ZLLDemo for a bridge device `APP_ZLL_DEVICE_TYPE` parameter in application `configuration.h` file shall be set to `APP_DEVICE_TYPE_COLOR_SCENE_REMOTE`. This device type represents a superset of functionalities of all other ZLL controller device types.

In ZLLDemo color scene remote controller device acts as a ZigBee end device and spends most of the time in sleep mode waiting for user input in order to send control commands. Such input can be triggered either by using on-board buttons on platforms that have them (see Section 5.8.2) or by using text commands to a serial console interface (see Section 5.5).

Color scene controller device can be connected to a network in two ways; using touchlink or using a classical ZigBee joining as described in Section 5.8.1.

It is also possible to perform firmware Over-the-Air Upgrade (OTAU) on color scene remote controller devices. For details, see Section 5.9.

5.8.1 Touchlink / Network Joining

For touchlink a color scene controller shall be brought close to a target device. Then the user shall initiate the touchlink procedure using on-board buttons (see Section 5.8.2). For example by pressing and holding PWR button on a Key Remote Control board with RCB256RFR2. Alternatively a `touchlink` serial console command can be used (see Section 5.5).

After that the touchlink procedure will be performed according to ZigBee Light Link specification [29].

Note: If touchlink between controller devices is not performed pairing a new controller device with a light that is already in the network will cause the light's leaving this network and form a new one.

To touchlink another color scene remote controller the procedure described above needs to be started simultaneously on both devices. Note that one of the controllers must have a factory new status.

A color scene controller can also be brought to an open ZLL network via classical ZigBee joining mechanism. To perform such joining the permit joining shall be enabled for the network (for example from a bridge device as described in Section 5.7.1) and then initiate actual joining using on-board buttons (see Section 5.8.2) or `nwkAssociation` console command.

If a color scene controller is brought to a network using classical ZigBee joining or touchlink with another remote then it is not able to control the light right away. It still needs to touchlink the lights in the network to pair to them.

5.8.2 Use of On-board Peripherals

Table 5-5 summarizes functionality of the board peripherals for a ZLLDemo light device. Additional information on HW setup is given in corresponding platform-specific sections as referenced in Section 3.3.

Table 5-5. Functionality of Board Peripherals for ZLLDemo Color Scene Controller Device

Development Board	LEDs Functionality	Buttons Functionality	Control via Serial Interface[1]
ATSAMR21-XPRO [34] (`BSP_SUPPORT` set to `BOARD_SAMR21_XPRO`)	N/A	For reset to factory new - hold the SW0 button within 1 second after reset is released and it must be held pressed for atleast 1 second	Via EDBG USB connection [2]. Set `APP_INTERFACE` to `APP_INTERFACE_USART` and `APP_USART_CHANNEL` to `USART_CHANNEL_1`
256RFR2 RCB on Key Remote Control board [16] (`BSP_SUPPORT` set to `BOARD_RCB_KEY_REMOTE`)	N/A For LCD indication see Section 5.8.2.2	See Section 5.8.2.3.	Via RS-232 connection on EXT header. Set `APP_INTERFACE` to `APP_INTERFACE_USART` and `APP_USART_CHANNEL` to `USART_CHANNEL_0`
256RFR2 RCB on STB [16] (`BSP_SUPPORT` set to `BOARD_RCB`)	N/A	N/A	Via USB connection. Set `APP_INTERFACE` to `APP_INTERFACE_USBFIFO`
AT256RFR2-XPRO [18] (`BSP_SUPPORT` set to `BOARD_ATMEGA256RFR2_XPRO`)	N/A	For reset to factory new - hold the SW0 button pressed when XPRO board is reset	Via DEBUG USB connection [2]. Set `APP_INTERFACE` to `APP_INTERFACE_USART` and `APP_USART_CHANNEL` to `USART_CHANNEL_1`

Notes: [1] – See Section 5.5 for configuration instructions.

[2] – Corresponding virtual COM port connection shall be used on PC side. Note that Flow Control shall be enabled on the PC when working with XPRO boards.

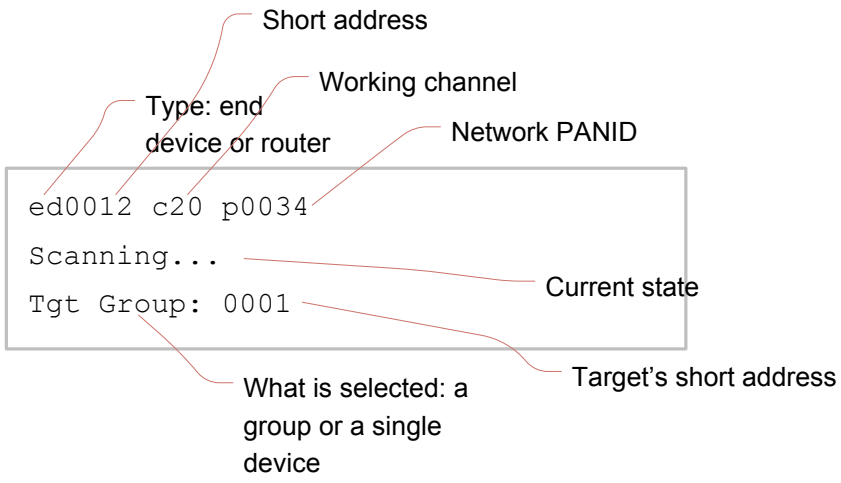
5.8.2.2 LCD Screen Output on Key Remote Control

When a Key Remote Control board is used for the color scene controller the application uses the LCD screen to output device's information:

- In the first line, network information is printed: `FN` and the active channel if the device is factory new, and device's type (end device or router, short address, working channel, and network PANID) otherwise
- In the second line, current application state is printed. In the idle state, it is `ZLL Remote`
- In the third line, information about the currently selected target is printed. That is, to which group or a single device a command will be sent

An example of LCD output with description of its components is shown in Figure 5-5.

Figure 5-5. Color Scene Controller's LCD Screen on a Key Remote Control Board



5.8.2.3 Button Functionality on Key Remote Board

Table 5-6 explains how to use buttons on a Key Remote Board to perform most common functionalities such as touchlink, reset to factory new, etc. Figure 5-6 depicts the key remote board keyboard and all commands that can be sent with the buttons from the color scene controller.

Table 5-6. Buttons for Executing Most Common ZLL Commands

Button(s)	Effect
PWR	Press and hold for three seconds in close proximity to the target device to perform touch link
PWR & R+	Press and hold for three seconds to perform classical ZigBee scanning
PWR & R-	Press and hold for three seconds to reset device to factory new state
PWR & R+ & R-	Press and hold for three seconds in close proximity to the target device to reset it to factory new state
L+/L-	Light's on/off
Up/Down	Increase/decrease light level
Left/Right	Increase/decrease saturation
Colored buttons	Set the corresponding color using the following values: Red: hue = 60000 or x = 40000, y = 20000 Green: hue = 30000 or x = 10000, y = 40000 Yellow: hue = 15000 or x = 30000, y = 30000 Blue: hue = 45000 or x = 10000, y = 10000
SEL	Select the next bound device and requests it to identify itself. This allows sending unicast commands to a single device. Groupcast mode will be entered automatically after five seconds of inactivity.
1/2/3	Store Scene if pressed for more than three seconds and Recall scene if pressed for less than three seconds
7/8/9	Set minimum/middle/maximum light level

The SEL button may be used to select a light. When this button is pressed and released, the next light becomes selected. This means that all commands initiated by the user (via button presses) will be sent to this light in a unicast manner. If no buttons are pressed for five seconds the remote will switch back to groupcast mode. So if unicast control for a particular light is needed for extended amount of time the intervals between consecutive button presses shall be not less than five seconds.

To reset a color scene controller device to the factory new state press the PWR button while holding the R- button and wait for three seconds. The LED located at the bottom of the RCB board will blink single time to indicate that the device has been reset to the factory new state.

Each button may be used to send up to four commands. What command is send by pressing a button depends on whether buttons R+ and R- are also pressed or not as shown in Table 5-7.

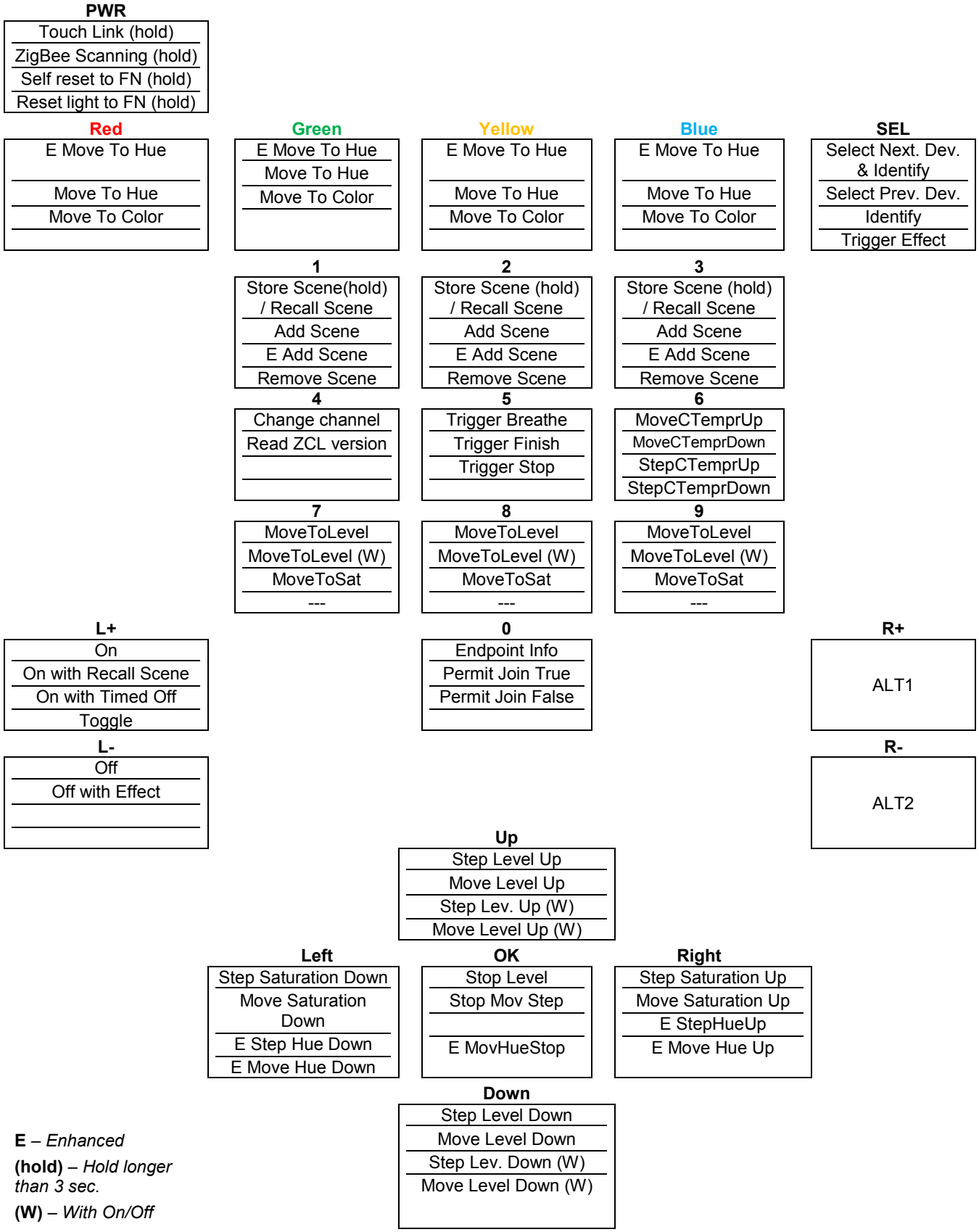
Table 5-7. Alternating a Command by holding R+ and R- Buttons

Number of Command	R+ and R- Buttons State
First command	Both are not pressed
Second command	R+ is pressed, R- is not pressed
Third command	R+ is not pressed, R- is pressed
Fourth command	Both are pressed

For example, to send a Toggle command press the L+ button while holding both the R+ and R- buttons pressed.

