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.

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

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

Chapter 1 provides an overview of the SDK, required development tools as well as 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):
- 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 [4] 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 BitCloud software development kit (SDK) 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 [26] 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 5.10)
- 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 [23] or IAR Embedded Workbench® [17], [18]), 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 necessary tools to compile application source code into binary images
- A programming device (for example, JTAGICE3 [20]), which may be used to program and debug the application on a target platform
- Optional: Atmel Serial/OTA Bootloader [10] if firmware programming over the serial interface or over-the-air is required
- Optional: ZigBee packet sniffer tool [21] for capturing over-the-air traffic

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

Note: Some specific BitCloud distributives or applications may support fewer development tool chains. For example, BitCloud ZLLDemo supports only IAR tool chain (project files and makefiles for Atmel Studio are not included in the package).

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	RF Transceiver	Supported Evaluation Kit	Supported IDEs
megaRFR2	ATmega256RFR2 [8] ATmega2564RFR2 [8]	Built-in	AT256RFR2-EK [13], ATMEGA256RFR2-XPRO [15]	IAR Embedded Workbench for AVR® 6.21.2 (with C/C++ compiler 6.21.50603) [17] Atmel Studio v6.1.2730 (with native GCC compiler) [23]

2. BitCloud Documentation

This Chapter describes the documentation set available for BitCloud SDK. It is intended to help user 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 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 [3]	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 [2]	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 [22]	Describes how to use Over-the-Air upgrade feature in BitCloud applications
AVR2057: ZAppSI User Guide [26]	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 [21]	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 [10]	Describes the standalone Serial Bootloader package, which is used to load firmware images to devices via serial connection. Not included into BitCloud SDK.

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) 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 developer guide: *AVR2050: BitCloud Developer Guide*. 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 documents in inevitable for the application development on top of BitCloud stack.

The user may find 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 API References.

Application notes focus on specific important topics. These kinds 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 setup 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 [26]
`./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 [3]
`./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

Directory/File	Description	Notes
./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)	
./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	
./Evaluation Tools/ZAppSINP/	ZigBee PRO network processor application.	See [26] for details
./Evaluation Tools/Runner/	Installation file for the WSNRunner application, which is used to run the ZLL test scripts	Described in Section 5.10
./Evaluation Tools/SLRemote/	ZLL Bridge GUI application's installation files.	See Section 5.6.1
./Evaluation Tools/Scripts/	ZLL certification test scripts	See Section 5.10
./Documentation/	BitCloud documentation, including this application note	See Chapter 2
./ThirdPartySoftware/	Third party software.	

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* [10]. OTAU use is fully described in *AVR2058: OTAU User Guide* [22].

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.

Installation procedure:

a. Download and install Atmel Studio [23] 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 [18] can be used to develop and debug applications for AVR-based platforms. IAR Embedded Workbench for ARM [17] 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 [18], 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 6.20`). 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.
- IAR Embedded Workbench for ARM:
  a. Download and install IAR Embedded Workbench for ARM [17], 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 6.50`). To do this, go to `Control Panel > System > Advanced > Environment Variables`, click `New` below `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

Platform	Refer to
megaRFR2	Appendix A

4. Building BitCloud Applications

This chapter provides 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 stack components and hardware drivers are provided in source code 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

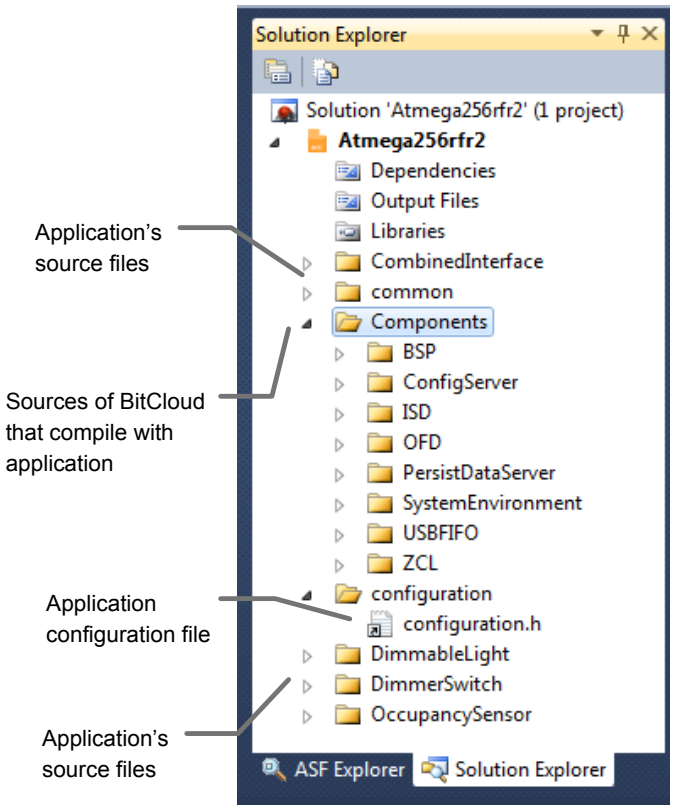
Atmel Studio can be used to develop and build Atmel BitCloud applications. Reference applications include Atmel Studio project files located in the `\atmelStudio_projects` subdirectory of the application root directory. These projects rely on the configurations given by external low-level makefiles (see 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 [25].

- **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 application.

