AVR2102: RF4Control - User Guide

Atmel MCU Wireless Solutions

Application Note

Features

- RF4Control is the Atmel ZigBee RF4CE Certified Platform
- Architecture overview
- APIs: Network and ZRC Profile
- Example application: Button Controller and Single Button Controller
- Example application: Terminal Target
- Example application: RF4CE Network Serial Interface, ZRC Serial Interface
- Transceiver support – 2.4 GHz: Atmel AT86RF231 and AT86RF233
- Transceiver support – 900MHz: Atmel AT86RF212, AT86RF212B
- MCU support: Atmel's AT xmega256a3 and AT xmega256a3bu
- MCU support: AT32UC3A3256S
- MCU support: Atmel's AT mega256RFR2
- MCU support: AT SAM4L
- Bootloader support for ATmega256RFR2
- Watchdog support.
- NVM Multi-write Support for ATmega256RFR2

1 Introduction

This document is the user guide for the Atmel® RF4Control software stack. The RF4Control stack is a ZigBee® RF4CE Certified Platform implementing the ZigBee RF4CE standard [11].

The RF4Control stack is used with Atmel microcontrollers and IEEE® 802.15.4 transceivers. Some microcontrollers, such as the Atmel ATxmega56a3 [6], are used for reference implementations. Other Atmel microcontrollers can be used based on the application requirements. The ZigBee RF4CE specification makes use of the 2.4GHz band, and Atmel IEEE 802.15.4 transceivers, such as the Atmel AT86RF231 [4], support the 2.4GHz band. In addition, the RF4Control stack supports the sub-1GHz bands, as defined in the IEEE 802.15.4-2006 standard [1], with the Atmel AT86RF212 [3]. For applications requiring the use of a single-chip implementation (transceiver and microcontroller SoC), the Atmel megaRF family provides such a single-chip solution. As a reference, the ATmega256RFR2 [5] is used.

This user guide introduces the RF4Control architecture and its implementation in section 2. Based on the stack, several example applications are implemented demonstrating the use of the stack's functionality and APIs. Chapter 3 describes the example applications.

Remote controlling is the main application area for RF4CE, and the Example applications section introduces a few application examples (Terminal Target and Key Remote Controller). Section 3.3 introduces a Single Button Controller example application and walks through its implementation. The Key Remote Controller,

Rev. 8357E-AVR-02/13

which uses the ATmega128RFA1 Radio Controller Board (RCB), is a certified ZigBee Remote Control application.The software stack provides an API that is aligned with the RF4CE network primitives, and which can be used directly from an application or firmware. A serial interface API is also provided. The serial interface API can be used for communication where the Atmel RF4CE stack is hosted on a separated communication microcontroller and controlled by an additional microcontroller via, for example, a UART, SPI or I^2C serial interface. The serial interface approach is described in section 3.4. An example application demonstrates using the serial interface API with an UART interface (see section 3.4.6).

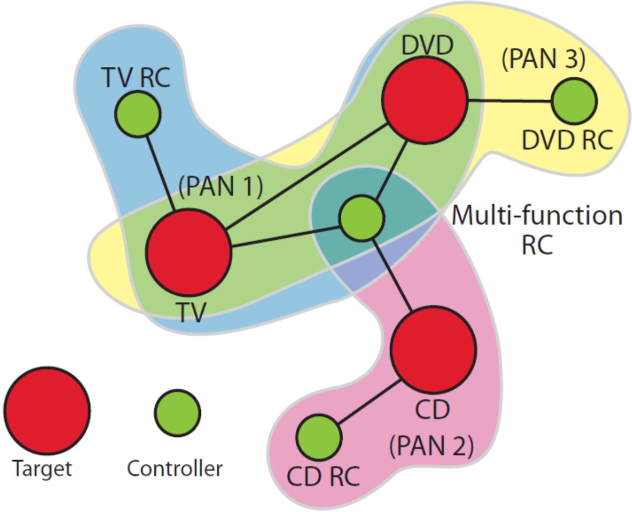
When working with the RF4Control stack, it is highly recommended to use the ZigBee RF4CE specification version 1.0 [12]. Terms used throughout this document are based on the ZigBee RF4CE specification. The ZigBee RF4CE also specified a profile for remote control applications – ZigBee Remote Control Profile (ZRC) [13]. Use this specification also as an additional source of information.

1.1 Remote controlling

Remote controlling is the main application scope of the RF4CE standard. The first profile published (ZigBee Remote Control profile, ZRC [13]) addresses the remote controlling of consumer goods.

The RF4Control package contains a remote control example application in which one board represents a TV (target node) while the other board represents a remote controller (controller node). The end-user applications on both boards use the ZRC profile, as defined by the RF4CE specification. A typical RF4CE network example is shown in Figure 1-1.

Figure 1-1. RF4CE network topology example.



Source: ZigBee RF4CE [11]

Nodes can be made known to each other using a procedure called pairing. The ZRC profile specification describes an automated/simplified pairing procedure, called push button pairing, between a target node and a controller node.

Besides the pairing procedure, the profile points to the HDMI specification [14] for the actual controller command codes (CEC – Consumer Electronics Control).

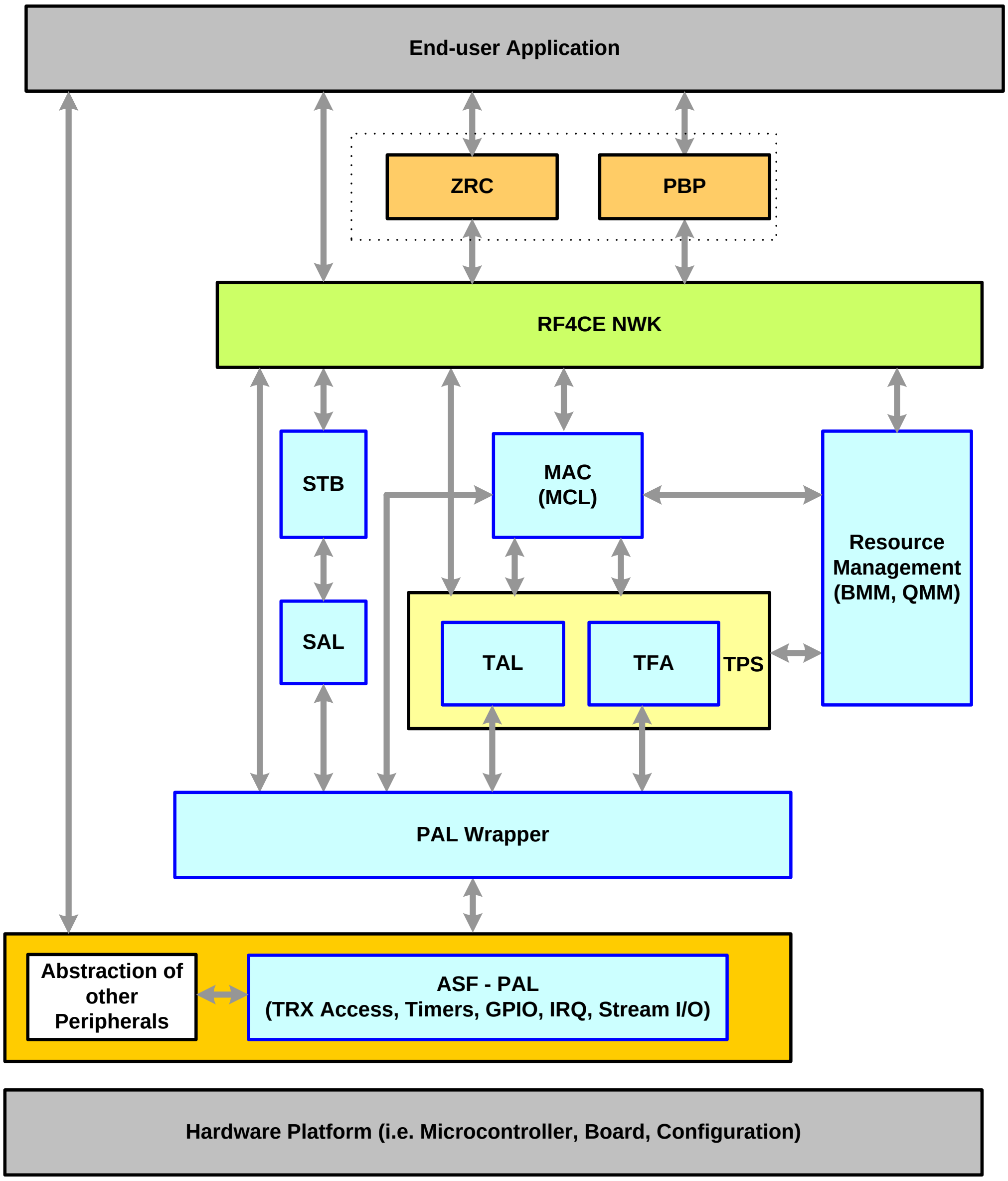
2 RF4Control – Stack implementation

2.1 Architecture

The Atmel RF4Control stack uses the Atmel IEEE 802.15.4 MAC as the underlying layer. For detailed information about the MAC layer, see the AVR2025 MAC Software Package [7].

Figure 2-1 shows the software architecture used for RF4Control stack implementation.

Figure 2-1. RF4Control software stack architecture.



The end-user application accesses the RF4CE network layer directly for initialization and configuration. If the ZRC profile is part of the configuration, the ZRC profile functions support the initialization and data exchange.

The Atmel MAC software implementation is modular, allowing different hardware to be used for RF4CE applications. The microcontroller and board are interfaced using the Platform Abstraction Layer (PAL). The transceiver is interfaced using the Transceiver Abstraction Layer (TAL). For further information about these layers, see the MAC software package user guide [7].

2.2 ZigBee Remote Control profile

The ZigBee Remote Control (ZRC) profile defines the protocol (structure and sequence of communication messages) between a ZRC-compliant remote control (RC) device and a ZRC-compliant target device, such as a TV, DVD, etc.

The ZRC profile is specified by [13]. Compared to the RF4CE network specification [12], the ZRC profile specification does not define primitives as Service Access Points (SAP). Therefore the primitives descriptions cannot be used as RF4Control API descriptions as is done within the network specification. The RF4Control API for the ZRC profile is described within the following sections. For detailed information about the API function, see also the reference manual provided in HTML format in section 2.5.

The ZRC profile interfaces to the RF4CE network layer to make use of the network's standardized pairing and data transmission mechanisms. The pairing mechanism specified by the ZRC profile is called push button pairing (PBP), and it includes the discovery and pairing mechanisms. Push Button Pairing is described in section 2.2.1.

The ZRC profile also defines RC command discovery and RC command handling procedures. These procedures are described in sections 2.2.2 and 2.2.3, respectively.

In general, ZRC profile features are included in the firmware build if the *ZRC_PROFILE* flag is defined within the Makefile or the IAR™ project file. Section 2.6 provides an overview of the build configuration.

2.2.1 Push button pairing

The push button pairing procedure uses and combines the discovery and pairing mechanisms of the RF4CE network layer. After getting a user stimulus (Button Press or PBP API call) on the controller, the PBP procedure automatically starts a discovery procedure. The target device enters the auto-discovery response mode if triggered by a Button Press or PBP API call. Once the discovery is successful, it automatically starts the pairing procedure.

Dedicated PBP API functions are used by the target and controller nodes. Some PBP API function parameters are used for discovery, and the remaining ones are used for the actual pairing. Table 2-1 lists them for the target and Table 2-2 lists them for the controller as implemented by the RF4Control stack. They are declared in the *pb_pairing.h* header file.

The PBP functionality is included in the firmware build if the *PBP_ORG/PBP_REC* flag is defined within the Makefile or the IAR project file. If the PBP_ORG/PBP_REC flag is set, the PBP API functions are included and the discovery and pairing API functions are hidden from the higher layers. The discovery and pairing functions are used by the PBP implementation. The discovery and pairing API functions are exposed to the application if PBP_REC/PBP_ORG is not set. Section 2.6 provides an overview of the build configuration.

Table 2-1. Push button pairing API – target side.

API function	Description
pbp_rec_pair_request (RecAppCapabilities, RecDevTypeList, RecProfileIdList, pbp_rec_pair_confirm)	Push button pairing recipient request: Initiates the push button pairing on the target side. Internally, the target starts the auto-discovery procedure. After successful discovery, it handles the incoming pairing request. RecAppCapabilities: The application capabilities of the target node (the device number and profile type supported by the target node). RecDevTypeList: The list of the supported device types. RecProfileIdList: The list of the supported profile types. pbp_rec_pair_confirm: Confirmation callback for the request
pbp_allow_pairing (Status, SrcIEEEAddr, OrgVendorId, OrgVendorString, OrgUserString, KeyExTransferCount)	Push button pairing allow pairing: Provides information to the target application about the incoming pairing request from the controller node. The application placed on the target can decide whether or not to allow pairing based on this information. Status: Status of the pair indication; here NWK_SUCCESS or NWK_DUPLICATE_PAIRING. SrcIEEEAddr: IEEE address of the device (controller) requesting to pair. OrgVendorId: Vendor identifier of the device (controller) requesting to pair. OrgVendorString: Vendor string of the device (controller) requesting to pair. OrgUserString: User string of the device (controller) requesting to pair. KeyExTransferCount: Key exchange transfer count of the incoming pair request.
pbp_rec_pair_confirm (Status, PairingRef)	Push button pairing confirm: This callback function provides the status of the push button pairing request. Status: Status of the push button pairing procedure. PairingRef: If pairing was successful, it contains the assigned pairing reference.