For buttons 1, 2, and 3: if a button is pressed for more than three seconds (when both R+ and R- are not pressed) a Store Scene command is sent; if a button is pressed for less than three seconds a Recall Scene command is sent.

Figure 5-6. Color Scene Controller's Keyboard Scheme



E – *Enhanced*

(hold) – *Hold longer than 3 sec.*

(W) – *With On/Off*

5.9 Over-the-Air Firmware Update

Nodes in a ZLL network can be updated with new firmware over the air. For this purpose, standard Over-the-Air Upgrade (OTAU) cluster is used.

This section gives an overview on how to configure ZLLDemo application with OTA support and how to perform OTA on remote devices. For more details on OTAU, refer to [24].

In ZLLDemo a Bridge device acts as OTAU server that provides new firmware images to other devices in the network that act as OTAU clients. SLRemote GUI application (see Section 5.7.2) running on the PC provides an example of the user interface for controlling OTAU process. Note however that SLRemote application is meant only for demo purposes.

To demonstrate OTAU procedure the ZLLDemo reference application shall be compiled with application (see Table 5-2) and ConfigServer stack parameters properly set in the application `configuration.h` file.

5.9.1 OTAU Configuration for ZLL Bridge

In ZLLDemo a Bridge device acts as OTAU server.

First of all, the `APP_USE_OTAU` flag shall be set to `1` to enable use of OTAU cluster and corresponding services (image storage driver, etc.). If OTAU is enabled, the bridge is by default configured to act as an OTAU server with `#define OTAU_SERVER` enabled and `#define OTAU_CLIENT` disabled.

Serial console and logging need to be enabled by setting `APP_DEVICE_EVENTS_LOGGING` and `APP_ENABLE_CONSOLE` parameters to `1`. Additionally `APP_USE_ISD_CONSOLE_TUNNELING` shall be set to `1` to combine serial console support with simultaneous operation of Image Storage Driver (ISD) on the same connection to the PC.

For more details on OTAU configuration in application and stack, see [24].

5.9.2 OTAU Configuration for ZLL Light and Controller

At least one light or a remote controller device shall be compiled as an OTAU client.

First of all, the `APP_USE_OTAU` flag shall be set to `1` to enable use of OTAU cluster and corresponding services (flash driver, etc.). If OTAU is enabled, a non-bridge device is by default configured to act as an OTAU client with `#define OTAU_CLIENT` enabled and `#define OTAU_SERVER` disabled.

If real external flash is used on the board then, in order to be able to switch to downloaded firmware, the device shall be programmed with an OTA bootloader [12] configured to support corresponding flash memory. If the application is configured to use fake flash memory (`APP_USE_FAKE_OFD_DRIVER` set to `1`) the presence of OTA bootloader is optional as firmware image won't be stored on external memory anyway.

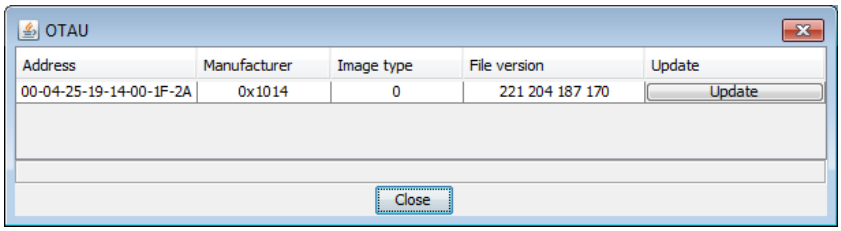
HW configuration of external flash memory (used to store application images during OTAU) is described in platform-specific appendices: B.1.4.

For more details on OTAU configuration in application and stack, see [24].

5.9.3 OTAU Procedure

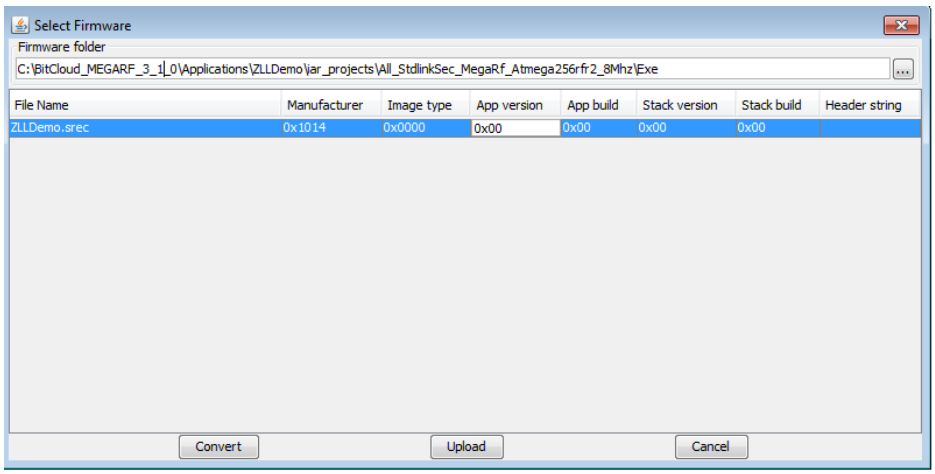
Bridge device shall be first connected via a serial interface to a PC. The user should launch the SLRemote application on the PC and bring Bridge as well as other ZLL devices into the same network as described in Section 5.7.2. After selecting the *Update firmware* item on the toolbar the dialog window will appear displaying information on devices in the network with OTAU support. An example is shown in Figure 5-7.

Figure 5-7. Selecting Target Device for OTAU



Among the displayed nodes select the target device which needs to be updated and press the *Update* button next to it. Another dialog window will open asking to provide the path to the folder with new firmware image either in *.srec* or *.zigbee* format as shown on Figure 5-8.

Figure 5-8. Selecting Target File for OTAU

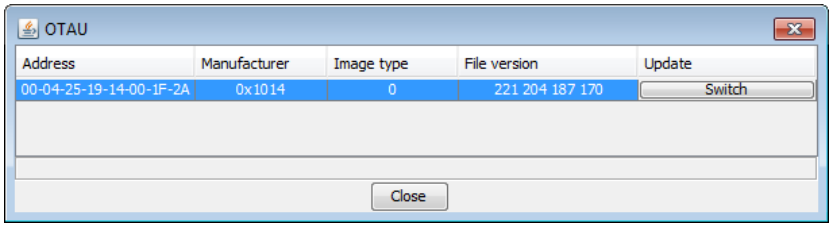


Among listed files select the one to be uploaded to the target device.

Note: Parameter values for the .srec file (Manufacturer ID, Image Type, App version, and build, etc.) can be modified via the GUI. If .srec file is selected for the upload the SLRemote will automatically apply them. Alternatively selected .srec file can also be updated to .zigbee format using *Convert* button.

To start the OTAU process, press the *Upload* button and the view will return to the dialog shown in Figure 5-7 with upload progress shown in the Update column. After file is successfully uploaded to the device the Update column for this device will turn into *Switch* button as shown on Figure 5-9. By pressing it the bridge will send a special ZCL command to the device instructing it to switch to the uploaded firmware.

Figure 5-9. Switching to Uploaded Firmware on the Target Device



5.10 Interoperability with ZHA Networks

ZLL devices are interoperable with ZigBee Home Automation (ZHA) networks.

If the `APP_SCAN_ON_STARTUP` parameter is set to `1` in the `configuration.h` file on a light then, on startup, it will scan channels and try to join discovered networks. All devices hold HA network key and link keys, which are required to be authenticated in an HA network.

To make a color scene controller join an HA network, press PWR and R+ buttons on the Key Remote Control board and hold for more than three seconds. Once both a color scene controller and a light join an HA network, perform touchlink between them and use the color scene controller to control the light. Commands can also be sent to the light from the HA dimmer switch device.

5.11 Running Certification Test Scripts

BitCloud SDK includes certification test scripts written in Python, which can be used to test the application against certification scenarios. Certification scripts are executed through the PC application WSNRunner also provided with the SDK. To use the test scripts the user shall first install the WSNRunner application (Section 5.11.2) and then run certification scripts with its help from the command line (Section 5.11.3).

5.11.1 Prerequisites

For execution of certification test scripts the demo application must be compiled with the `APP_ENABLE_CERTIFICATION_EXTENSION` parameter set to 1 in the `configuration.h` file. Other prerequisites are:

- Types of ZLL devices: extended color lights (up to six devices) and remote controls (up to two devices). For the first one, set `APP_ZLL_DEVICE_TYPE` to `APP_DEVICE_TYPE_EXTENDED_COLOR_LIGHT`. For the second, set `APP_ZLL_DEVICE_TYPE` to `APP_DEVICE_TYPE_COLOR_SCENE_REMOTE`.
- Boards: RCB with Sensor Terminal Board, or Key Remote Control Board
- For testing interoperability with Home Automation devices, one HA device of each of the following types are required: dimmer switch, dimmable light, and occupancy sensor. For information on obtaining firmware for these devices, contact Atmel support.

5.11.2 WSNRunner Setup

To install WSNRunner follow the instructions below:

1. Download and install Java Runtime Environment [13], if not already installed on the PC.
2. Download and install Jython [14].
3. Launch `WSNRunnerSetup.exe` located in the `.\Evaluation Tools\` folder of the BitCloud SDK, and follow the installation instructions.
    a. When prompted for Jython path, specify the folder where Jython was installed in Step 2.
    b. When prompted for Commands path, specify any path (not used by ZLL scripts).
    c. When prompted for Scripts path, specify any path (not used by ZLL scripts).
4. Add the path to the installed WSNRunner application to the system `PATH` environment variable. For that, go to `Control Panel > System > Advanced > Environment Variables`, select `Path` from the `System variables` list, click `Edit`, and append "`;`" followed by the actual path to the WSNRunner directory (by default `C:\Program Files\Atmel\WSNRunner`), then click `OK`.
5. Go to the `C:\Documents and Settings\<account name>\.config\` directory. Open the `wsnrunner.properties` file in any text editor.
6. In this file, you can:
    a. See and correct paths that were specified during installation (be careful; there should be no spaces at the end of the path strings).
    b. Add variables to be passed to the scripts; channel mask, for instance.
    c. Specify COM ports to be used for communication with devices.