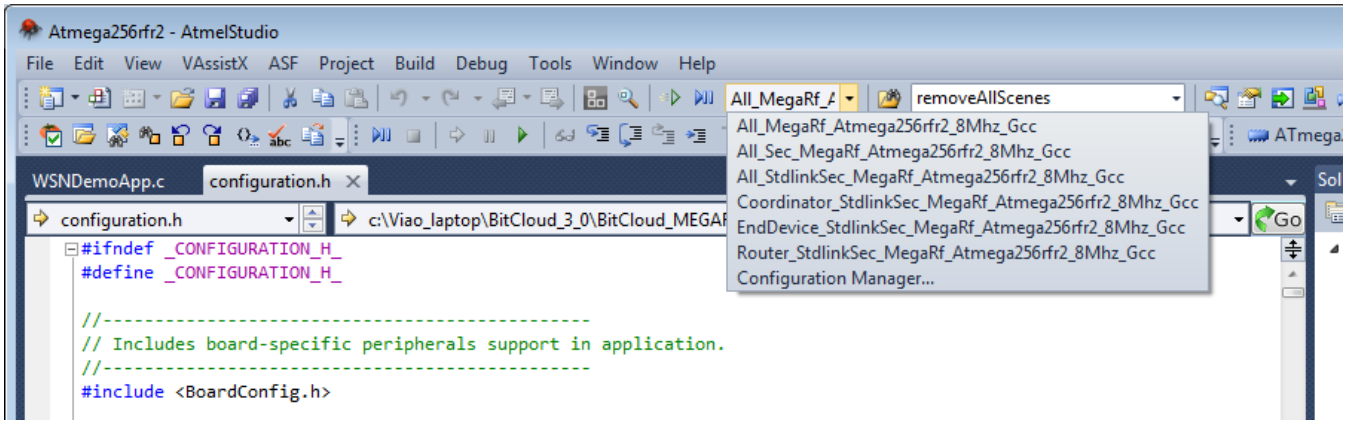
Note: Source files are included virtually to provide access to them. The set of files to be actually compiled is defined in corresponding external Makefile as described in Section 4.3.

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 IAR Embedded Workbench documentation [19].

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.

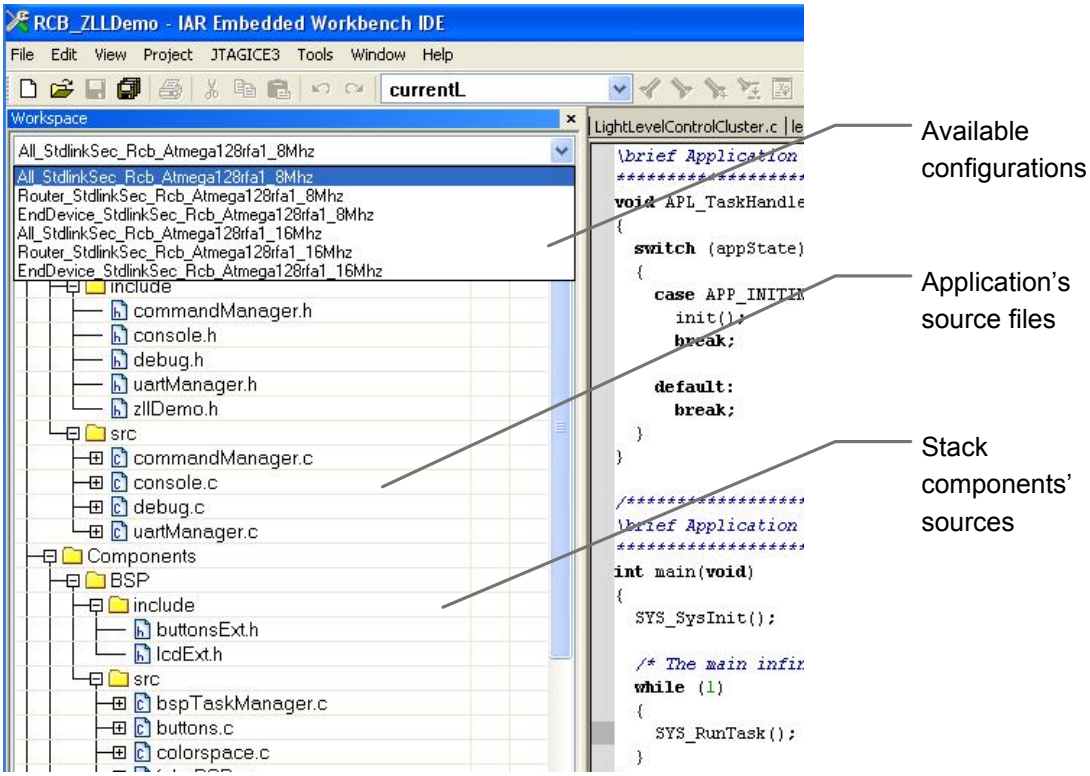
By default, the `.a90` file (for WSNDemo, `WSNDemo.a90`) will be generated in the `\iar_projects\Debug\exe` subdirectory (for WSNDemo, in the `<SDK-Root>\Applications\WSNDemo\iar_projects\Debug\exe` directory) with format as specified in Linker Output Options of the IAR project.

- ***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:

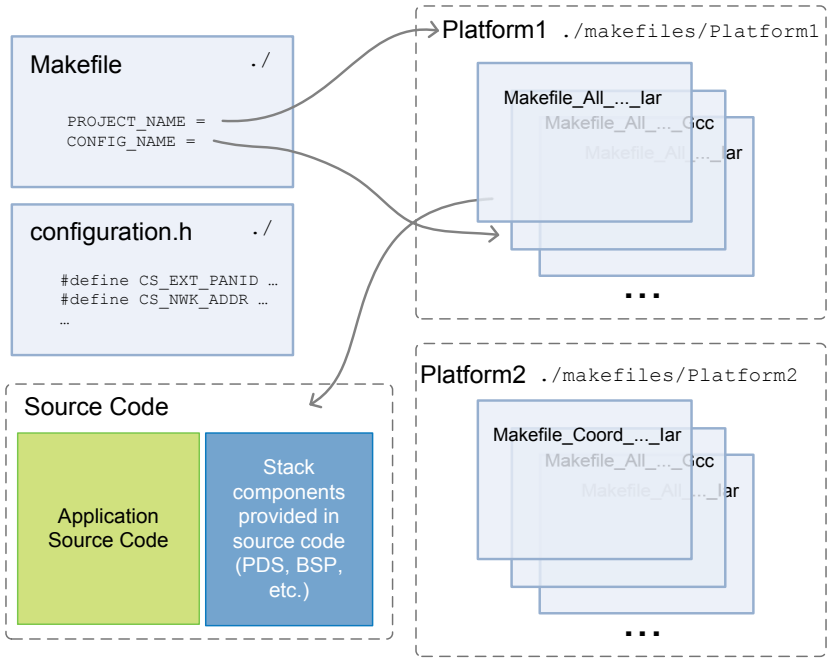
```
PROJECT_NAME = Atmega256rfr2
CONFIG_NAME = All_StdlinkSec_MegaRf_Atmega256fr2_8Mhz_Gcc
```

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 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:

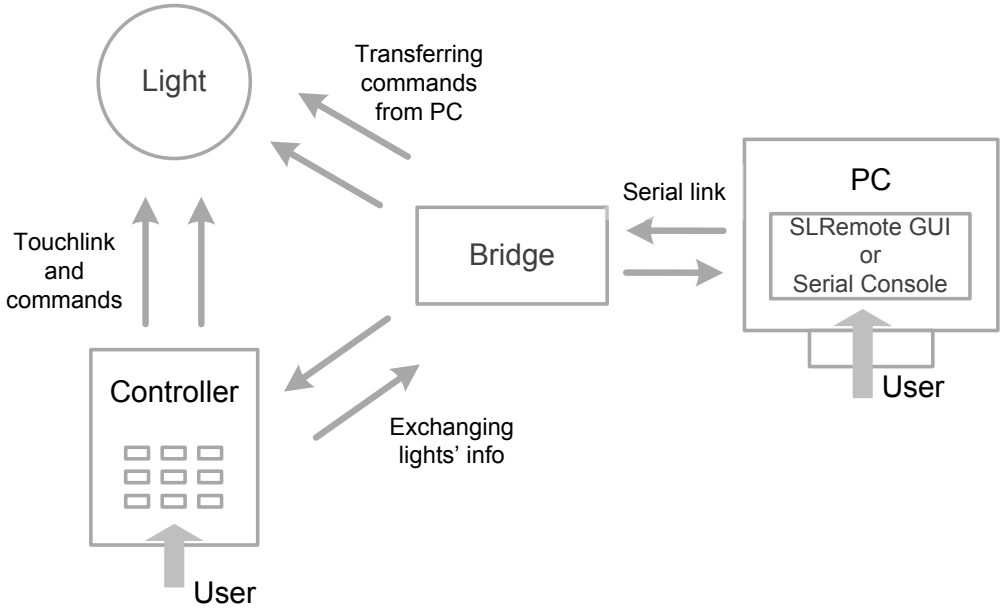
- ZigBee device type (`All`, `Coordinator`, `Router`, or `EndDevice`); All means that this configuration can be used for any device type
- Security supported (standard with link keys (`StdlinkSec`), standard `(Sec)` or none (empty))
- Platform (board or SoC family)
- MCU
- Radio chip, if applicable
- 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 [27] 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 1 light is needed and one color scene controller or a bridge. More devices (lights, controllers or bridges) may be added as required. Follow the instruction below to launch a demo:

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\ZLLDemo` directory. Precompiled light's firmware is for an extended color light device. To try other types of the light device, the application must be recompiled with changes made to the configuration.h file (see Section 5.4).

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

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

3. Power on the devices.
4. If color scene controller device is used:
   a. Perform a touchlink procedure between the light and the color scene controller by holding the PWR button on the controller's Key Remote, to form a network.
   b. Use color scene controller's buttons to control the light (see Section 5.7).
5. If bridge device is used:
   a. Create a network using a bridge device; open it for joining using SLRemote PC application or console commands as described in Section 5.6.
   b. Reset light device, on power up it will automatically join the network.
   c. Perform light discovery from the bridge using SLRemote or console commands.
   d. Control found lights using SLRemote or console commands.
6. For details in executing OTA see Section 5.8.
7. For details on interoperability with Home Automation networks see Section 5.9.
8. *Optionally:*
   - To send commands to a device and observe 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 (On for the XPRO board)
     ```

     The console commands can be sent following their syntax as described in Section 5.6.2.
   - More light devices and color scene controller devices may be added through touchlink with the controller device that is already in the network

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. Clusters Supported by the Demo Application (s – server, c – client)

Light	Color scene controller	Bridge
Basic (s) Commissioning (s) OnOff (s) Groups (s) Identify (s) Scenes (s) Level Control (s) (for dimmable light) Color Control (s) (for color light)	Basic (s) Commissioning (s) Basic (c) Commissioning (c) OnOff (c) Level control (c) Groups (c) Identify (c) Scenes (c) Color Control (c) Link Info (c) (manufacture-specific)	Basic (s) Commissioning (s) Link Info (s) (manufacture-specific) Commissioning (c) Identify (c) OnOff (c) Level Control (c) Groups (c) Scenes (c) Color Control (c)

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.

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 Parameters and their Meanings

Parameter	Description
APP_ZLL_DEVICE_TYPE	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
APP_ENABLE_CONSOLE	Set to 1 to enable commands sending through UART. If set to 1, it automatically disables sleep on remote controller and switches to emulated sleep.
APP_DEVICE_EVENTS_LOGGING	Set to 1 to enable output of information on application's events to console
BSP_SUPPORT	Set to: BOARD_RCB for compilation of firmware for a standalone RCB; BOARD_RCB_KEY_REMOTE for RCB mounted on a Key Remote Control board with output of device's information to the LCD screen and buttons support of a Key Remote Control board; BOARD_ATMEGA256RFR2_XPRO for Xplained Pro board BOARD_FAKE for custom boards.
APP_INTERFACE	Specifies serial interface used for connection with the PC. For USB connection on STB board shall be set to APP_INTERFACE_USBFIFO
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 and to 0 to disable it
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	Use fake implementation of the OFD driver. This may be useful for testing OTAU on boards without external Flash.
APP_SUPPORT_OTAU_PAGE_REQUEST	Enable or disable OTAU image page request feature (refer to [22] for details)
EXTERNAL_MEMORY	Specify 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 5.10) and to 0 otherwise

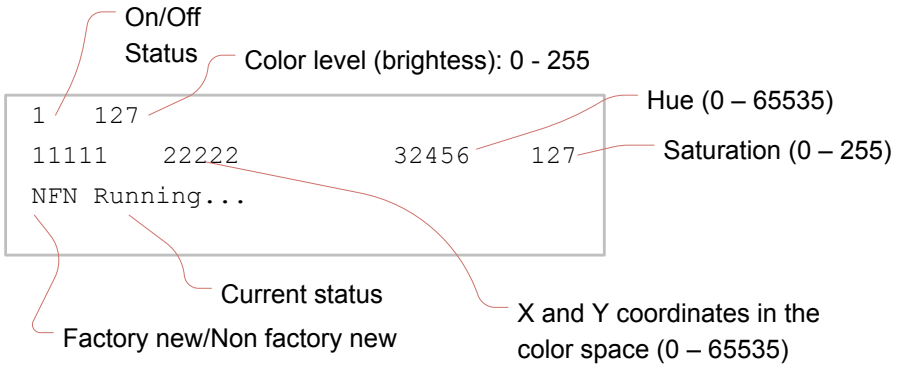
5.5 Light's Functions

A light device indicates its light status via LEDs on the RCB board, if Key Remote Control board is not used or via the LCD screen on the Key Remote Control board if the RCB is attached to it (see configuration options in Section 5.4.1).