Table 2-2. Push button pairing API – controller side.

API Function	Description
pbp_org_pair_request (OrgAppCapabilities, OrgDevTypeList, OrgProfileIdList, SearchDevType, DiscProfileIdListSize, DiscProfileIdList, pbp_org_pair_confirm)	Push button pairing originator pair request: Initiates the push button pairing on the controller side. Internally, the controller starts the discovery procedure. After a successful discovery, it automatically sends the pairing request to the target. OrgAppCapabilities: Application capabilities of the controller node. OrgDevTypeList: The list of the supported device types. OrgProfileIdList: The list of the supported profile types. SearchDevType: The device type that the controller is looking for (i.e., a TV). DiscProfileIdListSize: The size of the DiscProfileIdList (the next parameter). DiscProfileIdList: The list of profile identifiers against which profile identifiers contained in the received discovery response will be matched. pbp_org_pair_confirm: Confirmation callback for the request
pbp_org_pair_confirm (Status, PairingRef)	Push button pairing pair confirm: This callback function provides the status of the push button pairing request. Status: Status of the push button pairing procedure. PairingRef: If pairing was successful, PairingRef contains the assigned pairing reference.

2.2.2 Command discovery

The command discovery procedure enables a target or controller to query the CEC commands supported by the other node. The other node can respond by sending a command discovery response frame containing a bitmap of its supported CEC commands. The command discovery API is described in Table 2-3.

The command discovery functionality is included in the firmware build if the *ZRC_CMD_DISCOVERY* flag is defined within the Makefile or the IAR project file. Section 2.6 provides an overview of the build configuration.

Table 2-3. ZigBee remote control command discovery APIs.

API Function	Description
zrc_cmd_disc_request (PairingRef, zrc_cmd_disc_confirm)	Sends command discovery request command to other node. PairingRef: The pairing reference for the other node obtained during the push button pairing procedure. zrc_cmd_disc_confirm: Confirmation callback for the request
zrc_cmd_disc_confirm (Status, PairingRef, SupportedCmd	This callback function provides the status and supported command information from the other node. Status: Status of the command discovery request. SupportedCmd: The CEC commands that the responding node supports.

API Function	Description
zrc_cmd_disc_indication (PairingRef)	Indicates to the sending device that a command discovery request is received. PairingRef: The pairing reference of the originator node.
zrc_cmd_disc_response (PairingRef, SupportedCmd)	Allows a device to respond to an incoming command discovery request frame. PairingRef: The pairing reference of the originator node. SupportedCmd: The CEC commands that this node supports.

2.2.3 RC command handling

RC command handling allows a controller node to send the RC command (CEC) to a target node to perform the specified operation. For example, when a user presses a “channel up” button on the remote controller, it sends a command over the air to the target device (such as a TV) to increment the channel.

Three types of over-the-air commands are defined in the ZRC specification:

1. PRESSED command – When a user presses an RC button, the PRESSED command is sent to the target
2. REPEATED command – If the user holds down a remote key for some time, multiple REPEATED commands can be sent to the target
3. RELEASED command – To stop the operation of a target device (TV, for example), the user releases the pressed RC button and a RELEASED command is sent

The Button_Controller application example supports all three command types, while the Single Button Controller application example uses only the PRESSED command type.

The REPEATED and RELEASED functionality is excluded from the firmware build if the *ZRC_BASIC_PRESS_ONLY* flag is defined within the Makefile or the IAR project file. If the ZRC_BASIC_PRESS_ONLY compiler switch is set, only the basic PRESSED functionality is supported by the implementation. Section 2.6 provides an overview of the build configuration.

The API for sending the commands is shown in Table 2-4.

Table 2-4. RC command APIs.

API Function	Description
zrc_cmd_request (PairingRef, VendorId, CmdCode, CmdLength, Cmd, TxOptions, zrc_cmd_confirm)	Initiates the RC command request (key code) by the application. PairingRef: The pairing reference for the other node. VendorId: Vendor identifier; only use if vendor data transmit option is set. CmdCode: Specifies a command code. This could be a PRESSED command (device menu, for example) or a REPEATED command (volume up, for example). CmdLength: Length of the command payload. Cmd: Contains the CEC command and payload (if anything). TxOptions: Tx options, as defined in the RF4CE network layer specification. zrc_cmd_confirm: Confirmation callback for the request

API Function	Description
zrc_cmd_confirm (Status, PairingRef, RcCmd)	Provides the confirmation of a command request to application. Status: Status of the RC command request. PairingRef: The pairing reference for the other node. RcCmd: The RC (CEC) command to be sent.
zrc_cmd_indication (PairingRef, nsduLength, nsdu, RxLinkQuality, RxFlags)	Indicates that an RC command request command has been received. PairingRef: The pairing reference of the originator node. nsduLength: The length of the received RC command. nsdu: RC command payload. RxLinkQuality: Received link quality. RxFlags: Rx flags, as defined in the RF4CE network layer specification.

2.3 Channel agility

The RF4CE standard's frequency agility mechanism can be used to overcome a jammed RF channel scenario. Although, the standard specification refers to *frequency* agility, in reality *channel* agility is meant. In the context of the RF4Control stack, the term “channel agility” is used.

The following paragraphs describe the design constraints and the implementation / usage of the channel agility mechanism to supplement the RF4CE standard.

To detect a channel compromised by an external source of interference, a mechanism called energy detection (ED) is employed. This functionality is provided by the MAC layer, and is operated via ED scans. During ED scans the device cannot receive any frames. Long or frequent scans result in dead times. To avoid long offline durations, the most recently used channel (BaseChannel) is scanned first. If the measured channel energy exceeds the maximum ED threshold, all three channels are scanned in sequence, and the channel with the lowest energy is set as the new BaseChannel.

The Atmel RF4Control stack provides a set of API functions allowing the user to control the usage and behavior of the ED scans in the context of channel agility. Table 2-5 lists the API functions and their parameters that can be used to control the channel agility mechanism. The channel agility feature needs to be started by the application using the nwk_ch_agility_request() API function, and it is then handled automatically by the stack.

The channel agility API functions are included in the build process if the *CHANNEL_AGILITY* compiler switch is defined within the Makefile or the IAR project file. Section 2.6 provides an overview of the build configuration.

Table 2-5. Channel agility API functions.

API Function	Description

API Function	Description
nwk_ch_agility_request (AgilityMode, nwk_ch_agility_confirm)	Enables or disables the channel agility mode. AgilityMode: *AG_ONE_SHOT* - starts single scanning *AG_PERIODIC* - starts periodic scanning *AG_STOP* - stops periodic scanning nwk_ch_agility_confirm: Confirmation callback for the request
nwk_ch_agility_confirm (Status, ChannelChanged, LogicalChannel)	Confirms the previous call of the above request. Status: Status of the request. ChannelChanged: True if the channel has changed, else false. LogicalChannel: Current logical channel.
nwk_ch_agility_indication (LogicalChannel)	If the channel is changed during the periodic mode, this indication informs the application about it. LogicalChannel: New/current logical channel.
nlme_set_request (NIBAttribute, NIBAttributeIndex, NIBAttributeValue, nlme_set_confirm)	Sets the configuration parameters (NIBAttribute), such as *nwkPrivateChAgScanInterval*: Channel agility scan interval, set to 60s for example applications; *nwkPrivateChAgEdThreshold*: Channel agility ED threshold value, set to 10 (-80dBm) for example applications; *nwkScanDuration*: duration of a single scanning operation, set to 6 (~1s) for example applications. nlme_set_confirm : Confirmation callback for the request

For more details of the actual API functions, see the HTML-based reference manual; section 2.5.

2.4 Vendor-specific data handling

The RF4CE profiles define standard behavior to ensure compatibility between different vendors. But some application requirements are not covered by the profile. These requirements can be handled by application-specific frames. The RF4CE standard allows transmitting application-specific frames using vendor data frames handled in the profile context.

The Atmel RF4Control stack supports mechanisms (application hooks) for a dedicated vendor specific data exchange in the ZRC profile context. These mechanisms ensure the correct data handling without any impact on the standard profile-specific data handling. Table 2-6 shows the API functions for vendor data handling. The function prototypes can be found in the *vendor_data.h* header file located in the RF4CE/Inc directory.

The vendor-specific API functions are included in the build process if the *VENDOR_DATA* compiler switch is defined within the Makefile or the IAR project file. Section 2.6 provides an overview of the build configuration.

Table 2-6. Vendor data handling API functions.

API Function	Description
vendor_data_request (uint8_t PairingRef, profile_id_t ProfileId, uint16_t VendorId, uint8_t nsduLength, uint8_t *nsdu, uint8_t TxOptions)	Initiates a vendor data specific transmission. PairingRef: The pairing reference for the other node. ProfileId: Profile identifier used for the transmission. VendorId: The vendor identifier. If this parameter is equal to 0x0000, the vendor identifier of the stack is used. nsduLength: The number of octets contained in the payload/nsdu. Nsdu: Payload of the data frame. TxOptions: Transmission options for this command; see ZigBee RF4CE specification version 1.0 [12] for further details.
vendor_data_ind (uint8_t PairingRef, uint16_t VendorId, uint8_t nsduLength, uint8_t *nsdu, uint8_t RxLinkQuality, uint8_t RxFlags);	Indicates an incoming vendor specific data frame PairingRef: The pairing reference of the originator node. VendorId: The vendor identifier used by the originator. nsduLength: The number of octets contained in the payload/nsdu. nsdu: Payload of the data frame. RxLinkQuality: Link quality of the incoming frame. RxFlags: Information about the transmit modes used.
vendor_data_confirm (nwk_enum_t Status, uint8_t PairingRef, profile_id_t ProfileId, uint8_t Handle);	Provides the status of the last vendor data request. Status: Status of the data transmission. PairingRef: The pairing reference used for the transmission. ProfileId: Profile identifier used for the transmission. Handle: Used for data retry at the application level

The application needs to define and handle the semantics of the vendor data payload.

Some example applications, such as the Single Button Controller (section 3.3.5.2) in combination with the Terminal Target application, demonstrate the use of the vendor data exchange.

The ZRC Target (section 3.1.3 application example reveals the concept of vendor data exchange by implementing a firmware over-the-air (FOTA) upgrade feature.

2.5 RF4Control firmware API

The Atmel RF4Control stack API is documented using Doxygen-style comments.

2.6 Stack configuration

The RF4Control stack can be configured to match end-user application requirements. The configuration ensures that only functionality that is actually needed by the application is included into the stack and that the footprint meets the desired or minimum values.

The configuration is done in the same way as it is within the MAC software package [7]; see its user guide for general information about stack configuration.

The RF4Control stack uses as the default CPU clock 16 MHz while be run on a megaRF device. Depending on the application requirements the CPU clock can be reduced (from the default 16 MHz operation) to 4 or 8 MHz by setting the define F_CPU to 4000000 or 8000000 in the pal_config.h file. Reducing the CPU clock has impact to the execution speed of the entire application.

The Atmel RF4Control stack can be configured by build/compiler switches. It is defined within the app_config.h file, and is applicable to source code package releases only.

Table 2-7. Compiler/build switches.

API Function	Description
RF4CE_PLATFORM	If set, stack supports all device types. The actual device type needs to be configured by the application. This compiler switch includes also the build switch RF4CE_SECURITY.
RF4CE_TARGET	If set, stack supports functionality that is required to operate a target node. If not set, the stack only supports functionality that is required to operate a controller node.
RF4CE_SECURITY	If set (default), security is supported. If not set, the stack does not support security and the footprint is smaller. If set, the compiler switch STB_ON_SAL is required too.
RSSI_TO_LQI_MAPPING	If set (default), LQI calculation is based on RSSI value, as defined by [12].
MAC_USER_BUILD_CONFIG	If set (default), MAC user build configuration is enforced. Only MAC primitives required by the RF4CE network layer are included in the build process.
NWK_USER_BUILD_CONFIG	If set, the nwk_user_build_config.h file is included during the firmware build process. The header file contains compiler switches to enable or disable network layer features that are required or not required by the application. The Makefile / IAR project file needs to include the path to the nwk_user_build_config.h file.
TFA_BAT_MON	If included in the Makefile or IAR project file, the supply voltage measurement feature is available.
VENDOR_DATA	If included in the Makefile or IAR project file, the hooks to handle vendor specific data are available.
FLASH_SUPPORT	If included in the Makefile or IAR project file, functionality for self programming the flash are available.
ZRC_PROFILE	If included in the Makefile or IAR project file, the ZRC profile layer is included in the build process.
ZRC_CMD_DISCOVERY	If included in the Makefile or IAR project file, the command discovery functionality is available.
PBP_ORG	If included in the Makefile or IAR project file, the push button pairing originator functionality is available. This build switch needs to be set if ZRC_PROFILE is set.