Note: When installing WSNRunner environment for the first time, only paths will be present in this file. The chmask (for channel mask) and nodes variables need to be added manually. It is recommended that you copy `wsnrunner.properties-example` from the installation directory to `C:\Documents and Settings\<account name>\.config\wsnrunner.properties`. In case the WSNRunner application is reinstalled, the properties file is saved and is not replaced.

5.11.3 Run a Script from the Command Line

The WSNRunner application includes both the GUI version and the command line version. For execution of test scripts the command line version is used.

Once the WSNRunner is installed, to run a script, follow the instructions:

1. Configure the `nodes` parameter in the `wsnrunner.properties` file by setting this parameter with a comma-separating list of connection settings for all devices. For example, to be able to communicate with devices connected to the COM1 and the COM3 port, add the line:

   ```
   nodes = COM:COM2, COM:COM3
   ```

2. Open a test script in a text editor and modify the lines starting with `@connection` by adding, after a comma, `IDX<N>` pointing to the corresponding element with the index `<N>` in the nodes list (starting from 0), for example:

   ```
   @connection r1 = router, IDX0
   ```

   It is also possible to specify connection settings directly in the test script, for example:

   ```
   @connection r1 = router, COM:COM2
   ```

   Note that `router` lines should correspond to light devices, and `enddevice` lines to color scene controllers.
3. In the command line execute:

   ```
   runner.exe <script_name>.py
   ```

   or, to redirect the output and error log to the `1.txt` file (choose any name),

   ```
   runner.exe <script_name>.py >1.txt 2>&1
   ```

WSNRunner will load the `wsnrunner.properties` file and, using connection setting specified in this file, will execute the test script. Information about sent commands and received responses will be outputted to the console window or redirected to the specified files, as described in Step 3.

6. ZigBee Home Automation Reference Application

HADevice is a reference implementation of the ZigBee Home Automation profile [8]. It contains the following HA device types:

- Combined interface
- Dimmer switch
- Dimmable light
- Multi-sensor device that can combine following sensors types:
  - Occupancy sensor
  - Temperature sensor
  - Light sensor
  - Humidity sensor
- Thermostat
- IAS ACE

Other device types can be also implemented as described in BitCloud Developer's Guide [4].

To compile the application for a specific device type, the corresponding `APP_DEVICE_TYPE_<name>` parameter (see Section 6.3.1) must be enabled in the `configuration.h` file located in the application's root directory (see Table 3-1).

Particular sensor functionality for the multi-sensor device can be selected using `APP_SENSOR_TYPE_<sensorType>` defines (e.g. `APP_SENSOR_TYPE_OCCUPANCY_SENSOR, APP_SENSOR_TYPE_TEMPERATURE_SENSOR`). Note that several sensor functionalities can be selected.

The combined interface device serves as the network coordinator and the network trust center. On startup, the combined interface forms a network and other devices try to find an open HA network to join.

The user should send commands from a terminal emulator program (like HyperTerminal or RealTerm) on a PC connected to the devices via the serial interface (see Section 6.1 for details on starting of the application and serial connection settings).

6.1 Launching the Demo

To start the application, follow these instructions:

1. Assemble devices as instructed in Section 3.3.
2. Program devices with firmware images. The pre-built images are located in the `\Evaluation Tools\HADevice` directory.

Note: Prebuilt firmware images are built for *8MHz frequency*.

Note: Before programming, make sure the fuse bits are installed correctly (see Section B.3.1).

3. To send commands to a device and observe the device's output, connect the device to a PC; launch a terminal emulator (for example, RealTerm or HyperTerminal) on the PC, and point the terminal emulator to the COM port corresponding to the device. Use the following setting for the serial connection:

   `BAUD RATE:` `38400`
   `PARITY:` `None`
   `DATA BITS:` `8`
   `STOP BITS:` `1`
   `FLOW CONTROL:` `None` `(Hardware` for the Xplained-PRO boards`)`

   The console commands can be sent following their syntax as described in Section 6.4.
4. Power up combined interface device.
5. Start EZ-Mode on the combined interface device using **`startEZMode`** command in serial console. This will open network for joining of new devices.The CI will act as both Initiator and Target.

6. Power up another HA device and wait until it joins the network.
7. Perform device pairing. For this EZ-Mode shall be initiated on both devices within three minutes interval (using `startEZMode` command in serial console). After three minutes the EZ-mode expires and pairing won't be done. Based on the supported clusters (see Section 6.2) following pairings are possible by default in HADevice application:
   - a dimmable light to the combined interface;
   - an occupancy sensor to the combined interface;
   - a dimmer switch to a dimmable light;
   - a thermostat to the Combined Interface;
   - an ACE device to the Combined Interface;
8. After devices are paired console commands can be used to communicate between them (see Section 6.4).

For demonstration of Over-the-Air upgrade functionality, see Section Table 6-6.

6.2 Supported Clusters

Table 6-1 lists server and client clusters from ZigBee Cluster Library supported by the HA reference application for different device types. Other clusters can be added as described in BitCloud Developer's Guide [4].

Table 6-1. Clusters Supported by HA Devices

Device Type	Server Clusters	Client Clusters
Combined interface	Basic Identify Time IAS ACE OTAU (if enabled)	Basic Identify OnOff Level Control Groups Scenes Occupancy Sensing Temperature Measurement Relative Humidity Measurement Illuminance Measurement Thermostat Thermostat UI Diagnostics Alarms Power Fan Control IAS Zone
Dimmer switch	Basic Identify	Identify OnOff Level Control OTAU (if enabled)
Dimmable light	Basic Identify OnOff Level Control Groups Scenes	OTAU (if enabled)

Device Type	Server Clusters	Client Clusters
Multi-sensor device	Basic Identify Occupancy Sensing (if enabled) Temperature Measurement (if enabled) Relative Humidity Measurement (if enabled) Illuminance Measurement (if enabled)	Identify OTAU (if enabled)
Thermostat	Basic Identify Thermostat ThermostatUiConf OccupancySensing Diagnostics Alarms HumidityMeasurement Fan Control Groups Scenes TemperatureMeasurement	Occupancy Sensing Time Humidity measurement Temperature measurement OTAU (if enabled)
IAS ACE	Basic Identify IAS Zone Diagnostics	Identify IAS ACE OTAU (if enabled)

6.3 Source Code Organization

Application projects and source code are located in the `\Applications\HADevice` folder inside the SDK. The source code is divided into the common part and device-specific code. The entry `main()` function is located in the `zclDevice.c` file while device-specific initialization and endpoint registration for communication between clusters is registered in the device-specific .c file located in the device folder.

Supported clusters for each device are configured in `<deviceName>Clusters.c` files. A separate source code file is provided for each cluster supported by a specific device. Such file initializes structures needed for the cluster and implements callback functions that are called to indicate commands' responses. For example, see the `dlIdentifyCluster.c` file, which initializes the identify cluster for the dimmable light device.

The application's configuration is set in the `configuration.h` file located in the `\Applications\HADevice` folder. Serial interface used by the device to send information to a PC is also configured in this file. Additionally, in the application's source code UART is configured in the `\Applications\HADevice\common\src\uartManager.c` file.

6.3.1 Configuration

The HA reference application's configuration is set in the `configuration.h` file located in the root application's directory and the `appConsts.h` file located in the application's `\common\include\` directory. Key parameters are listed in 0. Once any of these parameters is changed, the application must be rebuilt and the device's firmware updated for changes to take effect.

Table 6-2. Key Configuration Parameters of the HA Reference Application

Parameter Name	Description
APP_DEVICE_TYPE_*<deviceType>*	Specifies the device type for which a reference application shall be compiled. COMBINED_INTERFACE, DIMMABLE_LIGHT, DIMMABLE_SWITCH, THERMOSTAT, IAS_ACE or MULTI_SENSOR shall be put in place of <deviceType> to define the device type.
BSP_SUPPORT	Specifies the board that will be used by the application. On-board peripherals such as buttons, LEDs, LCD, MAC address reading, etc. will be configured and compiled according to the made selection. User can extend BSP options to support its custom board. BOARD_FAKE option can be used for testing network communication as it substitutes BSP API with stub functions.
APP_ENABLE_CONSOLE	Specifies whether or not serial console commands are supported in the application. If set to 1 then reception of serial commands is enabled on defined APP_INTERFACE. Also in such case sleep on end devices is automatically disabled.
APP_DEVICE_EVENTS_LOGGING	Configures events logging. If set to 1 then application will print information on application's events to serial port defined via APP_INTERFACE.
APP_DEVICE_TYPE	The parameter determines the type of ZigBee device for the reference application (for example, coordinator, router, end device). Defined in appConsts.h.
APP_INTERFACE	Configures the serial interface used to connect the device to a PC. Available options depend on the platform and are listed in the configuration.h file. Values are of APP_INTERFACE_<name> format. For some interfaces (UART) additional parameters should be set such as APP_USART_CHANNEL.
APP_USE_OTAU	Set to 1 to enable OTAU support in the application and set to 0 to disable it
OTAU_CLIENT OTAU_SERVER	Defines whether device acts as OTAU client (device that will be upgraded) or OTAU server (device that will provide the new firmware). Only one role shall be selected. By default combined interface device acts as OTAU server and other devices as OTAU clients. Applicable only if APP_USE_OTAU is set to 1.
APP_USE_ISD_CONSOLE_TUNNELING	Support simultaneous usage of the same serial interface for passing (1) commands from console and (2) commands exchanged by the ISD driver and the bootloader PC tool. This parameter is valid for the OTAU server (the bridge device).
APP_USE_FAKE_OFD_DRIVER	Enables fake implementation of the OFD driver on OTAU client devices. This may be useful for testing OTAU on boards without external flash memory.
APP_SUPPORT_OTAU_PAGE_REQUEST	Configures use of OTAU image page request feature on OTA client devices (refer to [24] for details).
EXTERNAL_MEMORY	Specifies the type of external memory (where the new firmware image will be stored).
APP_ZAPPSI_INTERFACE	Defines primary serial interface type to be used by ZAppSI (for example APP_INTERFACE_USART).
APP_ZAPPSI_MEDIUM_CHANNEL	Defines particular channel of the serial interface used by ZAppSI (for example USART_CHANNEL_1).
APP_FRAGMENTATION	If set to 1 then stack parameters are automatically configured to support APS-level fragmentation.
APP_ZONE_TABLE_SIZE	Enable and define the size of the Zone Table used as part of CI (IAS CIE device functionality). By default the zone table size is defined as 3.