RCB board's LEDs are turned on when light is on and turned off when light is off. When the light's color level changes LEDs brightness also changes. When a light receives an identify command from a controller LEDs start blinking.

Figure 5-2 below describes functions of a light's LCD screen. For the on-off light only On/Off status is shown. A dimmable light also shows 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 – temperature value is shown instead of X-Y color coordinates. Extended color light's screen shows all these values.

Figure 5-2. Light's LCD Screen on a Key Remote Control board. Which Parameters are shown on the Screen depends on the type of the Light Device



Once a factory new light is powered on, it starts standard ZigBee scanning and displays `Scanning 1st...` (indicating that the device is doing ZigBee scan over primary ZLL channels), then `Scanning 2nd...`(indicating ZigBee scan over secondary ZLL channels) as the current status. When `Listening...` is output to the screen the light is ready for touch link.

5.5.2 Touch Link with a Controller Device

For touch link a color scene controller is brought closely (10-20cm range) to a light and the PWR button is pressed on the color scene controller for more than 3 seconds. The color scene remote sends an identify command to the light, which blinks several times with its LEDs or the LCD screen. The touch link procedure may be aborted at this moment, if the PWR button is released. To complete the procedure the user should wait for 3 seconds.

If touch link is a success, the light's screen shows `NFN Running...` status (where NFN stands for Non Factory New).

After that, the light can be controlled by commands sent from the color scene controller.

5.5.3 Reset to the Factory New State

There are two ways to reset a light device to the factory new state: sending a command from a color scene controller and using the PWR button if the light is assembled with a Key Remote Control board.

To trigger reset of a light to factory new:

- Using the color scene controller:
  a. Select the target light using the SEL button on the color scene controller.
  b. Hold R1, R2, and PWR buttons, altogether, on the color scene controller for 3 seconds.
- For a light assembled with a Key Remote Control board:
  a. Switch off the device by shifting back the red button at the right side of the RCB board.
  b. While holding the PWR button on the light device, switch it on by shifting the RCB's red button. Wait for three seconds and then release the PWR button.
- For a standalone RCB light:
  a. Press and the hold the black button on the light's RCB.
  b. While holding the button, switch power on and off by shifting the RCB's button back and forth.

5.6 Bridge's Functions

The bridge device is meant to pass commands from a PC or tablet application to a ZLL network, allowing usage of that PC or tablet as the color scene controller. In this Atmel's implementation the bridge is an RCB connected to a PC via UART. On the PC, the SLRemote desktop application should be installed, and configured to use the COM port assigned to the RCB (details is Section 5.6.1). Another option is to send commands to the bridge device connected to a PC manually, through a terminal program (see Section 5.6.2). The bridge device also serves as the OTAU server.

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

- *Automatically.* On startup, the bridge automatically starts searching for open ZLL networks and tries to join them. Once it succeeds in it, the bridge sends a Match Descriptor ZDP request on the Commissioning cluster to discover lights.
- *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 5.6.2) or use selecting *Create network* from the SLRemote GUI as described in Section 5.6.1.
- *Through touchlink with a color scene controller.* In this case, bring the color scene controller close to the bridge, hold the PWR button on the color scene controller for more than 3 seconds to initiate touch link and do not release the button until the procedure is finished. 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 send commands to the lights.

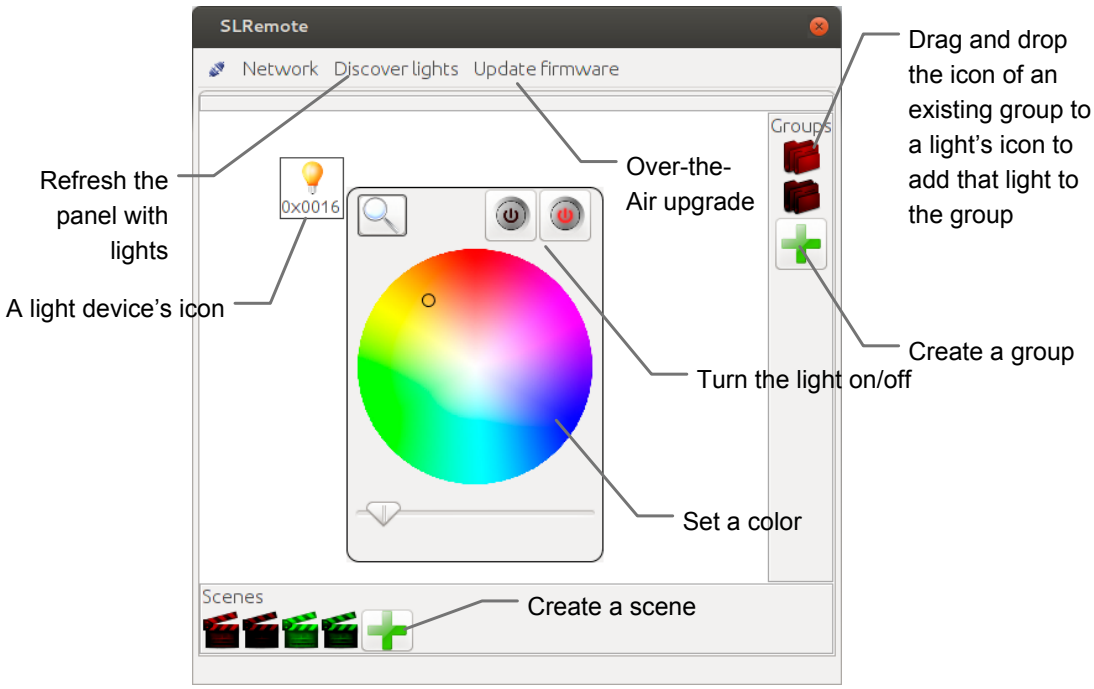
5.6.1 SLRemote GUI

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 5-3):

1. Download and install Java® Runtime Environment [11], 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, specifying the port corresponding to it.
6. Associate the bridge device with a network by clicking *Network* and selecting either:
   a. *Create network* to make the bridge create a new network and allow light devices to join it.
   b. *Discover network* to make the bridge device join an existing ZLL network and discover light devices in it.
7. 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.