API Function	Description
PBP_REC	If included in the Makefile or IAR project file, the push button pairing receipient functionality is available. This build switch needs to be set if ZRC_PROFILE is set.
CHANNEL_AGILITY	If included in the Makefile or IAR project file, the channel agility feature is included to the build process.
ZRC_BASIC_PRESS_ONLY	If included in the Makefile or IAR project file, the ZRC profile supports only the PRESSED command code. REPEATED and RELEASED are not available.
ENABLE_PWR_SAVE_MODE	If included in the Makefile or IAR project file, receiver is set to power save mode.
NO_32KHZ_CRYSTAL	If included in the Makefile or IAR project file, sleep functions are configured to operated without a 32 kHz crystal in place. This is used to demonstrate the implementation w/o 32KHz crystal on Single Button Controller application. If this build switch is used, the WATCHDOG_TIMER switch needs to be set as well.
STORE_NIB	If included in the Makefile or IAR project file, NIB is stored in the flash memory instead of EEPROM.
NVM_MULTI_WRITE	If included in the Makefile or IAR project file, frame-counter is stored in the flash memory instead of EEPROM.
WATCHDOG	If included in the Makefile or IAR project file, watchdog feature is enabled.
WATCHDOG_TIMER	If included in the Makefile or IAR project file, watchdog is enabled in the interrupt mode.
BOOT_FLASH	If included in the Makefile or IAR project file, bootloader support will be enabled and functionality for self programming the flash will be available through bootloader.
NLDE_HANDLE	If included in the Makefile or IAR project file, application/profile will be provided with the handle argument for network data retry handling.
RF4CE_CALLBACK_PARAM	If included in the Makefile or IAR project file, application/profile will be provided with the callback parameter for the confirmation.

Compiler/build switches others than those listed in Table 2-7 configure the underlying MAC layer and its transceiver and platform abstraction. See [7] for further information on MAC layer configuration.

Some special stack configurations are described below.

2.6.1 WATCHDOG

The watchdog, i.e. system reset, is enabled by including the WATCHDOG in the Makefile or IAR project file. By default it is enabled for all the supported platforms.The watchdog timeout is configured by the WDT_TIMEOUT_PERIOD in the app_config.h file. An example configuration, this value is set to eight/four seconds.

The controller has a watchdog module whose basic purpose is to trigger a system reset and start executing from the boot vector in case the program hangs due to some fault condition. When the system is running fine it regularly services the Watchdog timer by clearing it periodically.

2.7 Stack porting

This user guide describes how to use the Atmel RF4Control stack using a few example boards. For a customer- or application-specific design, the existing stack usually needs to be ported to a new hardware platform. The RF4Control stack is designed in a way that abstracts the hardware-specific characteristics through lower layers (Platform Abstraction Layer – PAL, Figure 2-1).

Because the higher layers, such as the MAC, network, and profile layers, are implemented independently from the underlying hardware platform, no changes are usually required to these layers.

It is recommended to use an existing hardware platform and software application as a basis for customer development. The application examples provided in chapter 3 are a good starting point for your own application development.

The “Platform Porting” section of the AVR2025 user guide [7] describes how to port from one hardware platform to another.

3 Example applications

The RF4Control stack package contains some example applications that can be used for demonstration purposes and for getting familiar with the implementation for customer application development. For demonstration purposes, the release package includes pre-compiled firmware binary files (in .hex file format using the GCC compiler or .d90/.a90 file format using the IAR compiler). These can be used out of the box.

3.1 Button Controller example application

3.1.1 Introduction

The button controller example application implements a button controller and its target, which represents a TV, DVD, STB or similar device.

For the button controller, Atmel uses designated hardware called a Button Controller. The counterpart of the button controller is the Terminal Target or ZRC Target application. See 3.2 for further information about the Terminal Target setup, and section.

The Terminal Target’s user interface is realized by using a standard terminal program, such as Windows® HyperTerminal. The target is controlled via the Terminal target applications, for information about the ZRC Target application. program, and the received button control commands are printed to the terminal program.

The handling of the Button Controller example application is described in section 3.2.1. The simpler button controller application, called a Single Button Controller, is described in section 3.3

3.1.2 Button Controller board setup

The button controller setup consists of two boards connected together: (1) XMEGA-A3BU-XPLAINED board and (2) the Transceiver board supports 2.4GHz and Sub GHz. The XMEGA-A3BU-XPLAINED board holds buttons, LED, USB Interface and Display. RF Communication handled by the Transceiver Board.

Figure 3-1 shows the XMEGA-A3BU-XPLAINED button controller application board.

Currently following transceivers are supported for the button controller pre-build image:

1. XMEGA-A3BU-XPLAINED Board with Atmel AT86RF231
2. XMEGA-A3BU-XPLAINED Board with Atmel AT86RF230B

If the example application is to be used in the sub-1 GHz band, the following board is supported:

3. XMEGA-A3BU-XPLAINED Board with Atmel AT86RF212

Board Setup

Connect the supported Transceiver Board to the XMEGA-A3BU-XPLAINED Board.

Insert the micro USB cable to the XMEGA-A3BU-XPLAINED & other end to PC or Laptop. Connect the ISP or JTAG to the XMEGA-A3BU-XPLAINED Board.

Button controller application available with Atmel Software Frame Work which is selected from ASF Wizard Example Application in Atmel Studio.

For IAR Projects download the ASF standalone zip file from http://www.atmel.com/asf

Extract the downloaded files into the director. The IAR workspace files are available from the below path

\thirdparty\wireless\avr2102_rf4control\apps\zrc\button_ctr\xmega_a3bu_xplained_rz600rf231\iar

\thirdparty\wireless\avr2102_rf4control\apps\zrc\button_ctr\xmega_a3bu_xplained_rz600rf212\iar

It is recommended to check the MCU fuses: Table 3-1 lists the recommended fuse settings. For further information about fuse settings, see [7] and the device datasheet.

Table 3-1. Recommended fuse settings.

Parameter	Value for RCB
BODLEVEL	Brown-out detection at VCC = 1.8V
OCDEN	Disabled
JTAGEN	Enabled
SPIEN	Enabled
WDTON	Disabled
EESAVE	Enabled
BOOTSZ	Boot flash size = 4096 words; start address = $F000
BOOTRST	Disabled
CKDIV8	Disabled
CKOUT	Disabled
SUT_CKSEL	Internal RC oscillator start-up time = 6CK + 0ms

Fuse settings can also be specified in terms of bytes as given below -

Extended : 0xFE

High : 0x91

Low : 0xC2

3.2 Terminal target example application

The Terminal Target example application, which represents a TV, DVD, etc., can be operated using several boards. The pre-compiled firmware for the supported boards is located in the directory:

```
\thirdparty\wireless\avr2102_rf4control\apps\zrc\terminal_ \<
  mcu_tranceiver_board>\GCC
```

or

```
\thirdparty\wireless\avr2102_rf4control\apps\zrc\terminal_tgt\<mcu_tra
  nceiver_board>\ IAR\Exe
```

where <mcu_tranceiver_board > represents the used hardware configuration, such as at32uc3a3256s_rz600_at86rf212.

The AVR2025 User Guide ([7], section 7.3) provides further information about firmware programming using AVR Studio.

Table 3-1 contains information about the recommended MCU fuse settings.

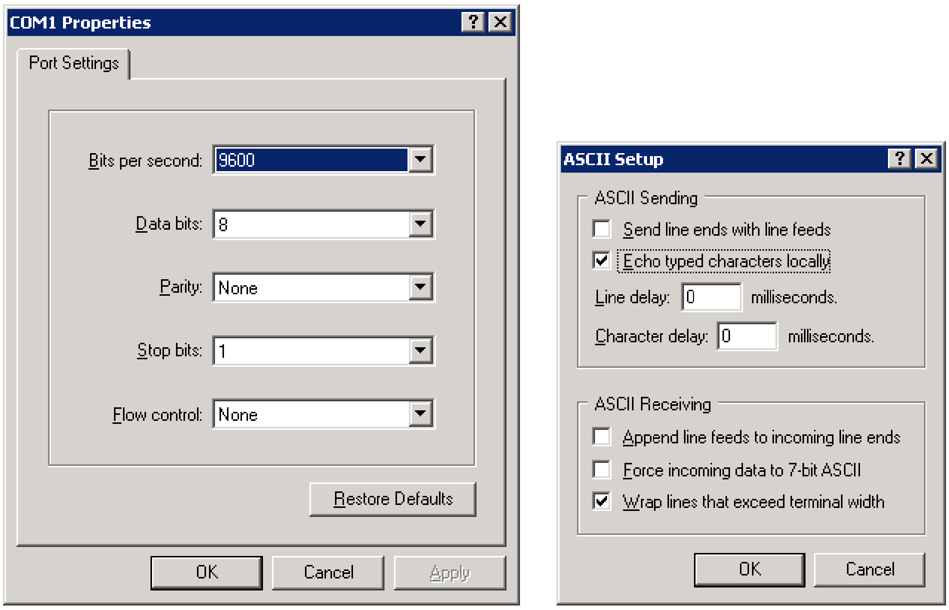
The board used for the Terminal Target application needs to be connected to a PC/laptop via a serial interface; that is, an RS232/UART or USB interface. The required USB drivers can be found here:

- Atmel CDC USB driver used with XMEGA-A3BU-XPLAINED board: [15]

At the PC/laptop, a terminal program (Windows HyperTerminal, for example) is used to control the Terminal Target application. Figure 3-1 shows the configuration of the HyperTerminal program used for the example application.

- If at32uc3a3256s_rz600_at86rf212 is used as terminal target, then Baudrate needs to be set to 9600

Figure 3-1. HyperTerminal settings.

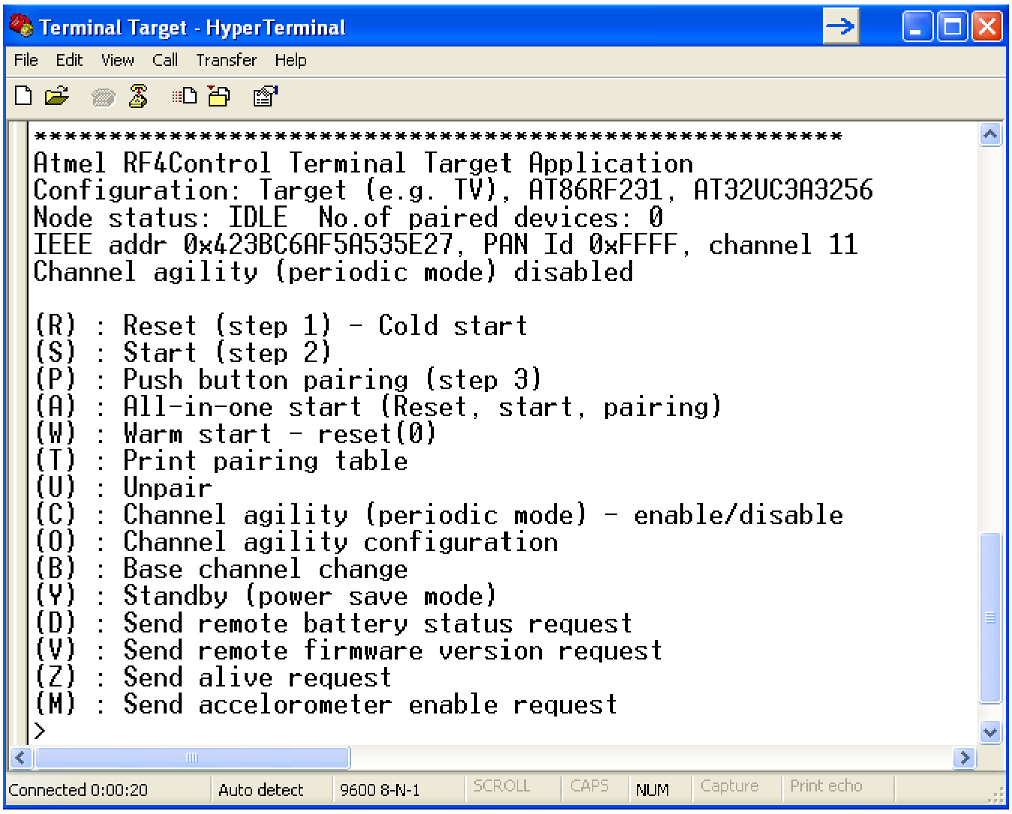


3.2.1 Remote controlling operations

3.2.1.1 Terminal target functions

Once the Terminal Target application is powered up, open the terminal program and press any key to print the menu to the terminal window. Figure 3-2 shows the terminal window with the application menu.

Figure 3-2. Terminal Target example application menu.



The following actions can be triggered from the menu by entering a letter in the HyperTerminal window.

(R) Perform a cold reset of the target device; NIB will be reset to default values and stored in EEPROM
(S) Start the target device
(P) Start the pairing procedure on target device
(A) All-in-one start-up. Perform all three previous steps; that is, reset, start, and pairing
(W) Perform a warm reset of the target device
(T) Print the pairing table
(U) Unpair a device and remove pairing entry from the pairing table of target
(C) Enable channel/frequency agility on the target device
(O) Open a sub-menu to configure channel agility
(B) Set the base channel on the target device
(Y) Toggle the standby mode of the target device. Target will sleep and then wake up for 16.8ms every second. If target receives any data in 16.8ms window, it will come out of standby mode
(D) Request the battery status from the controller. The target sends a battery status request to the controller. The controller will send the response. The target sends the request command continuously for one second (multi-channel mode) until the controller wakes up (16.8ms window) to receive the data
(V) Request the firmware version from the remote controller. The target sends a battery status request to the controller. The controller replies with the response. The target sends the request command continuously for one second (multi-channel mode) until the controller wakes up (16.8ms window) to receive the data

(Z) Request the remote controller life status. The target sends an alive request to the controller. The controller replies with the response. The target sends the request command continuously for one second (multi-channel mode) until the controller wakes up (16.8ms window) to receive the data. The LEDs on the controller will blink for some time indicating that an alive request is received

(M) Request the remote controller to enable the accelerometer for a defined duration(ON duration) and send the accelerometer position to the target periodically(200 ms).After receiving this request, the controller will blink once for indication and start sending the accelerometer position at regular interval(200 ms) till the ON duration expires.