6.4 Serial Console Commands

Console commands are sent over a serial interface from a terminal program on a PC to a node connected to that PC. Table 6-3 presents console commands supported by all HA device types.

Table 6-3. Console Commands Supported by *all* Device Types

Command Syntax	Description
help	Displays help instructions.
reset	Reset the device. Network and application parameters will be restored from non-volatile memory upon reset.
startEZMode	Start ezMode depending whether target or initiator. Device will broadcast ZDO Permit Join command to open the network for joining, if it can act as a target will set itself into identify mode and if able to act as initiator will broadcast ZCL identifyQuery command to find the peer node. Once peer node is found device will perform cluster discovery and binding to target clusters.
resetToFN	Reset to factory new settings; data saved in the non-volatile memory is deleted, and the device restarts with compile-time settings.
getDeviceType	Request for device type. Returns following string: "DeviceType = %d\r\n", with %d being 0x02 for HA coordinator, 0x03 for HA router and 0x04 for HA end device.
powerOff	Emulate powers off of the device by disabling RF.
setPermitJoin <dur>	Sets Permit Join to a given duration in seconds.
restartNwk <channel>	Restarts network on a given channel.

Table 6-4 lists commands supported only by combined interface device. Commands that result in over the air transmissions can be sent either as unicast frames, to specific devices, to bound devices.

Table 6-4. Additional Serial Console Commands Supported by the C*ombined Interface* Device

Command Syntax	Description
setEzModeType <type>	Sets EZ-Mode type: 0 - target, 1 - initiator
EzModeInvoke <addrMode> <addr> <ep> <action>	Send EZ-Mode Invoke command
readAttribute <addrMode> <addr> <ep> <clusterId> <attrId>	Read value of specified attribute on specified cluster from remote device
writeAttribute <addrMode> <addr> <ep> <clusterId> <attrId> <type> <attrValue> <attrSize>	Send Write Attribute command for specified attribute and cluster
updateCommissioningState <addrMode> <addr> <ep> <action> <mask>	Send Update Commissioning State command
readBasicAttr <addrMode> <addr> <ep> <attrId>	Read a specified Basic cluster's attribute from a specified destination
writeBasicAttr <addrMode> <addr> <ep> *<attrId> <type> <attrValue> <attrSize>*	Write a Basic cluster's attribute with the specified attribute ID, type, value, and size
identify <addrMode> <addr> <ep> *<identifyTime>*	Send the Identify command
identifyQuery <addrMode> <addr> <ep>	Send the Identify Query command
onOff *<addrMode> <addr> <ep> <"-on"/"-off">*	Turn the specified light device(s) on or off

Command Syntax	Description
moveToLevel *<addrMode> <addr> <ep> <level> <transitionTime> <onOff>*	Send the Move To Level (with On/Off) command
move *<addrMode> <addr> <ep> <rate> <onOff>*	Send Move (with On/Off) command
step *<addrMode> <addr> <ep> <mode> <stepSize> <transitionTime> <onOff>*	Send the Step (with On/Off) command
stop *<addrMode> <addr> <ep> <onOff>*	Send the Stop (with On/Off) command
addGroup *<addrMode> <addr> <ep> <groupId>*	Send the Add Group command
viewGroup *<addrMode> <addr> <ep> <groupId>*	Send the View Group command
getGroupMembership *<addrMode> <addr> <ep> <count> <groupId1> <groupId2> <groupId3> <groupId4> <groupId5>*	Send the Get Group Membership command. *<count>* specifies how many group IDs following it should be considered, but five values must be provided as group IDs always.
removeGroup *<addrMode> <addr> <ep> <groupId>*	Send the Remove Group command
removeAllGroups *<addrMode> <addr> <ep>*	Send the Remove All Groups command
addGroupIfIdentifying *<addrMode> <addr> <ep> <groupId>*	Send the Add Group If Identifying command
addScene *<addrMode> <addr> <ep> <groupId> <sceneId> <transitionTime> <onOff> <level>*	Send the Add Scene command
viewScene *<addrMode> <addr> <ep> <groupId> <sceneId>*	Send the View Scene command
removeScene *<addrMode> <addr> <ep> <groupId> <sceneId>*	Send the Remove Scene command
removeAllScenes *<addrMode> <addr> <ep> <groupId>*	Send the Remove All Scenes command
storeScene *<addrMode> <addr> <ep> <groupId> <sceneId>*	Send the Store Scene command
recallScene *<addrMode> <addr> <ep> <groupId> <sceneId>*	Send the Recall Scene command
getSceneMembership *<addrMode> <addr> <ep> <groupId>*	Send the Get Scene Membership command
configureOsReporting *<addrMode> <addr> <ep>* <min> <max>	Configure reporting of the occupancy sensor with <min> and <max> values
getDeviceType	Lists the type of the device type configured
setPermitJoin <duration>	Duration for the Permit join when the device is in EzMode
thermSetPointChange <*addrMode> <addr> <ep> <setPtmode> <amount***>**	Sends thermostat Set Point Change Command along with the mode and amount
setUTCTime <*dd:mm:yr:hr:min:sec***>**	Sets the UTC time
readTime	Reads the current time
setTimeZoneAndDST <*timeZone> <dstStart> <dstEnd> <dstShift>*	Sets time zone and DST
setTimeStatus <*timeZone> <dstStart> <dstEnd> <dstShift>*	Sets the time status of the device
resetAlarm <*addrMode > <addr> <ep> <ClusterId> <alarmCode>*	Resets the alarm with its code

Command Syntax	Description
alarmCmd <<*addrMode > <addr> <ep> < resetAllAlarm/getAlarm/resetAlarmLog>*	Alarm Command which could be resetAllAlarm/getAlarm/resetAlarmLog
ACEGetPanelStatusChangedCommand <*addrMode > <addr> <ep> <panel_status> <seconds_remaining> <audible_noti> <alarmstatus>*	Sends Panel Status Changed Command to the Ace device with its status and seonds remaining for the change with actions to be taken w.r.t alarmstatus and its audible notification
ACEZoneStatusChangedCommand <*addrMode > <addr> <ep> <zoneId> <zone_status> <audible> <zone_label>*	Sends Zone status Changed command to the Ace device with its status along with audible details and zone label
ZoneInitiateNormalOperatingModeCommand <*addrMode > <addr> <ep>*	Sends to initiate Normal Operating mode at the Ace device
ZoneInitiateTestModeCommand <*addrMode > <addr> <ep> <Test_Mode_Duration> <Current_Zone_Sensitivity Level>*	Sends to initiate Test operating Mode at the Ace device with its dutaion for the same with its current zone sensitivity level

Table 6-5. Additional Serial Console Commands Supported by the *Dimmer Switch* Device

Command Syntax	Description
onOff *<addrMode> <addr> <ep> <"-on"/"-off">*	Turn the specified light device(s) on or off
moveToLevel *<addrMode> <addr> <ep> <level> <transitionTime> <onOff>*	Send the Move To Level (with On/Off) command
move *<addrMode> <addr> <ep> <rate> <onOff>*	Send Move (with On/Off) command
step *<addrMode> <addr> <ep> <mode> <stepSize> <transitionTime> <onOff>*	Send Step (with On/Off) command
stop *<addrMode> <addrMode> <addr> <ep> <onOff>*	Send Stop (with On/Off) command

Table 6-6. Additional Serial Console Commands Supported by the Thermostat Device

Command Syntax	Description
setOccupancy *<0- UnOccupied / 1- Occupied>*	Sets the Occupancy state either to Occupied / UnOccupied
clusterAttrInitDefault <*ClusterID>*	Initializes all attributes to default values
setOccupancyState *<state>*	Sets the Occupancy state wither to Occupied or UnOccupied
setOccupancySensorType *<SensorType>*	Sets the Occupancy Sensor Type
triggerAlarm *<clusterId> <alarmCode> <Raise/Clear>*	Triggers Alarm State either to Raise /Clear with the AlarmCode
setAlarmMask *<clusterId> <alarmMask>*	Sets the AlarmMask
readAttribute <addrMode> <addr> <ep> <clusterId> <attrId>	Read value of specified attribute on specified cluster from remote device
writeAttribute <addrMode> <addr> <ep> <clusterId> <attrId> <type> <attrValue> <attrSize>	Send Write Attribute command for specified attribute and cluster

Table 6-7. Additional Serial Console Commands Supported by the ACE Device

Command syntax	Description
IASACEArmCommand *<addrMode> <addr>* <ep> *<ArmMode> <Arm/Code> <ZoneId>*	Sends IAS ACE Arm command to the CI (CIE) along with Arm mode with its code to the zoneId
IASACEBypassCommand *<addrMode> <addr>* <ep> *<zone_numbers> <zone_id1> <zone_id2> <zone_id3> <arm_code_code>*	Sends IAS ACE Bypass command with thte list of zoneIds along with the code to the CI (CIE)
IASACEEmergencyCommand *<addrMode> <addr>* <ep>	Sends IAS ACE Emergency command to the CI (CIE)
IASACEFireCommand *<addrMode> <addr>* <ep>	Sends IAS ACE Fire command to the CI (CIE)
IASACEPanicCommand *<addrMode> <addr>* <ep>	Sends IAS ACE Panic command to the CI (CIE)
IASACEGetZoneIdMapCommand *<addrMode> <addr>* <ep>	Sends IAS ACE GetZoneIdMap command to the CI (CIE)
IASACEGetZoneIdInformationCommand *<addrMode>* <addr> <ep> <zone_id>	Sends IAS ACE GetZoneIdInformation Command to get the complete information of the zoneId
IASACEGetPanelStatusCommand *<addrMode> <addr>* <ep>	Sends IAS ACE GetPanel Status Command to the CI (CIE) to get the Panel's status
IASACEGetZoneStatusCommand *<addrMode> <addr>* <ep> <starting_zone_id> <max_number_zoneids> <zonestatus_maskflag> <zonestatus_mask>	Sends IAS ACE GeZone Status Command to the CI (CIE) to get the Zone Status. The Command would list the starting zoneId with the maximum number to read from there.
ZoneEnrollReqCmd *<addrMode> <addr>* <ep> <zonetype> <manuf_code>	Sends IAS Zone Enroll Request Command to the CI (CIE) as part of Zone Enrollment.
ZoneStatusChangeNotifiCmd *<addrMode> <addr>* <ep> <zonestatus> <ext_status>	Sends IAS Zone StatusChange, a notification to the CI (CIE)
ZoneStatusChange <*Device_Spec_Bits***>**	To change the internal zone status to trigger a change notification to the CI (CIE)
GetByPassZoneList *<addrMode> <addr>* <ep>	Sends IAS ACE GetByPassZone list which were already bypassed and part of the bypass table

6.5 Over-the-Air Firmware Update

Nodes in a ZHA network can be updated with new firmware over the air. For this purpose, standard Over-the-Air Upgrade (OTAU) cluster is used.

This section gives an overview on how to configure HADevice application with OTA support and how to perform OTA on remote devices. For more details on OTAU refer to [24].