Figure 5-3. SLRemote Application's Main Window and the Context Menu of a Light Device with Controls allowing sending Commands to that Light Device



5.6.2 Bridge's Console Commands

A bridge device can be controlled through a terminal program (HyperTerminal, RealTerm, etc.) on a PC. Set connection settings as follows:

```
BAUD RATE: 38400
PARITY: None
DATA BITS: 8
STOP BITS: 1
```

FLOW CONTROL: None (On for the XPRO board)

Once connection settings and the COM port assigned to the bridge device are specified, type commands in the terminal window. All supported commands are listed in Table 5-3.

Table 5-3. Console Commands Supported by the Bridge Device

Command syntax	Description
help	Show help contents
reset	Reset the bridge device
startNetwork	Scan for networks and try join one
startDiscovery	Start discovery of the light devices
setBrightness *<addrMode> <addr> <endpoint> <level> <time>*	Set brightness level on one or several light devices, depending on the address mode used
onOff *<addrMode> <addr> <endpoint> <"-on" – turn on; "-off" – turn off>*	Turn the specified light device(s) on or off
addGroup *<addrMode> <addr> <endpoint> <group>*	Send the Add Group command to the specified light device(s)
removeGroup *<addrMode> <addr> <endpoint> <group>*	Send the Remove Group command to the specified light device(s)
setColor *<addrMode> <addr> <endpoint> <hue> <sat> <time>*	Set color via hue and saturation on the specified light device(s)
identify *<addrMode> <addr> <endpoint>*	Send the Identify command to the specified light device(s)
scene *<addrMode> <addr> <endpoint> <cmd: "-store" or "-remove"> <group> <scene>*	Send Store Scene or Remove Scene command to the specified light device(s)
stepSaturation *<addrMode> <addr> <endpoint> <mode> <size> <time>*	Send the Step Saturation command to the specified light device(s)
stepBrightness *<addrMode> <addr> <endpoint> <mode> <size> <time>*	Send the Step Brightness command to the specified light device(s)
createNetwork *<ePanIdHigh> <ePanIdLow> <ch> <panId> <addr> <updateId>*	Create a network with the provided parameters on the bridge device. Lights searching for networks may join it.
sendPermitJoin *<permit>*	Send 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)

5.7 Color Scene Controller's Functions

5.7.1 Main Commands

Table 5-4 lists buttons for the main color scene controller's commands. See the scheme of the Key Remote board with all available commands on Figure 5-5.

Table 5-4. Buttons for Executing most Frequent Commands

Button(s)	Effect
PWR	Press and hold for 3 seconds in close proximity to the target device to perform touch link
PWR & R+	Press and hold for 3 seconds to perform classical ZigBee scanning
PWR & R-	Press and hold for 3 seconds to reset device to factory new state

Button(s)	Effect
PWR & R+ & R-	Press and hold for 3 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 5 seconds of inactivity.
1/2/3	Store Scene if pressed for more than 3 seconds and Recall scene if pressed for less that 3 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 active, which means that all commands initiated by the user are sent to this light in a unicast manner, until not commands are sent during 5 seconds.

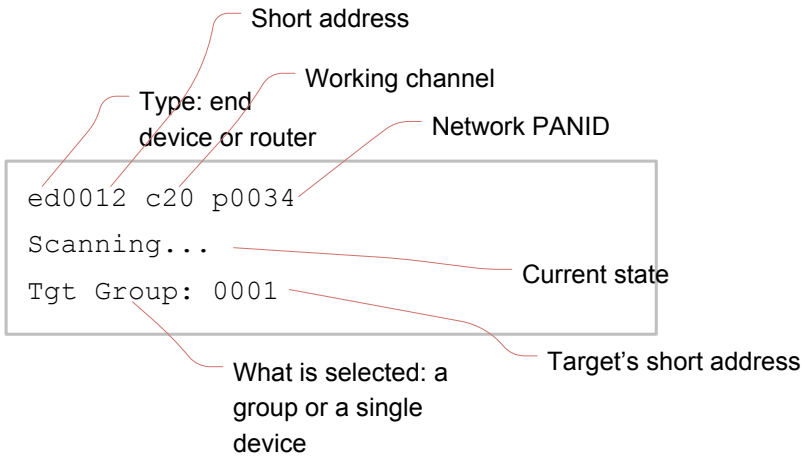
5.7.2 LCD Screen Output

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 on Figure 5-4.

Figure 5-4. Color Scene Controller's LCD Screen on a Key Remote Control Board



5.7.3 Reset to Factory New State

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.

5.7.4 Touch Link/forming a Network

For touch link a color scene controller is brought close to a new light, which is not in the network yet. By pressing PWR button on a color scene controller, the user initiates a commissioning procedure, which goal is to transfer network parameters to the light. At this moment the devices are not yet in the network, but the communication is possible, because it happens on the MAC level without a need for routing.

The light receives network parameters and starts the network as a router (the adjustments in the BitCloud stack allow this). Once the network is started, the color scene controller joins this network as an end device.

When the first light is commissioned, the color scene controller may be brought to the second light, which, receiving the network parameters, will not start the network once again, but will join the existing network as another router. The subsequent lights are commissioned in the same way.

5.7.4.1 Several Color Scene Controllers in a Network

A Light Link network may contain more than one color scene controller. Additional controller devices are added using touch link, which happens between two color scene controllers:

- Bring a factory new color scene controller to a color scene controller that is in the network and hold the PWR button on both devices for more than 3 seconds
- Perform touch link between the new color scene controller and each of the lights
- Use any of the controller devices to manipulate lights

Note: If touch link 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.

5.7.5 Controlling a Light Device – all Commands

Figure 5-5 depicts the key remote board keyboard and all commands that can be sent with the buttons from the color scene controller.

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 5-5.