3.2.1.2 Remote controller clearing

The remote controller might have stored any data to the microcontroller EEPROM from previous operations. Therefore, it is recommended to clear any data that is stored in the EEPROM and reset any previously stored pairing information. The pairing table is stored in the MCU EEPROM.

The remote controller application including EEPROM is cleared by executing a cold start reset. The cold start reset is initiated by pressing the SEL button first then keeping SEL button pressed, hold down the PWR button. The application indicates that it is ready for clearing when the LEDs next to the SEL & PWR buttons turn on. Releasing the SEL & PWR buttons clears all stored data. The clearing procedure is completed when LED ON. After clearing all previously stored data (expect IEEE address), the remote controller application sets itself to sleep mode for power saving. In some scenarios LED ON indicates that a problem has been detected. For example, the application detects that it is not paired to any other device.

3.2.1.3 Pairing

In order to control the Terminal Target by the Button Controller board, it is necessary to pair both boards with one another. The pairing procedure, called push button pairing, is defined by the ZigBee RF4CE Remote Control profile specification.

Using the example application, the easiest way to execute push button pairing is as follows:

Step 1: Enter 'A' at the terminal program to execute an "All-in-one start." This includes the reset of the node, initialization of the ZRC profile, start of the network layer, and the auto-discovery procedure as part of the push pairing sequence. The terminal program indicates that it is ready for the push button pairing procedure by printing "Press SEL key then keeping SEL pressed press any FUNC key This starts the push button pairing at the button controller." and by flashing all the LEDs.

Step 2: Start the push button pairing procedure on the button controller board by pressing the SEL button first then keeping the SEL button pressed, hold down one of Target-1 or Target-2 function key. The output of a successful pairing sequence is shown in Figure 3-3.

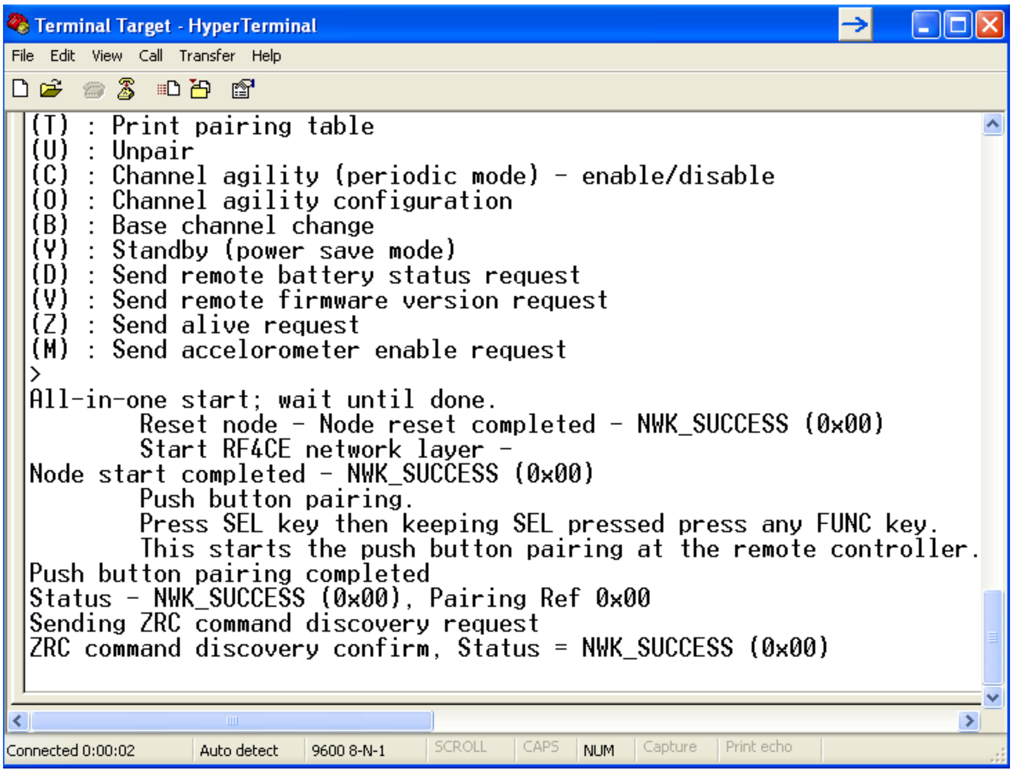
The information stored into the Terminal Target pairing table can be listed by selecting 'T' from the target menu.

Now the Button Controller can be used to send commands to the target.

The button controller board can be used to control different targets. For example, the Target-1 key can be used to control a TV and a Target-2 key can be used to control another target, like a DVD. The function key pressed during the pairing procedure determines the target node to be controlled.

If the all-in-one start is not used to establish the pairing, the manual sequence needs to be as follows: Reset, start, and then push button pairing.

Figure 3-3. Terminal Target example application output – successful pairing.



In order to control another target device by the same button controller, the push button pairing procedure needs to be repeated with another Terminal Target application using a different Target key.

The Terminal Target example application is limited to three paired devices/controllers at a time.

3.2.1.4 Operation

After successful pairing of the two boards (target and controller), the Button Controller board can be used to control the Terminal Target application. The command code (HDMI CEC [14]) of a key that is pressed at the Button Controller board is sent to the Terminal Target application and printed in the terminal window. If a data frame is received by the target, it flashes the data LED. All LEDs are flashed once if the Remote Controller Board does not get an acknowledgement from the target node. This can be used to check the coverage of the implementation. Pressing two push buttons simultaneously is not supported.

The SEL & Target-1 or Target-2 keys can be used to switch between different target devices. In order to do so, press the SEL button first and then press the desired function key that was used during the pairing procedure. If a function key is selected that has not been used for a pairing procedure with a terminal target application, all

LEDs will flash to indicate a malfunction. Wait until the LED flashing has stopped before continuing.

The button controller boards support all three command types: PRESSED, REPEATED, and RELEASED (section 2.2.3). Table 3-2 shows the button controller board buttons and their corresponding ZRC command codes.

Table 3-2. KEY_RC board ZRC command codes.

Button	Command code
PWR	PRESSED
VOLUME	REPEATED

If power to the remote controller board is momentarily disconnected, the active function needs to be reselected by pressing the SEL button followed by the function button that was used previously during the pairing procedure.

From the terminal output, menu item C, channel agility, can be used to toggle (enable or disable) the periodic channel agility mechanism at the target node. See section 2.2 for further information about channel agility. The current status (enabled or disabled) of the periodic channel agility mode is printed to the terminal program ("Channel agility (periodic mode) enabled," for example).

Channel agility becomes very useful in noisy channel environments. When the noise level on the current operating channel become too great (for demonstration purposes, the noise threshold level was set to -80dBm) and the adjacent channels yield better noise performance, the channel with the lowest noise energy will be selected as the new base channel. The parameters used for the channel agility mechanism can be configured using the menu item O, channel agility configuration.

If it is desired to demonstrate channel agility when the noise situation would not ordinarily warrant changing the current channel, menu entry B, base channel change, can be used to force a base channel change.

Menu item Y, standby, sets the Terminal Target application's transceiver to power save mode. During power save mode, the receiver is set periodically to sleep and wake up again. The transmit mechanisms of RF4CE allow the target to wake up during the power save mode by sending a command from the remote controller.

Menu item M provides the user to enable the accelerometer at the controller side. It also takes the input for the accelerometer ON duration. The target will receive the accelerometer position from the remote controller every 200ms till the accelerometer ON duration expires.

3.2.2 RF frame capture

Over-the-air RF frames that are exchanged between both nodes during startup, pairing, and remote control operation can be captured and displayed on the screen by using an RF sniffer.

Figure 3-4 shows an example of the RF frames exchanged during startup, discovery, and pairing between the Terminal Target and the Button Controller applications. The security is enabled at both nodes, and the KeyExTransferCount parameter is set to its minimum value of 3.

Figure 3-4. RF sniffer snapshot.

S...	Channel	Source P...	MAC Src	Destination ...	Destination Address	Protocol	Packet Type
1	15			Broadcast (0x..	Broadcast (0xffff)	IEEE 802.15.4	Command: Beacon Request
2	20			Broadcast (0x..	Broadcast (0xffff)	IEEE 802.15.4	Command: Beacon Request
3	25			Broadcast (0x..	Broadcast (0xffff)	IEEE 802.15.4	Command: Beacon Request
4	15		00:04:25:ff:ff:17:53:a5	Broadcast (0x..	Broadcast (0xffff)	ZigBee RF4CE NWK	RF4CE: Discovery request
5	20		00:04:25:ff:ff:17:53:a5	Broadcast (0x..	Broadcast (0xffff)	ZigBee RF4CE NWK	RF4CE: Discovery request
6	25		00:04:25:ff:ff:17:53:a5	Broadcast (0x..	Broadcast (0xffff)	ZigBee RF4CE NWK	RF4CE: Discovery request
7	15		00:04:25:ff:ff:17:53:a5	Broadcast (0x..	Broadcast (0xffff)	ZigBee RF4CE NWK	RF4CE: Discovery request
8	20		00:04:25:ff:ff:17:53:a5	Broadcast (0x..	Broadcast (0xffff)	ZigBee RF4CE NWK	RF4CE: Discovery request
9	20	0x457d	00:04:25:ff:ff:17:53:0c	Broadcast (0x..	00:04:25:ff:ff:17:53:a5	ZigBee RF4CE NWK	RF4CE: Discovery response
10	20					IEEE 802.15.4	Acknowledgment
11	20	Broadcast..	00:04:25:ff:ff:17:53:a5	0x457d	00:04:25:ff:ff:17:53:0c	ZigBee RF4CE NWK	RF4CE: Pair request
12	20					IEEE 802.15.4	Acknowledgment
13	20	0x457d	00:04:25:ff:ff:17:53:0c	Broadcast (0x..	00:04:25:ff:ff:17:53:a5	ZigBee RF4CE NWK	RF4CE: Pair response
14	20					IEEE 802.15.4	Acknowledgment
15	20	0x457d	00:04:25:ff:ff:17:53:0c	Broadcast (0x..	00:04:25:ff:ff:17:53:a5	ZigBee RF4CE NWK	RF4CE: Key seed
16	20					IEEE 802.15.4	Acknowledgment
17	20	0x457d	00:04:25:ff:ff:17:53:0c	Broadcast (0x..	00:04:25:ff:ff:17:53:a5	ZigBee RF4CE NWK	RF4CE: Key seed
18	20					IEEE 802.15.4	Acknowledgment
19	20	0x457d	00:04:25:ff:ff:17:53:0c	Broadcast (0x..	00:04:25:ff:ff:17:53:a5	ZigBee RF4CE NWK	RF4CE: Key seed
20	20					IEEE 802.15.4	Acknowledgment
21	20	0x457d	00:04:25:ff:ff:17:53:0c	Broadcast (0x..	00:04:25:ff:ff:17:53:a5	ZigBee RF4CE NWK	RF4CE: Key seed
22	20					IEEE 802.15.4	Acknowledgment
23	20	0x457d	00:04:25:ff:ff:17:53:a5	Broadcast (0x..	00:04:25:ff:ff:17:53:0c	ZigBee RF4CE NWK	RF4CE: Ping request
24	20					IEEE 802.15.4	Acknowledgment
25	20	0x457d	00:04:25:ff:ff:17:53:0c	Broadcast (0x..	00:04:25:ff:ff:17:53:a5	ZigBee RF4CE NWK	RF4CE: Ping response
26	20					IEEE 802.15.4	Acknowledgment
27	20	0x457d	0x2578	Broadcast (0x..	00:04:25:ff:ff:17:53:a5	RF4CE-CERC	CERC: Command discovery request
28	20					IEEE 802.15.4	Acknowledgment
29	20		0x571d	0x457d	0x2578	RF4CE-CERC	CERC: Command discovery response
30	20					IEEE 802.15.4	Acknowledgment
31	20		0x571d	0x457d	0x2578	RF4CE-CERC	CERC: Command discovery request
32	20					IEEE 802.15.4	Acknowledgment
33	20	0x457d	0x2578	Broadcast (0x..	00:04:25:ff:ff:17:53:a5	RF4CE-CERC	CERC: Command discovery response
34	20					IEEE 802.15.4	Acknowledgment
35	20		0x571d	0x457d	0x2578	RF4CE-CERC	CERC: User control pressed
36	20					IEEE 802.15.4	Acknowledgment

Target device: 0x00 04 25 FF FF 17 53 0C

Controller device: 0x00 04 25 FF FF 17 53 A5

3.3 Single Button Controller example application

To understand how to use the RF4Control API (see section 2.2 and section 2.5) in a user-defined application, a simple Single Button Controller is introduced. It is simpler than the Key Remote Controller application. It makes use of only a single button, and can be operated as one module. It needs an adapter board only for programming. The following description uses the Atmel ATmega128RFA1 RCB, called RCB_6_3_PLAIN [18]. Besides the RCB_6_3_PLAIN board, the Atmel ATmega128RFA1-EK1 [17] board, ATMEGA256RFR2_XPLAINED_PRO board used to run this application.