In HADevice application Combined Interface device acts as OTAU server that provides new firmware images to other devices in the network. PC Bootloader GUI application [12] provides an example of the user interface for controlling OTAU process. Note however that this GUI application is meant only for demo purposes.

To demonstrate OTAU procedure the HADevice reference application shall be compiled with application (see 0) and ConfigServer stack parameters properly set in the application `configuration.h` file.

6.5.1 OTAU Server Configuration

In HADevice application Combined Interface device acts as OTAU server.

First of all, the `APP_USE_OTAU` flag shall be set to `1` to enable use of OTAU cluster and corresponding services (image storage driver, etc.). If OTAU is enabled, the combined interface device is by default configured to act as an OTAU server with `#define OTAU_SERVER` enabled and `#define OTAU_CLIENT` disabled.

Serial console and logging need to be enabled by setting `APP_DEVICE_EVENTS_LOGGING` and `APP_ENABLE_CONSOLE` parameters to `1`. Additionally `APP_USE_ISD_CONSOLE_TUNNELING` shall be set to `1` to combine serial console support with simultaneous operation of Image Storage Driver (ISD) on the same connection to the PC.

For more details on OTAU configuration in application and stack, see [24].

6.5.2 OTAU Client Configuration

At least one device shall be compiled as an OTAU client.

Same as for the OTAU server the `APP_USE_OTAU` flag shall be set to `1` to enable use of OTAU cluster and corresponding services (flash driver, etc.). If OTAU is enabled, a non-combined interface device is by default configured to act as an OTAU client with `#define OTAU_CLIENT` enabled and `#define OTAU_SERVER` disabled.

If real external flash is used on the board then in order to be able to switch to downloaded firmware the device shall be programmed with an OTA bootloader [12] configured to support corresponding flash memory. If application is configured to use fake flash memory (`APP_USE_FAKE_OFD_DRIVER` set to `1`) then presence of OTA bootloader is optional as firmware image won't be stored on external memory anyway.

HW configuration of external flash memory (used to store application images during OTAU) is described in platform-specific appendices: B.1.4.

For more details on OTAU configuration in application and stack, see [24].

6.5.3 OTAU Procedure

Comined Interface device shall be first connected via a serial interface to a PC. Using serial console commands the user shall bring Combined Interface interface and other HA devices into the same network as described in Section 6.1.

After that PC Bootloader GUI application [12] shall be started. Procedure on how to use it for uploading firmware to the devices is described in [24].

7. WSNDemo Application

7.1 Overview

The network and radio frequency performance of the hardware components is demonstrated with the WSNDemo application, which is based on the Atmel BitCloud API. This application consists of the embedded firmware, which supports functions for coordinator, router, and end device, and the GUI visualization application, WSNMonitor, which is run on a PC. In WSNDemo, the nodes communicate based on a proprietary messaging protocol.

The SDK includes the WSNMonitor PC application in binary format (described in Section 7.4), and the WSNDemo embedded application is available in binary format and source code.

The source code for the WSNDemo application can be modified and extended, making it possible to develop WSN applications for a variety of application scenarios.

End devices, routers, and the coordinator devices emulate the sensor data reading for light and temperature sensors, and forward collected data to the WSNMonitor application for visualization.

End devices follow a duty cycle (that is, the microcontroller and radio transceiver are put to sleep periodically) and wake up to transmit data to the coordinator. Using the serial connection, the coordinator transmits the received packets, along with its own sensor data (or emulated sensor data), to the WSNMonitor application. Those transmitted values are displayed on WSNMonitor panes as temperature, light, and battery level measurements.

WSNMonitor also visualizes network topology by drawing a tree of nodes that have joined the network. For each of the nodes, parameters like node address, node sensor information, and link quality data are displayed.

RSSI indicates a link's current condition and is measured in dBm. The RSSI resolution is 3dBm. *LQI,* a numeric parameter defined within the 0 to 255 range, is used to measure the link quality. Larger values mean a better link, while values close to zero indicate a poor connection.

In WSNDemo, the number of routers and end devices is limited only by the network parameter settings.

7.2 Launching the Demo

To start WSNDemo, proceed as follows:

1. Assemble devices as instructed in Section 3.3.
2. Program devices with firmware images. One node shall be programmed as coordinator, others as routers or end devices.
   The pre-built firmware images are located in the `\Evaluation Tools\WSNDemo (Embedded)` directory.

Note: Prebuilt firmware images are built for *8MHz frequency*.

Note: Before programming, make sure the fuse bits are installed correctly (see Section B.3.1).

3. Connect the coordinator node to the PC using the serial interface.
4. Run WSNMonitor (see Section 7.4).
   Use the following setting for the serial connection of the WSNMonitor:
   `BAUD RATE:` 38400
   `PARITY:` None
   `DATA BITS:` 8
   `STOP BITS:` 1
   `FLOW CONTROL:` None (`Hardware` for the Xplained-PRO boards)
5. Observer coordinator node in the WSNMonitor.
6. Power on the rest of the nodes and observe them displayed in the WSNMonitor.
7. Select any router node and click on the bulb icon next to it, observe the device to blink its LEDs.

For demonstration of over-the-air upgrade functionality, see Section 7.8.

7.3 Network Startup

The coordinator organizes the wireless network automatically. Upon starting, every node informs the network of its role.

When the coordinator is powered on, it switches to an active state even though no child node is present. This is normal, and indicates that the coordinator is ready and the child nodes can join the network with the coordinator's extended PAN ID. By default, the coordinator uses extended PAN ID `0xAAAAAAAAAAAAAAAA`, which is recognized by all routers. A short PAN ID is chosen at random. The extended PAN ID can be modified by the user through the application's `configuration.h` file.

If the coordinator is absent or has not been turned on, the routers and end devices will remain in the network search mode. In this mode, routers scan the channels specified in the channel mask in search of a network with the specified extended PAN ID.

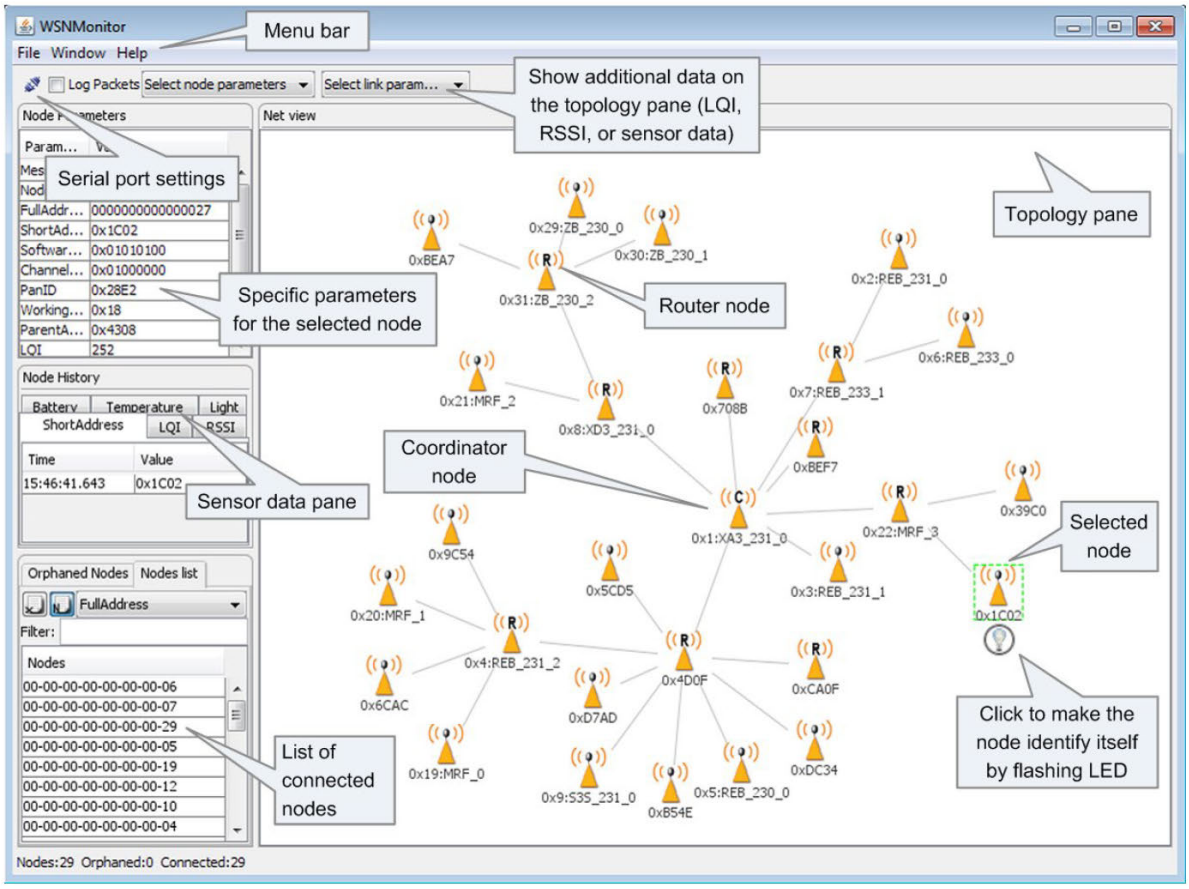
By default, the channel mask for all application images provided with the SDK contains a single channel. In rare cases, if the frequency corresponding to the radio channel is busy, the coordinator node may stay in the network search mode. If this happens, it may become necessary to change the application's channel mask to select another channel by changing the application's `configuration.h` file and recompiling the application.

Network health can be monitored through the WSNMonitor application described in the next section.

7.4 WSNMonitor

WSNMonitor is a PC counterpart to the WSNDemo embedded application, and can be used to display the ZigBee network topology and other information about a wireless sensor network. A typical WSNMonitor screen is shown in Figure 7-1. It contains topology, sensor data, node data panes, and application toolbars.

Figure 7-1. WSNMonitor GUI



The *topology pane* displays the network topology in real time, which helps the user monitor the formation of and dynamic changes in the network while nodes join, send data across, or leave the network. The network topology is constructed on the basis of next-hop information for each of the nodes, and each link is also tipped with RSSI and LQI values. Each of the nodes displayed is depicted by an icon, with the node's address or name below and sensor readings to the right of the icon, if required by settings.

The *sensor data pane* displays data coming from onboard sensors of the selected node (see Section 7.7). It is presented in graph and table form. Other parameters can be observed for each node in table form. The *node data pane* includes a *sensor selection combo-box*, which is used to switch between sensor types.

By default in the topology pane, nodes are labeled with their short addresses. However, another title can be assigned to any desired node by a double click. If "Cancel" is pressed in the opened window, the node's title is set back to the short address.