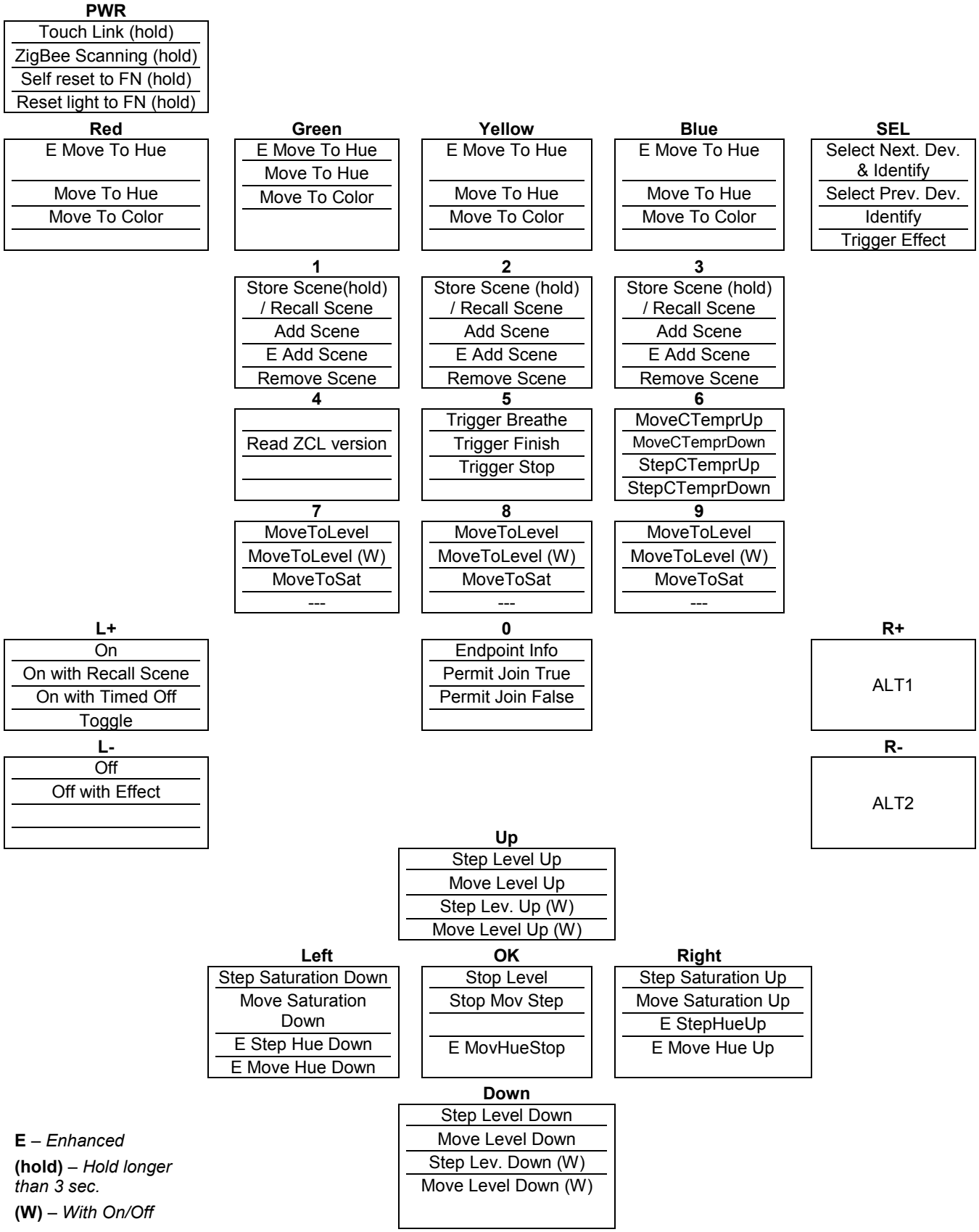
Table 5-5. Alternating a Command by holding R+ and R- Buttons.

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 5-5. Color Scene Controller's Keyboard Scheme



E – *Enhanced*

(hold) – *Hold longer than 3 sec.*

(W) – *With On/Off*

5.8 Over-the-air Firmware Update

Firmware of the nodes in a ZLL network can be updated over the air. For this purpose, the standard Over-the-Air Upgrade (OTAU) cluster is used. Nodes should be programmed with a special version of ZLLDemo application's firmware with OTAU support. To produce such firmware, set the `APP_USE_OTAU` flag to `1` in the `configuration.h` file and recompile the application.

The bridge device connected to a PC serves as the OTAU server, while the bridge GUI application (see Section 5.6.1) on the PC provides the user interface for controlling OTAU process. The user should launch the GUI application, select the *Update firmware* item on the toolbar and, providing a firmware file and specifying the nodes that should be upgraded, start the upgrade process.

For configuration of external Flash memory (used to store application images during OTAU) refer to Section A.1.1.2. For more details on OTAU refer to [22].

5.9 Interoperability with HA Networks

ZLL devices are interoperable with Home Automation (HA) 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 3 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 be also send to the light from the HA dimmer switch device.

5.10 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 5.10.2) and then run certification scripts with its help from the command line (Section 5.10.3).

5.10.1 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 listed below:

- 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's support.

5.10.2 WSNRunner Setup

To install WSNRunner follow the instructions below:

1. Download and install Java Runtime Environment [11], if not already installed on the PC.
2. Download and install Jython [12].

3. Launch `WSNRunnerSetup.exe` located in the `.\Evaluation Tools\` folder of the BitCloud SDK, and follow 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.

5.10.3 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 [7]. The reference application contains three separate applications implementing the following HA device types:

- Combined interface
- Dimmer switch
- Dimmable light
- Occupancy sensor

Other device types can be also implemented as described in BitCloud Developer's Guide [3].

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).

The reference network can include one device of each type to the total of four devices. The combined interface device serves as the network coordinator and the 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 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 the instructions below.

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 A.3.1).