3.3.1 Hardware

Figure 3-5. ATmega128RFA1 – RCB_6_3_PLAIN.

The Atmel ATmega128RFA1 RCB_6_3_PLAIN board contains three general purpose LEDs (D1, D2 and D3) and one push button for application control. Status LED D5 to the right of the button displays the Atmel ATmega128RFA1 reset state. LEDs and button are shown on the bottom side of Figure 3-5. For correct operation, the antenna needs to be connected to the RCB's SMA connector, and two batteries (AAA) need to be inserted into the RCB's battery holder. For further information about the RCB, see [18].

3.3.2 Firmware programming

The AVR2102 package contains pre-compiled binaries providing an out-of-the-box experience. The MAC User Guide ([7], section 7.3) provides further information about firmware programming using Atmel Studio. Table 3-1 contains information about the recommended MCU fuse settings.

For debugging and programming purposes, a JTAG [10] is required. The JTAG is connected to the RCB via a Breakout Board (BB), Sensor Terminal board

3.3.3 Application handling

Once the firmware is downloaded to the ATmega128RFA1 device and the JTAG pod and BB are disconnected, the application can be started. The RCB communication peer is the Terminal Target application (see section 0).

3.3.3.1 Cold start

The cold-start reset and push button pairing procedure is initiated by pushing the button on the controller and entering 'A' on the HyperTerminal menu on the Terminal Target. Either device can start the push button pairing procedure.

In order to pair the Single Button Controller with the Terminal Target, the push button pairing procedure is used. At the Terminal Target application, the push button pairing procedure is started by entering 'A' at the HyperTerminal menu on the Terminal Target. The device is reset and started. Then the Terminal Target application displays the ready message to the terminal window: "Press the push button pairing button at the remote controller now."

To start the push button pairing procedure, the RCB push button needs to be pressed as the board is switched on. The board LEDs show the current status of the pairing procedure:

LED 0 (D2): application reset and initialization; or error indication

LED 1 (D3): push button pairing (discovery and pairing); or error indication

LED 2 (D4): error indication

If the push button pairing procedure has been completed successfully, all three LEDs are switched on for about one second. The Single Button Controller has limited error handling capability. Blinking LEDs indicate that an error has occurred during discovery or pairing.

After successful pairing, the Atmel ATmega128RFA1 device is set to sleep. Pressing the push button wakes the MCU and sends an RF4CE frame (POWER_TOGGLE_FUNCTION command) to the paired device, that is, to the Terminal Target application. The Terminal Target application toggles its LED 1 and the relay 1 if the POWER_TOGGLE_FUNCTION command is received. If the Terminal Target application does not send an acknowledgement to the Single Button Controller, all three controller LEDs are switched on for about two seconds.

3.3.3.2 Warm start - Reinstating existing pairing table

The pairing information is stored to the non-volatile memory (NVM) of the ATmega128RFA1. The RCB can be switched off using the power switch (see left side in Figure 3-5). If the push button is not pressed during power up of the RCB, a warm start is performed. During the warm start, the pairing information is read from the NVM as the Single Button Controller is powered up again. The pairing table used in the last session is reinstated on power-up. All three LEDs are switched on at the same time and switched off in sequence, indicating that the warm start reset has been completed.

3.3.4 Development environment

Two different development environments are supported by the included project or Makefile files:

- IAR Embedded Workbench® for AVR;
  http://www.iar.com
- Atmel Studio

  http://www.atmel.com/

3.3.5 Application implementation

Using the library release package, the entire implementation of the Single Button Controller application requires only a few files:

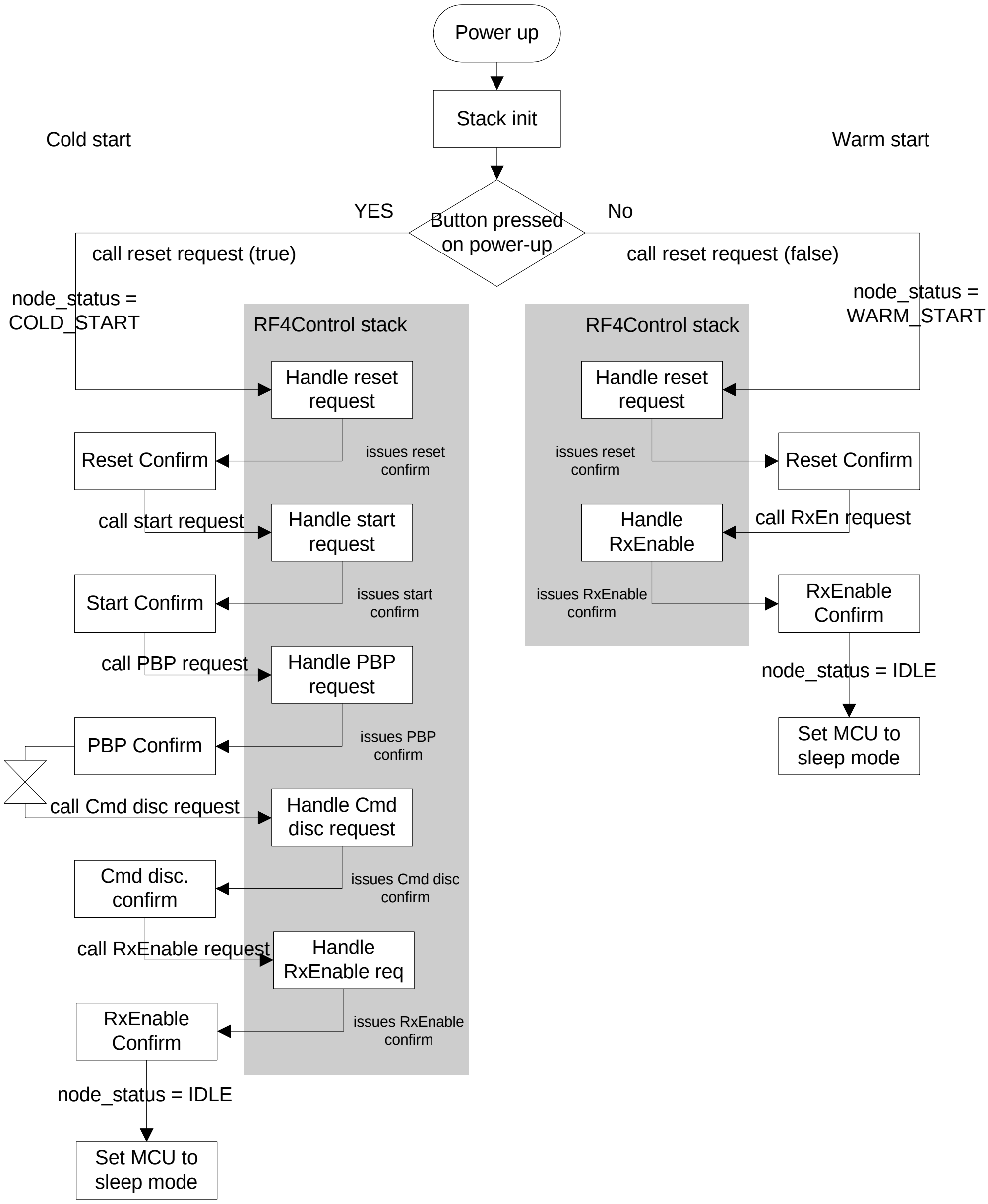
- Project file/Makefile:
  For IAR: Single Button Controller Project and Single Button Controller workspace
  For Atmel Studio: Single Button Controller solution & project files and Makefile
- RF4Control library:
  For IAR:
  \thirdparty\wireless\avr2102_rf4control\lib\zrc\ctr\<mcu_name>\iar\librf4ce-zrc-controller.r82

For Atmel Studio /GCC:
\thirdparty\wireless\avr2102_rf4control\lib\zrc\ctr\<mcu_name>\iar\librf4ce-zrc-controller.a

3.3.5.1 Program flowchart

The program flow of the Single Button Controller application is shown by Figure 3-6.

Figure 3-6. Single Button Controller program flowchart.



The following paragraphs describe the source code implementation of the Single Button Controller example application. It is recommended having the HTML-based API documentation handy while walking through the implementation (see section 2.5).

3.3.5.2 Vendor-specific data exchange

For vendor-specific data exchange, the RF4Control stack provides vendor-specific data handling API functions (see section 2.4). Every application can define the semantics of the vendor-specific data. The Single Button Controller application uses vendor-specific data exchange to implement the following features:

- Battery status request/response
- Alive request/response
- Firmware version request/response
- RxEnable request/response
- Firmware request/response for firmware over-the-air (FOTA) update
- Firmware swap

For this application, the request messages are sent by the target node (Terminal Target or ZRC Target application), and the controller node (Single Button Controller) answers the request with a response message.

Example: The user initiates a battery status request by entering option D on the Terminal Target menu. The target node sends the battery status request frame using multi-channel transmission to the controller node. The controller, operating in power save mode, switches its receiver on every second for a short duration. During this window, the controller receives this request frame. The controller stack analyzes the frame and calls the vendor data indication callback function vendor_data_ind(). Within the vendor_data_ind() function, the payload is parsed and a battery status request is identified. The controller measures its voltage level and replies with the battery response message frame.

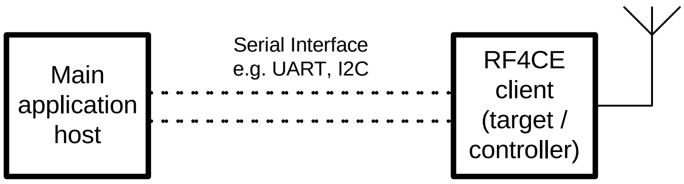
3.4 Serial Interface example application

3.4.1 Introduction

The Atmel RF4Control stack provides a Serial Interface example application that can be used for any inter-processor communication between a host controller running the main application and a client controller handling the RF4CE communication over the air. Both controllers use a serial interface to communicate. The host controller can be implemented as a standalone microcontroller, or it can also be a personal computer. Figure 3-7 shows a communication scenario example.

The client receives commands from the host, such as data transfer requests. The client indicates received data over the air from its communication peer by returning data indications to its host.

Figure 3-7. Communication scenario example.



The physical interface between the main application controller and the RF4CE client can be manifold, including:

- UART

- USB
- I^2C (TWI)
- Proprietary interface

The Serial Interface example application uses a UART (RS-232) or USB for the serial interface. The physical interface handling is implemented by the PAL; see AVR2025 MAC Software Package [7] for further information about the PAL.

The logical interface is handled by the example application within the file “serial_interface.c”. This file implements the logical protocol used for the communication between the main application host and the RF4CE client application. The same protocol scheme is used for host-to-client and client-to-host communications.

3.4.2 Message structure

The message structure of the logical protocol is described by the following table.

Table 3-3. Logical protocol message structure (in bytes).

	Message header		**Message payload**					**Message trailer**
SOT	PROTOCOL ID	Length of payload	Message code	data, byte 0	data, byte 1	…	data, byte LEN - 1	EOT
0x01	0x02	LEN	0x…	0x…	0x…		0x…	0x04

The message consists of the message header, message payload, and message trailer.

The start-of-text symbol (SOT) and the length of payload field form the message header. The value of the length field indicates how many bytes are contained in the actual message payload; that is, the number of message payload bytes before the message trailer end-of-text (EOT) symbol is expected. The message payload is appended after the message header. The message payload starts with the message code followed by the message data fields. The length and the code of each message are listed in Table 3-4.

The order of the payload bytes is aligned to the RF4CE primitive specification [12]. If more than one parameter is used by the primitive, the parameters are concatenated to the end of a byte stream in the message payload. Parameters whose size is longer than 8 bits in length are sent with the least-significant byte first. Parameters with 24-bit lengths are encoded as 32-bit values where the most-significant byte contains a dummy value and is ignored by the serial interface. Parameter lists such as DevTypeList and ProfileIdList, which have a variable length based on the primitive specification, consist of a fixed length when using the serial interface protocol. The serial interface protocol sets the maximum length for each list; that is, the size of DevTypeList is set to 3 and size of ProfileIdList is set to 7. List values of unused entries are ignored, but need to be present.

The following examples introduce this concept.

3.4.2.1 Message structure example 1: NLME-RESET.request primitive

If the main application host wants to reset the network layer of the RF4CE application, it sends the NLME-RESET.request command to the RF4CE client. See [12] for further information about the NLME-RESET.request primitive. This request command requires the SetDefaultNIB parameter. Following this example, the value of the SetDefaultNIB value is set to true (1). Using the Serial Interface application, the

NLME-RESET.request is encoded and sent as a byte stream via the serial link as follows:

Listing 3-1. NLME-RESET.request command byte stream.