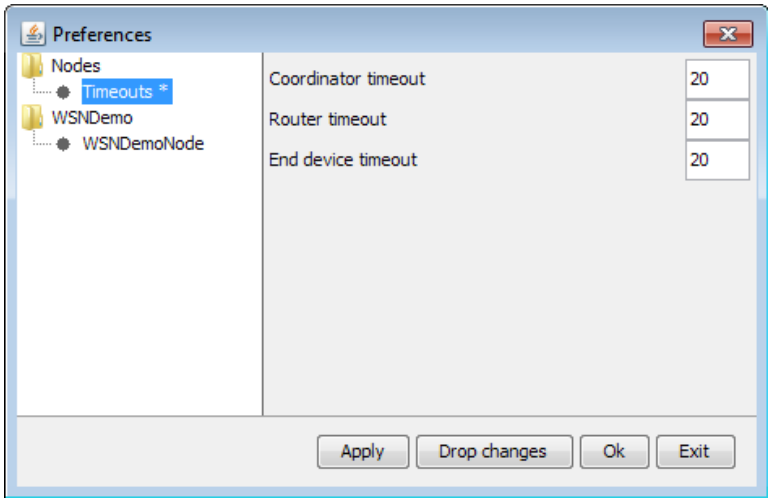
7.5 Identifying Nodes

When a user clicks a node in the topology pane a button that can be used to identify the node appears under the node's icon. When the user clicks this button WSNMonitor sends a command, which is delivered to the coordinator through the serial connection and wirelessly to the target node. The target node, receiving the command, blinks with its LED for several seconds.

7.6 Node Timeouts

The `Window/Preferences` menu of WSNMonitor contains a number of parameters used to control application behavior. Timeouts are used to tune visualization of coordinator, routers, and end devices as the nodes disappear from the network each time a connection is lost, power is down, or a reset has occurred. A node timeout corresponds to the time the WSNMonitor application waits for a packet from a particular node before assuming that the node is no longer part of the network. Note that this value does not correspond to the frequency with which data are transmitted by each type of device. To get smooth topology visualization, setting timeouts to 20 seconds is recommended for coordinator and router, and 30 seconds is recommended for an end device. Assuming a default application configuration, these timeouts cover three periods between sending a packet, and so at least three packets would need to be lost before a node is removed from the WSNMonitor topology pane.

Figure 7-2. WSNMonitor Preferences Menu



7.7 Sensor Data Visualization

Each board sends temperature/light/battery sensor readings (or emulated values) to the coordinator, which in turn sends it to the PC. WSNMonitor displays the readings from onboard sensors next to a node icon inside the topology pane. A corresponding option can be selected in the node/link parameters from the quick settings toolbar.

The user can select any node in the topology pane to monitor the node's activity and see the node data in one of three different forms:

- Text table
- Chart

The onboard sensor's data displayed next to each node in the topology pane. These values are also tipped with arrows indicating whether the value increased or decreased in relation to the previous sample.

A given node is selected when it is clicked on and a dashed frame is visible around it.

The same values are shown on the sensor data pane, enabling the user to observe how the values change over a period of time.

The sensor data pane includes a sensor selection combo-box. Use the button on the sensor control toolbar to display the desired types of sensor data.

7.8 Over-the-Air Upgrade

Over-the-Air demonstration requires one WSNDemo device (most commonly network coordinator) programmed with an application image supporting OTAU server functionality and at least one device programmed as an OTAU client.

Atmel devices serving as OTAU clients shall be connected with an external flash device as described in corresponding platform-specific Sections (see Section 3.3 for references).

The user should configure and install devices as follows:

1. Configure and compile the WSNDemo application with `APP_USE_OTAU` defined as `1`, and definition of `OTAU_CLIENT` uncommented, in the `configuration.h` file. Load this application image to the board with a an external flash device connected:
   a. Program the embedded bootloader image file from the Serial Bootloader package [12] that corresponds for the target platform.
   b. The application image should be converted to `*.srec` format and installed using the Bootloader PC tool from the Serial Bootloader package. The device is now able to perform as an OTA client, as defined in [24]. The above process should be repeated for every node that the user intends to upgrade over the air.
2. Configure and compile the WSNDemo application with `APP_USE_OTAU` defined as `1`, and definition of `OTAU_SERVER` uncommented, in the `configuration.h` file. Load this application image to the board selected to serve as an OTAU server, following steps 1.a and 1.b.
3. Once the images are programmed and WSNDemo devices are joined to the network, follow instructions given in [24] to update the firmware over the air.

Note: When coordinator acts as OTA the server network activity cannot be monitored on the WSNMonitor as the same serial interface will be used for communication with the OTA bootloader tool.

Appendix A. SAMR21 Specifics

BitCloud supports ATSAMR21-XPRO [34] development board with ATSAMR21G18A. BitCloud reference applications can be compiled for different boards' configurations by selecting corresponding value for `BSP_SUPPORT` parameter in application `configuration.h` file. Setting this parameter to `BOARD_FAKE` will disable all board-specific peripherals and can be used for custom boards.

A.1 Hardware Setup

A.1.1 Required Hardware

Make sure that all necessary hardware is available:

- two or more ATSAMR21-XPRO boards [33] each with an external 2.4GHz antenna and a Micro-USB cable
  or
- two or more custom boards with ATSAMR21G18A or ATSAMR21E18A devices and one Atmel JTAGICE3 [22].

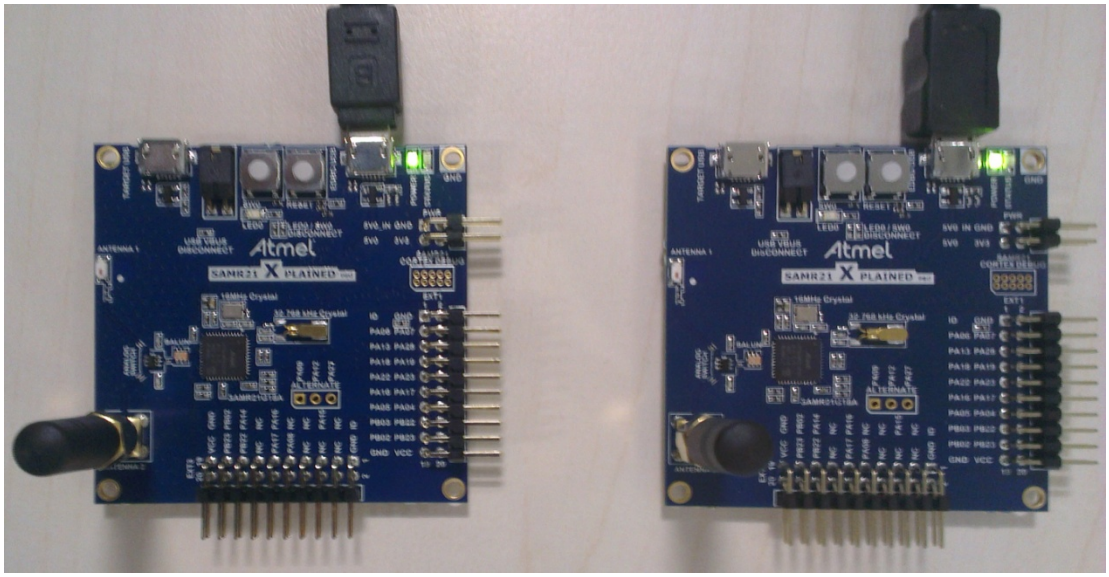
A.1.2 ATSAMR21 Xplained PRO Setup

The SAM R21 Xplained Pro contains the Atmel Embedded Debugger (EDBG) for on-board debugging. The EDBG is a composite USB device of three interfaces; a debugger, Virtual COM Port, and Data Gateway Interface (DGI). In conjunction with Atmel Studio, the EDBG debugger interface can program and debug the ATSAMR21G18A. On the SAMR21 Xplained PRO, the SWD interface is connected between the EDBG and the ATSAMR21G18A. The Virtual COM Port is connected to a UART port on ATSAMR21G18A and provides an easy way to communicate with the target application through simple terminal software. This subsection contains only brief instructions. Detailed hardware description for this kit is given in corresponding user guide [34].

To assemble a device:

1. Attach an antenna to the SMA connector on ATSAMR21-XPRO.
2. Connect the Micro-USB cable to the EDBG USB section on ATSAMR21-XPRO to power up (see Figure 7-3).

Figure 7-3. ATSAMR21 – Xplained PRO Setup



Note: To properly support on-board peripherals such as LEDs, buttons, and serial connection present on ATSAMR21-XPRO BitCloud applications shall be configured with the following defines selected in the application's `configuration.h` file:

```
#define BSP_SUPPORT BOARD_SAMR21_XPRO
#define APP_INTERFACE APP_INTERFACE_USART
#define APP_USART_CHANNEL USART_CHANNEL_0
```

A.1.3 OTAU Hardware Setup

To demonstrate OTA upgrade functionality, the pins of the external flash memory device must be connected to the ATSAMR21 pins as shown in Table A-1. More information on the OTA upgrade can be found in [24].The possible configurations of M25P40 and AT25DF041A Flash variants are mentioned in Table A-1.

Table A-1. External Flash and MCU Pin Assignment

External Flash Pins – AT25DF041A	External Flash Pins – M25P40	ATSAMR21 Pins
CS	S#	PA14 (Pin23)
SO	Q	PA15 (Pin24)
WP	W#	n/a
SI	D	PA08 (Pin15)
SCK	C	PA09 (Pin16)
HOLD	HOLD	n/a

A.2 Pre-built Firmware Images

BitCloud SDK for SAMR21 devices provides precompiled firmware images for supported Atmel development boards (see Section 1.3). The images are located in corresponding application folders present in `./Evaluation Tools/`directory. The user can load them onto the board as described in Section A.3.

A.3 Programming the Boards

Firmware images can be loaded to the boards by using the following methods: programming using JTAG/EDBG either in IAR Embedded Workbench (see Section A.3.2) or Atmel Studio (see Section A.3.3), and programming with Serial Bootloader (see Section A.3.4).

A.3.1 Extended (MAC) Address Assignment

For the proper communication ZigBee require unique 64-bit MAC address assigned for each device. A node is not able to join any ZigBee network unless its extended address is non-zero and smaller than `0xFFFFFFFFFFFFFFFA`.

In BitCloud the `CS_UID` parameter defines such address and by default a compile time is set to invalid value `0x0`. It is the responsibility of the application to obtain the correct value and assign it to the `CS_UID` parameter.

Reference applications provided by Atmel assign the MAC address as follows. If `CS_UID` is set to zero at compile time, the application attempts to load the MAC address from a dedicated board-specific source for the UID value using API of the BSP component. This can be external EEPROM, or specially programmed user page of the chip. If such address cannot be obtained, then MAC address is kept as `0`. Hence for custom boards applications shall be updated to assign valid MAC address to the node.

For SAMR21-XPRO the `CS_UID` should be set to a non-zero value from the application.

A.3.2 Programming with IAR Embedded Workbench

A.3.2.1 Precompiled Images

When using IAR Embedded Workbench to program precompiled images provided with the SDK, the user first needs to create a project containing the precompiled image.