3. To send commands to a device and observe 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
   ```
   (`On` for the XPRO board)

   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.
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 3 minutes interval (using **`startEZMode`** command in serial console). After 3 minutes 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.
8. After devices are paired console commands can be used to communicate between them (see Section 6.4).

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 [3].

Table 6-1. Clusters Supported by HA Devices

Device type	Server clusters	Client clusters
Combined interface	Basic Identify OTAU (if enabled)	Basic Identify OnOff Level Control Groups Scenes Occupancy Sensing
Dimmer switch	Basic Identify	Identify OnOff Level Control OTAU (if enabled)
Dimmable light	Basic Identify OnOff Level Control Groups Scenes	OTAU (if enabled)
Occupancy sensor	Basic Identify Occupancy Sensing	Identify 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.

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\ZLLDemo\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 Table 6-2. Once any of these parameters is changed, the application must be rebuilt and 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_*<device>*	The device type for which a reference application is to be compiled. Put COMBINED_INTERFACE, DIMMABLE_LIGHT, DIMMABLE_SWITCH or OCCUPANCY_SENSOR in place of <device>. ZigBee device type is set for the device by the application according to this parameter, in the appConsts.h file.
APP_*<device>*_EXT_ADDRESS	Precompiled extended value for corresponding device type. Assigned to CS_UID parameter in appConsts.h.
APP_ENABLE_CONSOLE	Set t 1 to enable commands sending through UART. Automatically disables sleep on remote controller and switches to emulated sleep.
APP_DEVICE_EVENTS_LOGGING	Set to 1 to enable output of information on application's events to console
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 and to 0 to disable it
APP_USE_FAKE_OFD_DRIVER	Use fake implementation of the OFD driver. This may be useful for testing OTAU on boards without external Flash.
APP_SUPPORT_OTAU_PAGE_REQUEST	Enable or disable OTAU image page request feature (refer to [22] for details)
EXTERNAL_MEMORY	Specify the type of external memory (where the new firmware image will be stored)

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	Display help instructions
reset	Reset the device
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 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 *combined 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
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
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 allways.
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

Command syntax	Description
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

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 the Step (with On/Off) command
move *<addrMode> <addr>* <ep> <rate> <onOff>	Send Move (with On/Off) command

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 A.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 XPRO board)
```

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.

7.2.2 Demonstrating OTA Upgrade Functionality

Over-the-air demonstration requires one WSNDemo device programmed with an application image supporting OTAU server functionality and a 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 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 [10] 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 [22]. 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 [22] to update the firmware over the air.

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 it indicates that the coordinator is ready and that 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.

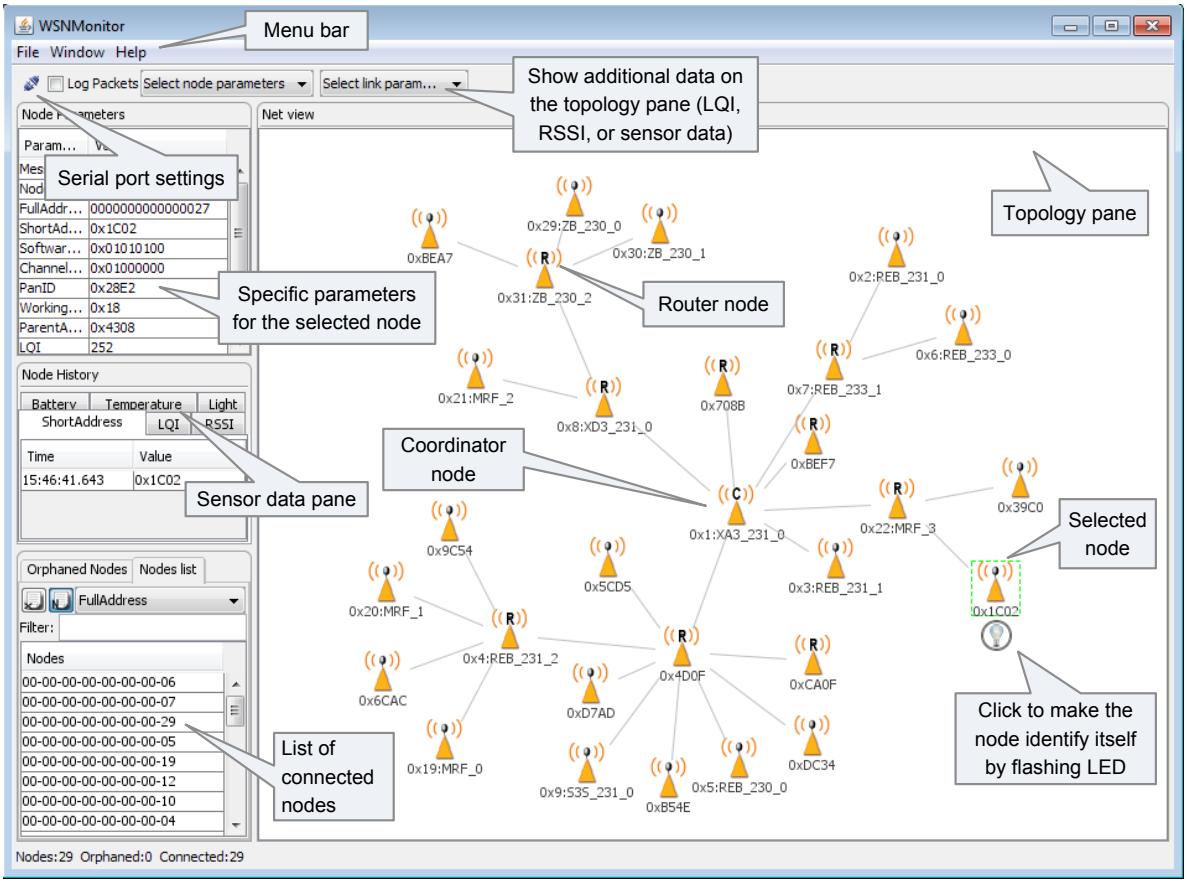
By default, the channel mask for all application images provided with the SDK contains a single channel. In rare cases, if the frequency corresponding to the radio channel is busy, the coordinator node may stay in the network search mode. If this happens, it may become necessary to change the application's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