Byte stream from application host to RF4CE client via serial interface:

```
0x01 0x02 0x02 0x2A 0x01 0x04
```

Data interpretation:

```
0x01: SOT
0x02: RF4CE Protocol ID
0x02: Length field value
0x2A: Message code for NLME-RESET.request
0x01: Message parameter SetDefaultNIB; here 0x01 = true
0x04: EOT
```

The RF4CE client answers a NLME-RESET.request with a NLME-RESET.confirm primitive. Using the Serial Interface application, the NLME-RESET.confirm message is encoded and sent as a byte stream via the serial link as follows:

Listing 3-2. NLME-RESET.confirm command byte stream.

Byte stream from RF4CE client to application host via serial interface:

```
0x01 0x02 0x02 0x3D 0x00 0x04
```

Data interpretation

```
0x01: SOT
0x02: RF4CE Protocol ID
0x02: Length field value
0x3D: Message code for NLME-RESET.confirm
0x00: Message parameter status; here 0x00 = SUCCESS
0x04: EOT
```

3.4.2.2 Message structure example 2: NLME-AUTO-DISCOVERY.confirm primitive

The RF4CE client application generates a NLME-AUTO-DISCOVERY.confirm primitive as the result of the NLME-AUTO-DISCOVERY.request. Listing 3-3 shows the NLME-AUTO-DISCOVERY.confirm primitive message that is forwarded from the RF4CE client application to the main application host.

Listing 3-3. NLME-AUTO-DISCOVERY.confirm message.

Byte stream from RF4CE client to application host via serial interface:

```
0x01 0x02 0x0A 0x36 0x00 0x08 0x07 0x06 0x05 0x04 0x03 0x02 0x01
0x04
```

Data interpretation:

```
0x01: SOT
0x02: RF4CE Protocol ID0x0A: Length field value
0x36: Message code for NLME-AUTO-DISCOVERY.confirm
0x00: Message parameter "Status"; here 0x00 = SUCCESS
0x08 … 0x01: Message parameter "SrcIEEEAddr"; here
0x0102030405060708
0x04: EOT
```

3.4.2.3 Message structure exception

As described, the message data payload is aligned to the RF4CE primitive order and size, in general. There are two exceptions to this rule, however: (1) the NLDE-DATA.request and (2) the NLDE-DATA.indication primitive messages. The parameter order for these primitives is changed in comparison to the RF4CE specification.

Listed below are the primitives with their own parameter order for the Serial Interface application example:

- NLDE-DATA.request parameter order:
  PairingRef, ProfileId, VendorId, TxOptions, nsduLength, nsdu
- NLDE-DATA.indication:
  PairingRef, ProfileId, vendorId, RxLinkQuality, RxFlags, nsduLength, nsdu

3.4.3 Message codes

Table 3-4 lists the message codes and message lengths supported by the Serial Interface protocol.

Table 3-4. Message codes and message lengths (bytes).

RF4CE Network Primitive	Message code	Message length
NLDE-DATA.request	0x24	≥7 + data len
NLDE-DATA.indication	0x34	≥8 + data len
NLDE-DATA.confirm	0x35	3
NLME-AUTO-DISCOVERY.request	0x25	15
NLME-AUTO-DISCOVERY.confirm	0x36	10
NLME-COMM-STATUS.indication	0x37	14
NLME-DISCOVERY.request	0x26	29
NLME-DISCOVERY.indication	0x38	48
NLME-DISCOVERY.response	0x27	22
NLME-DISCOVERY.confirm	0x39	4 + n * 49 n ≥1
NLME-GET.request	0x2B	3
NLME-GET.confirm	0x3A	≥5
NLME-PAIR.request	0x28	24
NLME-PAIR.indication	0x3B	50
NLME-PAIR.response	0x29	24

RF4CE Network Primitive	Message code	Message length
NLME-PAIR.confirm	0x3C	38
NLME-RESET.request	0x2A	2
NLME-RESET.confirm	0x3D	2
NLME-RX-ENABLE.request	0x2C	5
NLME-RX-ENABLE.confirm	0x3E	2
NLME-SET.request	0x2D	≥4
NLME-SET.confirm	0x3F	4
NLME-START.request	0x2E	1
NLME-START.confirm	0x40	2
NLME-UNPAIR.request	0x2F	2
NLME-UNPAIR.indication	0x41	2
NLME-UNPAIR.response	0x30	2
NLME-UNPAIR.confirm	0x42	3
NLME-UPDATE-KEY.request	0x31	18
NLME-UPDATE-KEY.confirm	0x43	3
NWK_CH_AGILITY_REQUEST	0x32	2
NWK_CH_AGILITY_INDICATION	0x44	2
NWK_CH_AGILITY_CONFIRM	0x45	4

For better readability, the Atmel RF4Control stack uses the header file nwk_msg_code.h to assign symbolic names to the message codes. For functional compatibility, enumeration and assigned numbers should not be changed in this header file.

3.4.4 Serial Interface - message structure

The message structure of all the supported network primitives is listed out below..

3.4.4.1 NLDE-DATA.request

	Message header		Message payload							Message trailer
SOT	Protocol ID	Length of payload	Msg code	Pair. ref	Profile id	Vendor Id	Tx options	Nsdu length	nsdu	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	2 bytes	1 byte	1byte	LEN * 1 byte	1 byte
0x01	0x02	1+LEN	0x24	0x…	0x…	Byte 0-1	0x..	LEN		0x04

3.4.4.2 NLDE-DATA.indication

	Message header	Message payload	Message trailer

	Message header		Message payload	Message trailer
SOT	Protocol ID	Length of payload		… EOT
1 byte	1 byte	1 byte		1 byte
0x01	0x02	8+LEN	…	0x04

Message payload										
Msg code	Pair. ref	Profile ID	Vendor ID		Rx Link Quality	Rx Flags	nsdu length	Data byte0	..	Data Byte LEN-1
1 byte	1 byte	1 byte	2 bytes		1 byte	1 byte	1 byte	1 byte		1 byte
0x34	0x..	0x..	byte0	byte1	0x..	0x..	LEN	0x..		0x..

3.4.4.3 NLDE-Data.confirm

	Message header		Message payload				Message trailer
SOT	Protocol ID	Length of payload	Message code	status	Pair. Ref.	Profile id	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	4	0x35	0x…	0x…	0x..	0x04

3.4.4.4 NLME_AUTO_DISCOVERY.Request

	Message header		Message payload					Message trailer
SOT	Protocol ID	Length of payload	Message code	RecApp Capabilities	RecDev Type List	Rec ProfileIdList	Auto DiscDuration	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	DevTypeList Size 3* 1 byte	Profile list Size 7 * 1 byte	4 bytes	1 byte

0x01	0x02	16	0x25	0x…	0x…	0x..	Byte 0	Byte 1	Byte 2	Byte 3	0x04

3.4.4.5 NLME-AUTO-DISCOVERY.confirm

	Message header		Message payload										Message trailer
SOT	Protocol ID	Length of payload	Message code	status	Src IEEE addr								EOT
1 byte	1 byte	1 byte	1 byte	1 byte	8 bytes								1 byte
0x01	0x02	10	0x36	0x…	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	0x04

3.4.4.6 NLME-COMM-STATUS.indication

	Message header		Message payload							Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	Pair. ref	Dst PAN ID		Dst Addr Mode	Dst addr	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	2 bytes		1 byte	8 bytes	1 byte
0x01	0x02	14	0x37	0x…	0x…	byte 0	Byte 1	0x..	Byte 0- 8	0x04

3.4.4.7 NLME-DISCOVERY.request

	Message header		Message pay load	Message trailer
SOT	**Protocol ID**	**Length of payload**		**EOT**
1 byte	1 byte	1 byte		1 byte
0x01	0x02	29	Shown below	0x04

Message payload									
Msg code	PanID	Nwk addr	Org App Cap.	Devtype list	Org Profile ID list	Search devtype	disc Profile list size	disc Profile ID list	Disc duration
1 byte	2 bytes	2 bytes	1 byte	Devtype Size 3 * 1 byte	Profile id List size 7* 1 byte	1 byte	1 byte	Profile id List size 7* 1 byte	4 bytes
0x26	Byte 0-1	Byte 0-1	0x..	0x..		0x..	0x..		Byte0-3

3.4.4.8 NLME-DISCOVERY.indication

	Message header		Message pay load	Message trailer
SOT	Protocol ID	Length of payload		**EOT**
1 byte	1 byte	1 byte		1 byte
0x01	0x02	48	Shown below	0x04

Message pay load											
Msg code	status	Src IEEE addr	Org Node Cap.	Org Vendor ID	Org Vendor string	Org App Cap.	Org User string	Org Devtype List	Org Profile ID list	Search dev type	Rx Link quality
1 byte	1 byte	8 bytes	1 byte	2 bytes	7 bytes	1 byte	15 bytes	Devtype Size 3 * 1 byte	Profile id List size 7* 1 byte	1 byte	1 byte
0x38	0x..	Byte 0-7	0x..	Byte 0-1	Byte 0-6		Byte 0-14	0x..		0x..	0x..

3.4.4.9 NLME-DISCOVERY.response

	Message header		Message pay load							Message trailer
SOT	Protocol ID	Length of payload	Message code	status	Dst IEEE Addr	Rec App cap	rec Devtype List	rec Profile ID list	Disc ReqLQI	EOT

1 byte	1 byte	1 byte	1 byte	1 byte	8 bytes	1 byte	Devtype Size 3 * 1 byte	Profile id List size 7* 1 byte	1 byte	1 byte
0x01	0x02	22	0x27	0x…	Byte 0-7	0x..			0x..	0x04

3.4.4.10 NLME-DISCOVERY.confirm

	Message header		**Message pay load**	**Message trailer**
SOT	Protocol ID	Length of payload		**EOT**
1 byte	1 byte	1 byte		1 byte
0x01	0x02	4 + 49(node_des_size)* num_of_nodes	Shown below	0x04

Message pay load				
Msg code	status	Num nodes	Desc. List size	Desc. List.
1 byte	1 byte	1 byte	1 byte	49(node_des_size)* num_of_nodes
0x39	0x..	0x..	49(node_des_size)* num_of_nodes	

3.4.4.11 NLME-GET.request

	Message header		**Message pay load**			**Message trailer**
SOT	Protocol ID	Length of payload	Msg code	Nib attribute	Attribute index	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	3	0x2B	0x…	0x…	0x04

3.4.4.12 NLME-GET.confirm

									Message trailer
	Message header		Message pay load						
SOT	Protocol ID	Length of payload	Msg code	status	Nib attribute	Attribute index	Attribute len	Attribute value	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	Attribute Len * 1 byte	1 byte
0x01	0x02	5+ attribute len	0x3A	0x..	0x…	0x…	0x..		0x04

3.4.4.13 NLME-PAIR.request

	Message header		Message pay load	Message trailer
SOT	Protocol ID	Length of payload		**EOT**
1 byte	1 byte	1 byte		1 byte
0x01	0x02	24	Shown below	0x04

Message payload								
Msg code	Logical channel	Dst panID	Dst IEEEaddr	Org App Cap.	Org Devtype list	Org Profile ID list	Key ExTransfer count	EOT
1 byte	1 byte	2 bytes	8 bytes	1 byte	DEV TYPE LIST SIZE 3 * 1 byte	PROFILE ID LIST SIZE 7 * 1 byte	1 byte	1 byte
0x28	0x..	Byte0-1	Byte0-7	0x..			0x…	0x04

3.4.4.14 NLME-PAIR.indication

	Message header		Message pay load	Message trailer
SOT	Protocol ID	Length of payload		**EOT**
1 byte	1 byte	1 byte		1 byte
0x01	0x02	50	Shown below	0x04

Message pay load												
Msg code	status	src pan ID	src IEEE addr	Org node Cap.	Org Vendor id	Org Vend. String	Org App Cap.	Org user String	Org Devtype list	Org Profile ID list	Key Ex Trans. cnt	Prov Pair. Ref.
1 byte	1 byte	2 bytes	8 bytes	1 byte	2 bytes	7 bytes	1 byte	15 bytes	DEV TYPE LIST SIZE 3 * 1 byte	PROFILE ID LIST SIZE 7 * 1 byte	1 byte	1 byte
0x3B	0x..	Byte 0-1	Byte 0-7	0x..	Byte 0-1	Byte 0-6	0x..	Byte 0-14			0x …	0x..