1. Start IAR Embedded Workbench for ARM [19].
2. Select `File > New > Workspace`.
3. Select `Project > Create New Project…`
4. In the `Create New Project` dialog, select `Externally build executable` in `Project templates`:
5. Select a name for the project, and click `Save`.
6. Follow the instructions in `readme.txt`.
7. Once the project is set up, select `Project > Options`.
8. In the `General options` category, set `Target -> Device` as `Atmel ATSAMR21G18A` or `Atmel ATSAMR21E18A`, depending on the target device.
9. In `Debugger` category > `Setup` tab select in the `Driver` drop-down `CMSIS DAP` if EDBG is used or `JTAGICE3` when programming via JTAG.
10. Click `OK` button.
11. Select `Project > Download and debug`.
12. Once the debugging session has started, click `Stop debugging`.

The image is now installed on the device.

A.3.2.2 Application Workspace

1. Open target application project in the IAR Embedded Workbench for ARM [19].
   This can be done either by double-clicking on the workspace file (for example, `HADevice.eww`) or by opening such file directly in the IAR Workspace using `File > Open > Workspace.`
2. Select the desired application configuration (for example, `All_StdlinkSec_SamR21_Atsamr21b18a_Rf233_8Mhz`) from the drop-down box in the Workspace pane.
3. Select `Project > Options`. Then in `Debugger` category > `Setup` tab select in the `Driver` drop-down menu `CMSIS DAP` if EDBG is used or `JTAGICE3` when programming via JTAG. Click `OK` button.
4. Select `Project > Download and debug`.
5. Once the debugging session has started, click `Stop debugging`.

The image is now installed on the device.

Note: Using a JTAG/EDBG to program the microcontroller will erase the embedded bootloader, if present. As a result, loading of application images with Serial Bootloader will become inoperable until the embedded bootloader is loaded to the device again.

A.3.3 Programming with Atmel Studio

1. In Atmel Studio, open the `Tools > Device Programming…` dialog.
2. From the `Tool` drop-down menu select the programming tool (for example `JTAGICE 3/EDBGATMLXXXX`).
3. Select the right device (`ATSAMR21G18A`) in the `Device` drop-down menu and select the Interface as SWD.
4. Press `Read` button in the `Device Signature` field to verify that connection with the device is correct.
5. Click on the `Program` tab.
6. In the `Flash` section of the dialog, select the precompiled .hex file to be programmed.
7. Click `Program`.

The image is now installed on the device.

A.3.4 Programming with Serial Bootloader

Programming using Serial Bootloader requires that the embedded bootloader code is loaded to the device via JTAG/EDBG. Firmware images for the embedded bootloader as well as the Bootloader PC tool, which is needed to load the application image from a PC to the device, are included in the Atmel Serial Bootloader software package available for downloading from the Atmel website.

Images that shall be loaded to device via JTAG may be found under the `\Embedded_Bootloader_images\ATSAMR21` directory in the package:

1. Install and run the Bootloader PC tool from the command line or use the GUI. Specify the target image file in `.srec` format and the COM port, and launch the firmware upload (see [12]).
2. Perform a hardware reset on the board by using the reset button, if requested.
3. The Bootloader PC tool indicates the operation progress. Once the upload is successfully completed, the board will restart automatically. If an upload fails, the Bootloader PC tool will indicate the reason. In rare cases, the booting process can fail due to communication errors between the board and the PC. If this happens, attempt booting again. If booting fails, the program recently written to the board will be corrupted, but the board can be reprogrammed again as the embedded bootloader should remain intact.

Warning: Using JTAG/EDBG to program the microcontroller will erase the embedded bootloader, making the loading of application images with Serial Bootloader impossible until the embedded bootloader firmware is reprogrammed to the device.

A.4 Reserved Hardware Resources

Table A-2. Hardware Resources Reserved by the Stack on ATSAMR21

Resource	Description
Processor main clock	16/24/48 MHz from XOSC32K Oscillator
Processor main clock	4/8 MHz from internal RC Oscillator
PA08, PA09, PA14, PA15	External DataFlash, when OTAU functionality is used
Timer/counter 3	System Timer
Timer/counter 4	Mac Timer Implementation - Rtimer
Flash	Bootloader: 8KB. Required if OTAU is used. Can be disabled otherwise. NVM area: 16KB. Configurable via application linker scripts. FW image: Application dependent. EEPROM: Emulated in Flash. Part of it is used when OTAU is supported. More details see [24].

Appendix B. ATmegaRFR2 Specifics

BitCloud supports out of the box two different ATmegaRFR2 development boards with ATmega256RFR2 SoC: AT256RFR2-EK [15] and AT256RFR2-XPRO [17].The instructions below will highlight the differences between the two platform configurations, where present. It is also possible to compile applications for custom boards without relying on any board-specific peripherals.

BitCloud reference applications can be compiled for different boards' configurations by selecting corresponding value for `BSP_SUPPORT` parameter in application `configuration.h` file. Setting this parameter to `BOARD_FAKE` will disable all board-specific peripherals and can be used for custom boards.

B.1 Hardware Setup

B.1.1 Required Hardware

Make sure that all necessary hardware is available:

- one or more AT256RFR2-EK kits [15] and one Atmel JTAGICE3 [22]
  or
- two or more AT256RFR2-XPRO [17] boards, each with an external 2.4GHz antenna and a Micro-USB cable
  or
- two or more custom boards with ATmega256RFR2 or ATmega2564RFR2 devices and one Atmel JTAGICE3 [22].

B.1.2 AT256RFR2-EK Setup

This subsection contains only brief instructions. Detailed hardware description for this kit is given in corresponding user guide [16].

Radio Controller Board (RCB) with ATmega256RFR2 is used as a base board. Depending on the application RCB might be used as a standalone, mount to a Key Remote Control board or to a Sensor Terminal Board (STB).

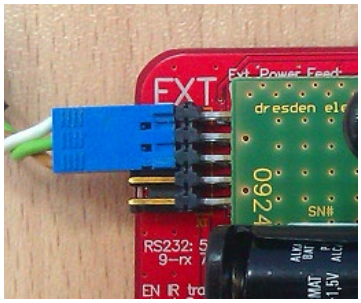
To assemble a device:

1. Attach an antenna to the SMA connector on RCB.
2. Insert two AAA batteries into the RCB if it will be used in standalone mode or on top of Key Remote Control board. Do not use batteries if the STB board is powered by USB.
3. Attach RCB on top of a Key Remote Control board or STB board.
4. Power on the device by shifting the red button at the right side of the RCB (see Figure 7-4).

Figure 7-4. Assembled RCB Hosted on a Key Remote Control Board

5. To use serial output, attach the serial cable, connected to a PC, to the port marked EXT on a Key Remote Control board as shown on Figure 7-5, or use the USB connection on the STB board.

Figure 7-5. Serial Cable attached to a Key Remote Control Board



Note: Different board setups available in 256RFR2-EK provide application with access to different types of peripherals such as LEDs, buttons, LCD and serial connection. BitCloud applications shall be configured differently for each board configuration tos properly support such peripherals. This is done via selection of the following defines in the application's `configuration.h` file:

- For an RCB mounted on a Key Remote Board:

```
#define BSP_SUPPORT BOARD_RCB_KEY_REMOTE
#define APP_INTERFACE APP_INTERFACE_USART
#define APP_USART_CHANNEL USART_CHANNEL_0
```

- For an RCB mounted on an STB:

```
#define BSP_SUPPORT BOARD_RCB
#define APP_INTERFACE APP_INTERFACE_USBFIFO
```

- For a standalone RCB:

```
#define BSP_SUPPORT BOARD_RCB
#define APP_INTERFACE APP_INTERFACE_USART
#define APP_USART_CHANNEL USART_CHANNEL_1
```

B.1.3 AT256RFR2-XPRO Setup

Detailed hardware description for the 256RFR2 Xplained Pro board is given in corresponding user guide [18].

Note: Due to board-specific control, `Flow control` of the COM port setting on the PC shall be set to `Hardware` to enable serial communication between the Xplained Pro board with BitCloud application and a PC (for serial terminal and PC GUI applications such as WSNMonitor or SLRemote). Even though configuration in the embedded application firmware is not using HW flow control for serial communication.

Note: To properly support on-board peripherals such as LEDs, buttons and serial connection present on AT256RFR2-XPRO BitCloud applications shall be configured with following defines selected in application's `configuration.h` file:

```
#define BSP_SUPPORT BOARD_ATMEGA256RFR2_XPRO
#define APP_INTERFACE APP_INTERFACE_USART
#define APP_USART_CHANNEL USART_CHANNEL_0
```

B.1.4 OTAU Hardware Setup

To demonstrate OTA upgrade functionality, the pins of the external flash memory device (by default AT25DF041A is supported) must be connected to the ATmega256RFR2 pins, as shown in Table B-1. More information on the OTA upgrade can be found in [24].

Table B-1. External Flash and MCU Pin Assignment

External Flash Pins	ATmega256RFR2 Pins
CS	PE3 (pin49)
SO	PE0 (pin46)
WP	n/a
SI	PE1 (pin47)
SCK	PE2 (pin48)
HOLD	n/a

B.2 Pre-built Firmware Images

BitCloud SDK for megaRF devices provides precompiled firmware images for supported Atmel development boards (see Section 1.3). The images are located in corresponding application folders present in `./Evaluation Tools/`directory. The user can load them onto the board as described in Section B.3.

B.3 Programming the Boards

Firmware images can be loaded to the boards using the following methods: programming using JTAG/EDBG either in IAR Embedded Workbench (see Section B.3.3) or Atmel Studio (see Section B.3.4), and programming with Serial Bootloader (see Section B.3.5). Before programming; make sure that the fuse bits are configured correctly, as described in Section B.3.1.

B.3.1 Setting Fuse Bits

Table B-2 presents the fuse bit configuration for ATmega256(4)RFR2 devices running BitCloud applications. It also describes some use cases when certain fuse bits require values different from the default ones. Based on own application-specific requirements, the fuse bits can be changed as well. See device datasheet [9] for detailed fuse bits description.

Note: Modifying fuse bit settings cannot be done with the Serial Bootloader tool.