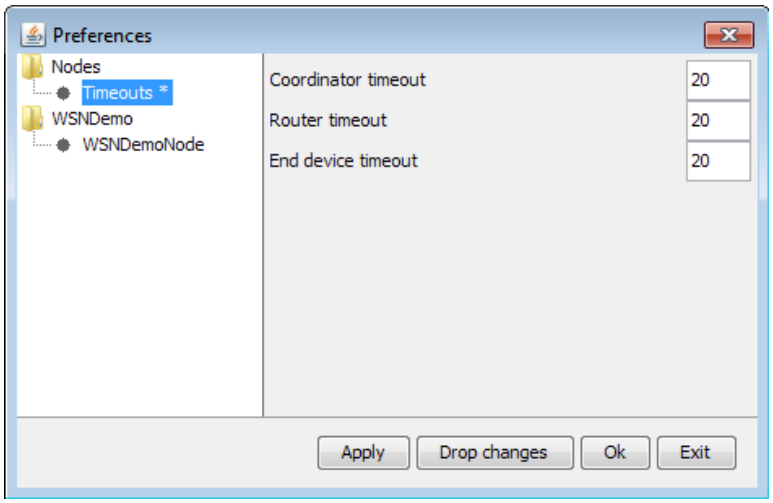
7.5 Identifying Nodes

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 upgrade (OTAU) can be executed on supported hardware platforms by loading a special version of the WSNDemo application with OTAU client cluster support. This demonstration also requires an additional device performing the role of the upgrade server which is the device that sends new firmware images to other devices on the network: this one should be programmed with a version of the WSNDemo application supporting OTAU server functionality. For example, this role may be given to the coordinator. Note that when coordinator acts as OTA server network activity cannot be monitored on the WSNMonitor as same serial interface will be used for communication with OTA bootloader tool [10]. More details on OTA upgrade can be found in corresponding application note [22].

Appendix A. ATmegaRFR2 Specifics

BitCloud supports out of the box two different ATmegaRFR2 development boards with ATmega256RFR2 SoC: AT256RFR2-EK [13] and AT256RFR2-XPRO [15]. 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.

A.1 Hardware Setup

A.1.1 Required Hardware

Make sure that all necessary hardware is available:

- AT256RFR2-EK and Atmel AVR JTAGICE mkII or Atmel JTAGICE3
  or
- two or more AT256RFR2-XPRO boards, each with and 2.4GHz antenna
  or
- two or more custom boards with ATmega256RFR2 or ATmega2564RFR2

A.1.1.1 AT256RFR2-EK Setup

This subsection contains only brief instructions. Detailed hardware description for this kit is given in corresponding user guide [14].

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). Note that different firmware configuration should be used if board specific periphery such as LCD, buttons, serial connections is expected to be used.

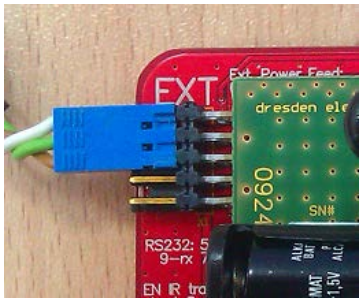
To assemble a device:

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 STB board will be used 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 7-3).

Figure 7-3. 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 7-4 or use the USB connection on the STB board.

Figure 7-4. Serial Cable attached to a Key Remote Control Board



A.1.1.2 AT256RFR2-XPRO Setup

Detailed hardware description for the 256RFR2 Xplained Pro board is given in corresponding user guide [16].

Note: Due to board-specific control, `Hardware Flow` control of the COM port setting on the PC shall be set to `On` 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.

A.1.1.3 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 A-1. More information on the OTA upgrade can be found in [22].

Table A-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

A.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 A.3.

A.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 or Atmel Studio, and programming with Serial Bootloader. Before programming; make sure that the fuse bits are configured correctly, as described in Section A.3.1.

A.3.1 Setting Fuse Bits

Table A-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 [8] for detailed fuse bits description.

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

Table A-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=2048 words start address=$F800	Boot Flash size=2048 words start address=$F800	Specifies section size in words (two bytes) reserved in flash memory for the embedded bootloader Applied only if BOOTRST fuse is enabled Shall be set to `Boot Flash size=2048 words start address=$F800`, if OTAU support is required on the device. For more information see [10]
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 [10]
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	0x9B	0x9B	
Low	0x62	0xF7	

A.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.

A.3.3 Programming with IAR Embedded Workbench

A.3.3.1 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 AVR.
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. Click OK.
10. Select JTAGICE 3/JTAGICE mkII > Fuse Handler.
11. Click Read Fuses, and make sure that the device fuses are set as specified in Section A.3.1.
12. If fuses are set incorrectly, select the correct fuse settings, and click Program fuses.
13. Select Project > Download and debug.
14. Once the debugging session has started, click Stop debugging.

The image is now installed on the board.

A.3.3.2 Application Workspace

1. Double-click on a workspace (for example, HADevice.eww), and select JTAGICE 3/JTAGICE mkII > Fuse Handler.
2. Click Read Fuses, and make sure that the device fuses are set as specified in Section A.3.1.
3. If fuses are set incorrectly, select the correct fuse settings, and click Program fuses.
4. Select the desired application configuration (for example, All_StdlinkSec_MegaRf_ ATmega256RF2_ _8Mhz_Iar) from the drop-down box in the Workspace pane.
5. Select Project > Download and debug.
6. 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.

A.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 A.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.

A.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 A.3.1.

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

Table A-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 [10]).
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.

A.4 Reserved Hardware Resources

Table A-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)
Timer/counter 2	Asynchronous timer (optional – can be disabled)
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

Appendix B. References

[1] AVR2052: Atmel BitCloud Quick Start Guide (this document).

[2] BitCloud API Reference (available in BitCloud SDK).

[3] AVR2050: BitCloud Developer's Guide.

[4] 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).

[5] ZigBee PRO specification (053474r20).

[6] ZigBee Cluster Library specification (075123r04).

[7] 053520r25ZB_HA_PTG Home Automation Profile Specification.

[8] Atmega256(4)RFR2 device.

[9] MinGW C/C++ Compiler.

[10] AVR2054: Serial Bootloader User Guide.

[11] Java Runtime Environment.

[12] Jython.

[13] 256RFR2-EK.

[14] AVR10002: ATmega256RFR2 Evaluation Kit – User Guide.

[15] ATmega256RFR2 Xplained Pro Evaluation Kit.

[16] ATmega256RFR2 Xplained Pro User Guide.

[17] IAR Embedded Workbench for Atmel ARM.

[18] IAR Embedded Workbench for Atmel AVR.

[19] IAR Embedded Workbench IDE User Guide.

[20] JTAGICE3.

[21] AT02597: ZigBee PRO Packet Analysis with Sniffer.

[22] AVR2058: BitCloud OTAU User Guide.

[23] Atmel Studio download.

[24] Atmel Studio archive.

[25] Atmel Studio online help.

[26] AT02698: ZAppSI User Guide.

[27] ZigBee Light Link Profile specification (11-0037-10).

Appendix C. Revision History

Doc. Rev.	Date	Comments
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 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

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Atmel:

ATAVR128RFA1-EK1