3.4.4.15 NLME-PAIR.response

	Message header		Message pay load									Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	Dst panID	Dst IEEE addr	Rec App Cap.	Rec Devtype list	Rec Profile ID list	Prov. Pair. ref	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	2 bytes	8 bytes	1 byte	DEV TYPE LIST SIZE 3 * 1 byte	PROFILE ID LIST SIZE 7 * 1 byte	1 byte	1 byte
0x01	0x02	24	0x3A	0x..	Byte0-1	Byte0-7	0x..			0x..	0x04

3.4.4.16 NLME-PAIR.confirm

	Message header	Message pay load	Message trailer

SOT	Protocol ID	Length of payload	Msg code	status	Pair. Ref.	Rec. Vendor ID	Rec. Vendor string	Rec App Cap.	Rec. user string	Rec Devtype list	Rec Profile ID list	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	2 bytes	7 bytes	1 byte	15 bytes	DEV TYPE LIST SIZE 3 * 1 byte	PROFILE ID LIST SIZE 7 * 1 byte	1 byte
0x01	0x02	38	0x3C	0x..	0x..	Byte0-1	Byte0-6	0x..	Byte 0-14			0x04

3.4.4.17 NLME-RESET.request

	Message header		Message pay load		Message trailer
SOT	Protocol ID	Length of payload	Msg code	setDefaultNIB	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	2	0x2A	0x..	0x04

3.4.4.18 NLME-RESET.confirm

	Message header		Message pay load		Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte

0x01	0x02	2	0x3D	0x..	0x04

3.4.4.19 NLME-RX-ENABLE.request

	Message header		**Message pay load**		**Message trailer**
SOT	Protocol ID	Length of payload	Msg code	rxonDuration	EOT
1 byte	1 byte	1 byte	1 byte	4 bytes	1 byte
0x01	0x02	2	0x2C	Byte0-3	0x04

3.4.4.20 NLME-RX-ENABLE.confirm

	Message header		**Message pay load**		**Message trailer**
SOT	Protocol ID	Length of payload	Msg code	status	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	2	0x3E	0x..	0x04

3.4.4.21 NLME-SET.request

	Message header		**Message pay load**				**Message trailer**
SOT	Protocol ID	Length of payload	Msg code	NIB attribute	NIB Attribute index	NIB Attribute value	EOT

1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	LEN	1 byte
0x01	0x02	3 + LEN	0x3E	0x..	0x..		0x04

3.4.4.22 NLME-SET.confirm

	Message header		Message pay load				Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	NIB attribute	NIB Attribute index	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	4	0x3F	0x..	0x..	0x..	0x04

3.4.4.23 NLME-START.request

	Message header		Message pay load	Message trailer
SOT	Protocol ID	Length of payload	Msg code	EOT
1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	1	0x2E	0x04

3.4.4.24 NLME-START.confirm

	Message header	Message pay load	Message trailer

SOT	Protocol ID	Length of payload	Msg code	status	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	2	0x40	0x..	0x04

3.4.4.25 NLME-UNPAIR.request

	Message header		**Message pay load**		**Message trailer**
SOT	Protocol ID	Length of payload	Msg code	Pair. Ref.	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	2	0x2F	0x..	0x04

3.4.4.26 NLME-UNPAIR.confirm

	Message header		**Message pay load**			**Message trailer**
SOT	Protocol ID	Length of payload	Msg code	status	Pair. Ref/	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	3	0x42	0x..	0x..	0x04

3.4.4.27 NLME-UPDATE-KEY.request

	Message header	**Message pay load**	**Message trailer**

SOT	Protocol ID	Length of payload	Msg code	Pair. ref	New Link key	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	16 bytes	1 byte
0x01	0x02	18	0x31	0x..	Byte 0-15	0x04

3.4.4.28 NLME-UPDATE-KEY.confirm

	Message header		Message pay load			Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	Pair. Ref/	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	3	0x43	0x..	0x..	0x04

3.4.4.29 NWK_CH_AGILITY_REQUEST

	Message header		Message pay load		Message trailer
SOT	Protocol ID	Length of payload	Msg code	Agility mode	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	2	0x32	0x..	0x04

3.4.4.30 NWK_CH_AGILITY_CONFIRM

	Message header		Message pay load				Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	Channel changed	Logical Channel	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	4	0x45	0x..	0x..	0x..	0x04

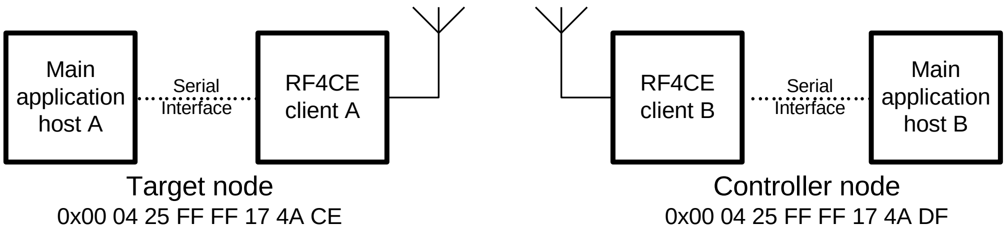
3.4.5 Protocol adaption

The message structure described here is an example implemented by the Serial Interface application. The protocol or message structure can easily be adapted to the end-user's application needs. For example, a checksum, such as CRC, can be added to detect and correct errors that might occur over the serial link.

3.4.6 Serial interface usage

As introduced in section 3.4.1, the Serial Interface application can be used in a scenario where the Atmel RF4CE stack in hosted on one microcontroller and the main application processor controls it via a serial interface. The following section explains how to use the Serial Interface application to set up a communication link. The following figure shows such a setup.

Figure 3-8. Application setup using serial interface.



The main application microcontroller A hosts an application, such as a TV, controlling the RF4CE client A. The other main application microcontroller B hosts an application, such as a remote controller, controlling the RF4CE client B. The main application microcontrollers use a serial interface to communicate with their RF4CE clients.

The application hosts send commands to their RF4CE clients to configure the RF4CE communication. The charts below show a typical scenario of commands that establish an RF4CE link and send a data frame to the target.

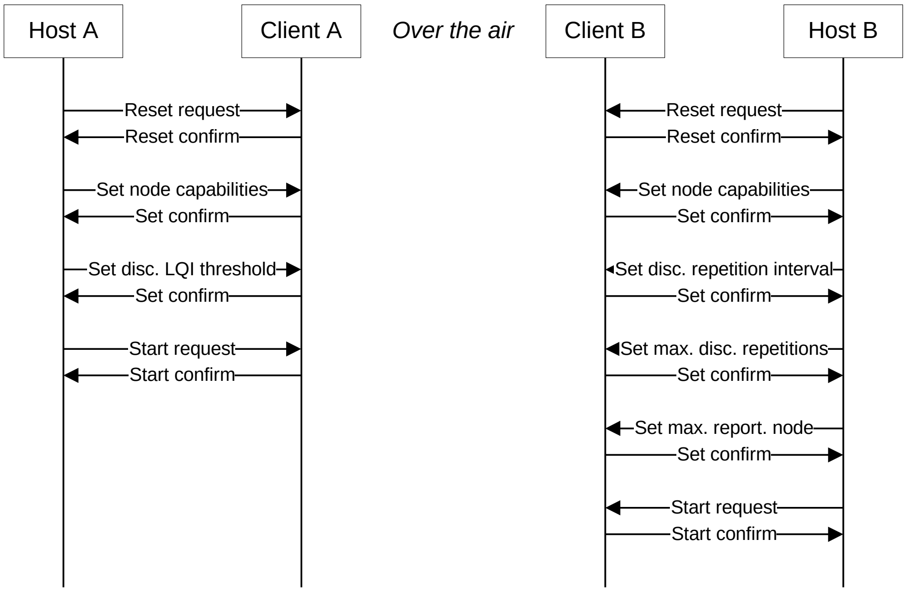
The data communication between the host and the client serial interface is described in section 3.4.2.

A typical RF4CE network application can be realized using the following hypothetical operating scenario:

- Step1: Node initialization: Each client is configured (reset, set capabilities, set LQI threshold, etc.)

- Step 2: Discovery and pairing. Each client is directed to start discovery and pairing procedures
- Step 3: Data transmission. Each client is controlled to transmit and receive RF4CE application data

3.4.6.1 Step 1 – Initialization

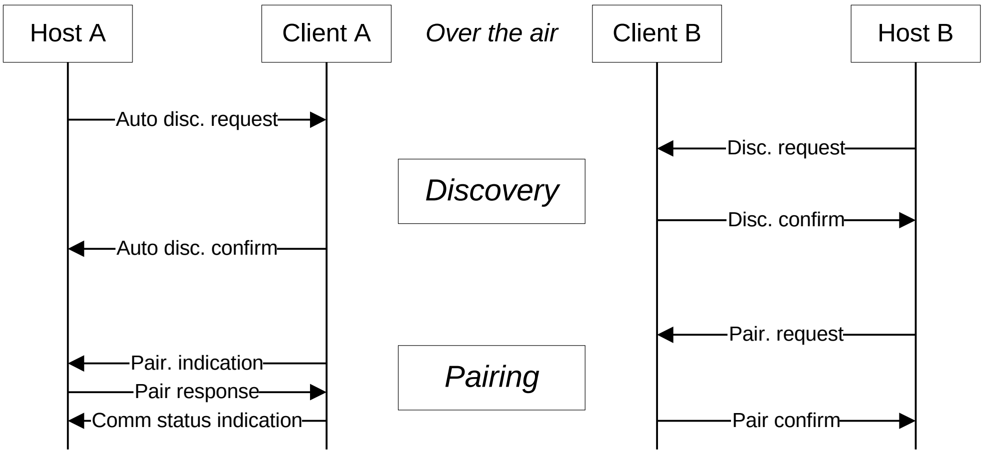


Command / Message	Description	Byte stream over serial interface (message payload)
Reset request	Resets the RF4CE stack and underlying layers	0x2a 0x01
Reset confirm	Returns the results of the reset request	0x3d 0x00
Set node capabilities	Sets the capabilities of the RF4CE client, such as node type (target or controller) and security support	Target node: 0x2d 0x73 0x00 0x01 0x0f Controller node: 0x2d 0x73 0x00 0x01 0x0c
Set confirm / node capabilities	Returns the result of the previous set confirm	0x3f 0x00 0x73 0x00
Set disc. LQI threshold	Sets the LQI threshold for the incoming discovery requests; here: 0x01	0x2d 0x62 0x00 0x01 0x01
Set confirm / disc. LQI threshold	Returns the result of the previous set confirm	0x3f 0x00 0x62 0x00
Set disc. repetition interval	Sets the duration of the discovery repetition interval; here: 0x00044AA2 symbols or 4.5 second	0x2d 0x63 0x00 0x04 0xa2 0x4a 0x04 0x00

Command / Message	Description	Byte stream over serial interface (message payload)
Set confirm / disc. repetition interval	Returns the result of the previous set confirm	0x3f 0x00 0x63 0x00
Set max. disc. repetitions	Sets maximum number of discovery repetitions; here: 0x1E	0x2d 0x69 0x00 0x01 0x1e
Set confirm / max. disc. repetitions	Returns the result of the previous set confirm	0x3f 0x00 0x69 0x00
Set max report nodes	Sets the maximum number of node descriptors that should be reported during discovery	0x2d 0x6c 0x00 0x01 0x01
Set confirm / max report nodes	Returns the result of the previous set confirm	0x3f 0x00 0x6c 0x00
Start request	Starts the RF4CE client	0x2e
Start confirm	Returns the result of the start confirm	0x40 0x00

There is no specific order required for the commands during configuration, but the start request command should not be issued before setting the node capabilities.

3.4.6.2 Step 2 – Discovery and pairing

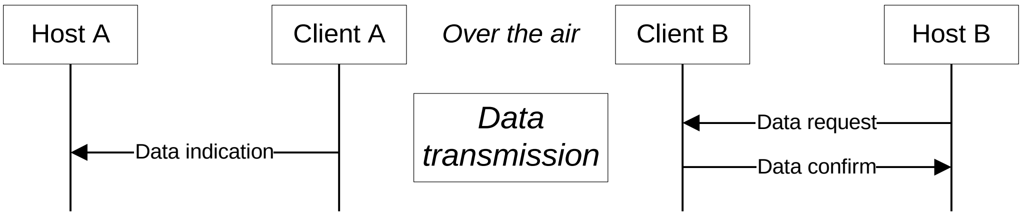


Command / Message	Description	Byte stream over serial interface (message payload)
Auto disc. request	Starts the auto discovery procedure	0x25 0x12 0x02 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x38 0x9c 0x1c 0x00
Auto disc. confirm	Returns the result of the auto discovery procedure	0x36 0x00 0xdf 0x4a 0x17 0xff 0xff 0x25 0x04 0x00

Command / Message	Description	Byte stream over serial interface (message payload)
Disc. request	Starts the discovery procedure	0x26 0xff 0xff 0xff 0xff 0x12 0x01 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x12 0x7a 0x00 0x00
Disc. confirm	Returns the result of the discovery procedure	0x39 0x00 0x01 0x31 0x00 0x0f 0x20 0x1e 0xce 0x4a 0x17 0xff 0xff 0x25 0x04 0x00 0x0f 0x34 0x12 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x12 0x4b 0x49 0x02 0x18 0x39 0x01 0x21 0x58 0x11 0x5b 0x9f 0xef 0x22 0x91 0x40 0x02 0xa2 0xbd 0x01 0x8c 0xc3 0xe4 0xf6 0xe7 0xe5 0x94
Pair request	Starts the pair procedure; parameters are used from the discovery result	0x28 0x0f 0x20 0x1e 0xce 0x4a 0x17 0xff 0xff 0x25 0x04 0x00 0x12 0x01 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x03
Pair indication	Indicates a pairing request	0x3b 0x00 0xff 0xff 0xdf 0x4a 0x17 0xff 0xff 0x25 0x04 0x00 0x0c 0x34 0x12 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x12 0x94 0x78 0x60 0xc4 0x35 0xf2 0x16 0x16 0x2a 0x05 0x00 0x00 0x00 0x03 0xfe 0x01 0x0c 0x34 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x03 0x00
Pair response	Responses to the pairing request, such as allowing to pair	0x29 0x00 0xff 0xff 0xdf 0x4a 0x17 0xff 0xff 0x25 0x04 0x00 0x12 0x02 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Pair confirm	Returns the result of the pair request	0x3c 0x00 0x00 0x34 0x12 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x12 0x09 0xe9 0xce 0x29 0x1e 0xc9 0xc5 0xf3 0x07 0x47 0x08 0x79 0x72 0x87 0x6f 0x02 0x63 0x9a 0x01 0xbb 0xcb 0x0f 0xf4 0x10 0x9d
Comm status indication	Returns the result of the last response, here: pair response	0x37 0x00 0x00 0xff 0xff 0x01 0xdf 0x4a 0x17 0xff 0xff 0x25 0x04 0x00

There is no need to synchronize the auto discovery request on the target node and the discovery request on the controller node because the discovery request commands are sent by the RF4CE client several times, depending of the discovery repetition interval and the maximum discovery repetitions. The target node (Client A) is ready for the discovery request from the controller node and it is based on the duration parameter used for the auto discovery request command.