Table B-2. Fuse Bit Settings for the ATmega256(4)RFR2 Device

Option	Value for 8MHz	Value for 16MHz	Comments
BODLEVEL	1V8	1V8	Can be changed according to application-specific requirements.
OCDEN	Disabled	Disabled	Can be changed during application development. Must be disabled for final products.
JTAGEN	Enabled	Enabled	Must be enabled for JTAG access.
SPIEN	Enabled	Enabled	Must be enabled SPI programming.
WDTON	Disabled	Disabled	Can be changed according to application-specific requirements.
EESAVE	Disabled	Disabled	Can be changed according to application-specific requirements.
BOOTSZ	Boot Flash size=4096 words start address=$1F000	Boot Flash size=4096 words start address=$1F000	Specifies section size in words (two bytes) reserved in flash memory for the embedded bootloader. Applied only if BOOTRST fuse is enabled. Must be set to `Boot Flash size=4096 words start address=$1F000`, if Atmel bootloader or OTA is used on the device. For more information see [12].
BOOTRST	Disabled	Disabled	Shall be enabled if device needs to be programmed with Serial Bootloader or if OTAU support is required. Can be disabled in other cases. For more information see [12].
CKDIV8	Enabled	Disabled	
CKOUT	Disabled	Disabled	
SUT_CKSEL	Int. RC Osc.; Start-up time: 6 CK + 65ms	Transceiver Oscillator; Startup time: 16K CK + 65ms	Can be changed according to application-specific requirements, but external clock source shall not be used by sleeping devices.
Resulting bytes:			
Ext	0xFE	0xFE	
High	0x19	0x19	
Low	0x62	0xF7	

B.3.2 Extended (MAC) Address Assignment

For the proper communication ZigBee require unique 64-bit MAC address assigned for each device. A node is not able to join any ZigBee network unless its extended address is non-zero and smaller than `0xFFFFFFFFFFFFFFFA`.

In BitCloud the `CS_UID` parameter defines such address and by default at compile time is set to invalid value `0x0`. It is responsibility of the application to obtain the correct value and assign it to the `CS_UID` parameter.

Reference applications provided by Atmel assign the MAC address as follows. If `CS_UID` is set to zero at compile time, then the application attempts to load the MAC address from a dedicated board-specific source for the UID value using API of the BSP component. This can be external EEPROM, or specially programmed user page of the chip. If such address cannot be obtained, then MAC address is kept as `0`. Hence for custom boards applications shall be updated to assign valid MAC address to the node.

B.3.3 Programming with IAR Embedded Workbench

B.3.3.1 Loading Precompiled Images

When using IAR Embedded Workbench to program precompiled images provided with the SDK, the user first needs to create a project containing the precompiled image.

1. Start IAR Embedded Workbench for AVR [20].
2. Select `File > New > Workspace`.
3. Select `Project > Create New Project…`
4. In the `Create New Project` dialog, select `Externally build executable` in `Project templates`:
5. Select a name for the project, and click `Save`.
6. Follow the instructions in `readme.txt`.
7. Once the project is set up, select `Project > Options`.
8. In the `General options` category, set `Processor Configuration` to `ATmega256RFR2` or `ATmega2564RFR2`.
9. In `Debugger` category > `Setup` tab select `JTAGICE3` in the `Driver` drop-down menu.
10. Click `OK` button.
11. From the menu select `JTAGICE 3 > Fuse Handler`.
12. Click `Read Fuses`, and make sure that the device fuses are set as specified in Section B.3.1.
13. If fuses are set incorrectly, select the correct fuse settings, and click `Program fuses`.
14. Select `Project > Download and debug`.
15. Once the debugging session has started, click `Stop debugging`.

The image is now installed on the board.

B.3.3.2 Programming from Application Workspace

1. Open target application project in the IAR Embedded Workbench for AVR [20].
   This can be done either by double-clicking on the workspace file (for example, `HADevice.eww`) or by opening such file directly in the IAR Workspace using `File > Open > Workspace`.
2. Select the desired application configuration (for example, `All_StdlinkSec_MegaRf_ ATmega256RF2_ _8Mhz_Iar`) from the drop-down box in the Workspace pane.
3. Select `Project > Options`. Then in `Debugger` category > `Setup` tab select `JTAGICE3` in the `Driver` drop-down menu. Click `OK` button.
4. From the menu select `JTAGICE 3 > Fuse Handler`.
5. Click `Read Fuses`, and make sure that the device fuses are set as specified in Section B.3.1.
6. If fuses are set incorrectly, select the correct fuse settings, and click `Program fuses`. Then press `Close` button.
7. Select `Project > Download and debug`.
8. Once the debugging session has started, click `Stop debugging`.

The image is now installed on the device.

Note: Using a JTAG to program the microcontroller will erase the embedded bootloader, if present. As a result, loading of application images with Serial Bootloader will become inoperable until the embedded bootloader is loaded to device again.

B.3.4 Programming with Atmel Studio

1. In Atmel Studio, open the `Tools > Device Programming…` dialog.
2. From the `Tool` drop-down menu select the programming tool (for example JTAGICE 3).
3. Select the right device (`ATmega256RFR2/ATmega2564RFR2`) in the `Device` drop-down menu.
4. Press `Read` button in the `Device Signature` field to verify that connection with the device is correct.

5. Click on the Fuses tab and make sure that the device fuses are set as specified in Section B.3.1.
6. If fuses are set incorrectly, select the correct fuse settings, and click `Program`.
7. Click on the `Program` tab.
8. In the `Flash` section of the dialog, select the precompiled .hex file to be programmed.
9. Click `Program`.

The image is now installed on the device.

B.3.5 Programming with Serial Bootloader

Programming using Serial Bootloader requires that the embedded bootloader code is loaded to the device via JTAG. Firmware images for the embedded bootloader as well as the Bootloader PC tool, which is needed to load the application image from a PC to the device, are included in the Atmel Serial Bootloader software package available for downloading from the Atmel website.

Images that shall be loaded to device via JTAG may be found under the `\Embedded_Bootloader_images\Atmega256rfr2` directory in the package.

The fuse bits should be configured properly; namely, the `BOOTRST` fuse should be enabled as described in Section B.3.1.

If the embedded bootloader is loaded connect with a serial interface to a PC ensuring pin connections as shown in Table B-3. If bootloader image corresponding to the supported Atmel development board (Table 1-1) is used then such mapping is guaranteed already.

Table B-3. Host UART and MCU Pin Connections

UART Pin on Host Device	ATmega256RFR2 MCU Pin
RXD	PD2
TXD	PD3
GND	D_GND

1. Install and run the Bootloader PC tool from the command line or use the GUI. Specify the target image file in `.srec` format and the COM port, and launch the firmware upload (see [12]).
2. Perform a hardware reset on the board by using the reset button, if requested.
3. The Bootloader PC tool indicates the operation progress. Once the upload is successfully completed, the board will restart automatically. If an upload fails, the Bootloader PC tool will indicate the reason. In rare cases, the booting process can fail due to communication errors between the board and the PC. If this happens, attempt booting again. If booting fails, the program recently written to the board will be corrupted, but the board can be reprogrammed again as the embedded bootloader should remain intact.

Warning: Using JTAG to program the microcontroller will erase the embedded bootloader, making the loading of application images with Serial Bootloader impossible until the embedded bootloader firmware is reprogrammed to the device.

B.4 Reserved Hardware Resources

Table B-4. Hardware Resources Reserved by the Stack on ATmega256RFR2

Resource	Description
Processor main clock	8/16MHz from internal RC oscillator
TRX24	Radio
ATmega ports PG3, PG4	Asynchronous timer interface (optional – can be disabled)

Resource	Description
Timer/counter 2	Asynchronous timer (optional – can be disabled via HAL)
Timer/counter 4	System timer
External IRQ4	Wake-up on DTR (optional – can be enabled)
PE0..PE2, PG5	External DataFlash, when OTAU functionality is used
EEPROM	Part of EEPROM is used when OTAU is supported. More details see in [24]
Flash	Bootloader: 8KB. Required if OTAU is used. Can be disabled otherwise. Configurable via fuse bits (see Section). NVM area: 16KB. Configurable via application linker scripts. FW image: Application dependent.

8. References

[1] BitCloud Software Development Kit
[2] AVR2052: BitCloud Quick Start Guide (this document)
[3] BitCloud API Reference (available in BitCloud SDK)
[4] AVR2050: BitCloud Developer's Guide
[5] IEEE Std 802.15.4™-2006 Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)
[6] ZigBee PRO specification (05-3474r20)
[7] ZigBee Cluster Library specification (07-5123r04)
[8] ZigBee Home Automation Profile Specification (05-3520-29)
[9] Atmega256RFR2 device
[10] Atmega2564RFR2 device
[11] MinGW C/C++ Compiler
[12] AVR2054: Serial Bootloader User Guide
[13] Java Runtime Environment
[14] Jython
[15] 256RFR2-EK kit
[16] AVR10002: ATmega256RFR2 Evaluation Kit – User Guide
[17] 256RFR2-XPRO: ATmega256RFR2 Xplained Pro Evaluation Kit
[18] ATmega256RFR2 Xplained Pro User Guide
[19] IAR Embedded Workbench for Atmel ARM
[20] IAR Embedded Workbench for Atmel AVR
[21] IAR Embedded Workbench IDE User Guide
[22] JTAGICE3
[23] AT02597: ZigBee PRO Packet Analysis with Sniffer
[24] AVR2058: BitCloud OTAU User Guide
[25] Atmel Studio download
[26] Atmel Studio archive
[27] Atmel Studio online help
[28] AT02698: ZAppSI User Guide
[29] ZigBee Light Link Profile specification (11-0037-10)
[30] AT03663: Power Consumption of ZigBee End Devices
[31] ATSAMR21G18A device
[32] ATSAR21E18A device
[33] SAMR21-XPRO: SAMR21 Xplained Pro Evaluation Kit
[34] ATSAMR21 Xplained PRO User Guide
[35] MSYS Extension for Atmel Studio
[36] AT08550: ZigBee Attribute Reporting

9. Revision History

Doc. Rev.	Date	Comments
Q	02/2015	Updated for BitCloud SDK 3.2.0. Description of new device types, existing device types as part of ZHA reference application along with minor improvements throughout document.
P	08/2014	Updated for BitCloud SDK 3.1.0. Description of SAMR21-specifics is added. Minor improvements through all the document.
O	03/2014	Updated for BitCloud SDK 3.0.0. Merged with AVR2055 BitCloud Profile Suite Quick Start Guide.
N	05/2012	Updated for BitCloud SDK 1.14.0.

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

© 2015 Atmel Corporation. All rights reserved. / Rev.: 8200Q–MCU–02/2015

Atmel®, Atmel logo and combinations thereof, AVR®, BitCloud®, Enabling Unlimited Possibilities®, and others are registered trademarks or trademarks of Atmel Corporation in U.S. and other countries. Windows® is a registered trademark of Microsoft Corporation in U.S. and or other countries. ARM® and Cortex® are registered trademarks of ARM Ltd. Other terms and product names may be trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

X-ON Electronics

Largest Supplier of Electrical and Electronic Components

Click to view similar products for Zigbee Development Tools (802.15.4) ***category:***

Click to view products by Atmel ***manufacturer:***

Other Similar products are found below :

76000956 ATZB-X-212B-XPRO SKY66114-11-EK1 SKY66403-11EK1 WRL-14549 ATREB212BSMA-EK ATZB-X-212B-USB XBIB-CU-TH XK3-Z8S-WZM XKA2C-Z7T-U IS.OMB-001 MIKROE-4277 ENWC9B01AQEF RBK-ZW500-E2 RBK-ZW500-H2 RBK-ZW500-U2 KIT-15936 CC2538DK CC2538EMK