3.4.6.3 Step 3 – Data transmission



Command / Message	Description	Byte stream over serial interface (message payload)
Data request	Requests to send a data frame	0x24 0x00 0x01 0xf1 0xff 0x0c 0x02 0x12 0x34
Data indication	Indicates the reception of a data frame	0x34 0x00 0x01 0x1e 0x16 0x94 0x02 0x02 0x12 0x34
Data confirm	Returns the result of the data request	0x35 0x00 0x00

3.5 ZigBee Remote Control serial interface

The RF4Control stack provides a ZRC Serial Interface application that is similar to Serial Interface application. The major differences are that the ZRC Serial Interface application supports:

- push button pairing API instead of the normal discovery and pairing mechanism
- remote control command discovery
- RC command handling instead of normal data transfer
- vendor-specific commands
- a controller and target configuration instead of a generic platform configuration

3.5.1 ZRC Serial Interface message codes

The underlying architecture and message structure of the ZRC Serial Interface application remain the same as those of the Serial Interface application described in section 3.4.

Table 3-5 lists the message codes and message lengths supported by ZRC Serial Interface protocol.

Table 3-5. Message codes and message lengths for the ZRC API.

ZRC API functions	Message codes	Message lengths
pbp_org_pair_request	0x46	21
pbp_rec_pair_request	0x48	12
pbp_pair_org_confirm	0x47	3
pbp_pair_rec_confirm	0x49	3
zrc_cmd_disc_request	0x4D	2
zrc_cmd_disc_indication	0x4E	2
zrc_cmd_disc_confirm	0x4F	34
zrc_cmd_disc_response	0x50	35

ZRC API functions	Message codes	Message lengths
zrc_cmd_request	0x4A	6 + payload_length
zrc_cmd_indication	0x4B	5 + payload_length
zrc_cmd_confirm	0x4C	4
vendor_data_request	0x51	7 + Payload_length
vendor_data_indication	0x52	8 + Payload_length
vendor_data_confirm	0x53	3
Unsupported cmd code	0xFF	1

3.5.2 ZRC serial interface message structure

The message structure of all the supported ZRC primitives is listed out below.

3.5.2.1 pbp_org_pair_request

	Message header		Message pay load	Message trailer
SOT	Protocol ID	Length of payload		EOT
1 byte	1 byte	1 byte		1 byte
0x01	0x02	21	Shown below	0x04

Message pay load						
Msg code	Org App Cap.	Org Devtype list	Org Profile ID list	Search Dev type	Disc Profile id List size	Disc Profile id list
1 byte	1 byte	DEV TYPE LIST SIZE 3 * 1 byte	PROFILE ID LIST SIZE 7 * 1 byte	1 byte	1 byte	PROFILE ID LIST SIZE 7 * 1 byte
0x46	0x..			0x..		

3.5.2.2 pbp_rec_pair_request

	Message header	Message pay load	Message trailer

SOT	Protocol ID	Length of payload	Msg code	rec App Cap.	rec Devtype list	rec Profile ID list	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	DEV TYPE LIST SIZE 3 * 1 byte	PROFILE ID LIST SIZE 7 * 1 byte	1 byte
0x01	0x02	12	0x48	0x..			0x04

3.5.2.3 pbp_pair_org_confirm

	Message header		Message pay load			Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	Pair. Ref/	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	3	0x47	0x..	0x..	0x04

3.5.2.4 pbp_pair_rec_confirm

	Message header		Message pay load			Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	Pair. Ref/	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	3	0x49	0x..	0x..	0x04

3.5.2.5 zrc_cmd_disc_request

		Message header	Message pay load		Message trailer
SOT	Protocol ID	Length of payload	Msg code	Pair. Ref/	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	2	0x4D	0x..	0x04

3.5.2.6 zrc_cmd_disc_indication

		Message header	Message pay load		Message trailer
SOT	Protocol ID	Length of payload	Msg code	Pair. Ref/	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	2	0x4E	0x..	0x04

3.5.2.7 zrc_cmd_disc_confirm

		Message header	Message pay load				Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	Pair. Ref/	Supported cmd	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	32 bytes	1 byte
0x01	0x02	35	0x4F	0x..	0x..		0x04

3.5.2.8 zrc_cmd_disc_response

	Message header	Message pay load	Message trailer

SOT	Protocol ID	Length of payload	Msg code	pair. Ref/	Supported cmd	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	32 bytes	1 byte
0x01	0x02	34	0x50	0x..		0x04

3.5.2.9 zrc_cmd_request

	Message header		Message pay load							Message trailer
SOT	Protocol ID	Length of payload	Msg code	pair. Ref/	Vendor id	Cmd code	Tx options	Cmd length	Cmd payload	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	2 bytes	1 byte	1 byte	1 byte	LEN	1 byte
0x01	0x02	7+LEN	0x4A	0x..	Byte 0-1	0x..	0x..	LEN		0x04

3.5.2.10 zrc_cmd_indication

	Message header		Message pay load						Message trailer
SOT	Protocol ID	Length of payload	Msg code	pair. Ref/	Rx Link quality	RX Flags	Nsdu length	nsdu	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	LEN	1 byte
0x01	0x02	5+LEN	0x4B	0x..	0x..	0x..	LEN		0x04

3.5.2.11 zrc_cmd_confirm

	Message header		Message pay load				Message trailer
SOT	Protocol ID	Length of payload	Msg code	status	Pair. Ref/	RC cmd	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	4	0x4C	0x..	0x..		0x04

3.5.2.12 vendor_data_request

	Message header		Message pay load							Message trailer
SOT	Protocol ID	Length of payload	Msg code	pair. Ref/	Profile Id	Vendor Id	Tx Options	Nsdu length	nsdu	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	2 bytes	1 byte	1 byte	LEN	1 byte
0x01	0x02	7+LEN	0x51	0x..	0x..	Byte 0-1	LEN	LEN		0x04

3.5.2.13 vendor_data_indication

	Message header		Message pay load								Message trailer
SOT	Protocol ID	Length of payload	Msg code	pair. Ref/	Profile Id	Vendor Id	RX Link quality	Rx flags	Nsdu length	nsdu	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	2 bytes	1 byte	1 byte	1 byte	LEN	1 byte
0x01	0x02	8+LEN	0x52	0x..	0x..	Byte 0-1	0x..	LEN	LEN		0x04

3.5.2.14 vendor_data_confirm

	Message header	Message pay load	Message trailer

SOT	Protocol ID	Length of payload	Msg code	status	Pair. Ref/	EOT
1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	3	0x53	0x..	0x..	0x04

3.5.2.15 Unsupported cmd

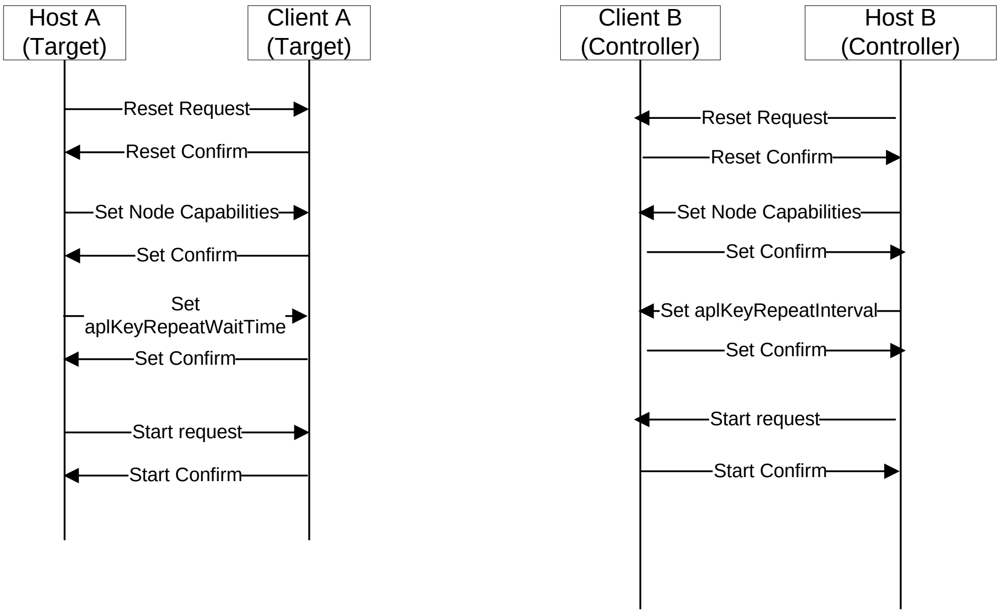
	Message header		Message pay load	Message trailer
SOT	Protocol ID	Length of payload	Msg code	EOT
1 byte	1 byte	1 byte	1 byte	1 byte
0x01	0x02	1	0xFF	0x04

3.5.3 ZRC Serial Interface usage

This section describes the usage of the ZRC Serial Interface. The description is divided into five steps.

1. Initialization
2. Push button pairing
3. RC command discovery
4. RC command handling
5. Vendor-specific data handling

3.5.3.1 Step 1 – Initialization



The initialization step also provides a way to assign values to parameters that need to be set differently than their default values. The table below shows setting the example values.

Command / Message	**Description**	**Byte stream over serial interface (message payload)**
Reset request	Resets the RF4CE stack and underlying layers	0x2a 0x01
Reset confirm	Returns the results of the reset request	0x3d 0x00
Set node capabilities	Sets the capabilities of the RF4CE client, such as node type (target or controller) and security support	Target node: 0x2d 0x73 0x00 0x01 0x0f Controller node: 0x2d 0x73 0x00 0x01 0x0c
Set confirm / node capabilities	Returns the result of the previous set confirm	0x3f 0x00 0x73 0x00
Set aplKeyRepeatInterval	Sets the key repeat interval time on controller	0x2d 0x80 0x00 0x01 0x64
Set Confirm / aplKeyRepeatInterval	Returns the result of the previous set request	0x3f 0x00 0x80 0x00
Set aplKeyRepeatWaitTime	Sets KeyRepeatWaitTime on target	0x2d 0x81 0x00 0x01 0xc8
Set Confirm / aplKeyRepeatWaitTime	Returns the result of the previous set request	0x3f 0x00 0x81 0x00

Command / Message	Description	Byte stream over serial interface (message payload)
Set aplResponseWaitTime	Sets aplResponseWaitTime	0x2d 0x6c 0x04 0x00 0x00 0x6a 0x18
Set Confirm / aplResponseWaitTime	Returns the result of the previous set request	0x3f 0x00 0x6d 0x00
Start request	Starts the RF4CE client	0x2e
Start confirm	Returns the result of the start confirm	0x40 0x00

There is no specific order required for the commands, but the start request command should not be issued before setting the node capabilities.

3.5.3.2 Step 2 – Push button pairing

Command / Message	Description	Byte stream over serial interface (message payload)
pbp_org_pair_request	Starts the push button pairing procedure at controller	0x46 0x13 0x01 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x01 0x01 0x00 0x00 0x00 0x00 0x00 0x00
pbp_rec_pair_request	Starts the push button pairing procedure at target	0x48 0x13 0x02 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00
pbp_org_pair_confirm	Push button pairing status on controller	0x47 0x00 0x00
pbp_rec_pair_confirm	Push button pairing status on target	0x49 0x00 0x00

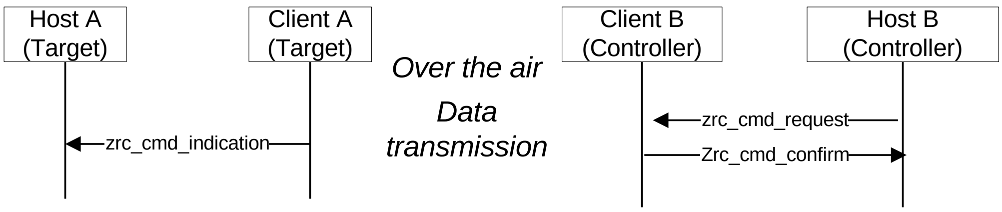
3.5.3.3 Step 3 – RC command discovery

RC command discovery is used to exchange information about the supported commands. After pairing, the target sends the RC command discovery request to the controller. The controller answers back to the target with a response message. After a blackout period, the controller sends the command discovery to the target, and the target sends the response back to controller.

Command / Message	Description	Byte stream over serial interface (message payload)
zrc_cmd_disc_request	Sends the RC command discovery request	0x4c 0x00

Command / Message	Description	Byte stream over serial interface (message payload)
zrc_cmd_disc_indication	Indication on the receiver of the command discovery	0x4d 0x00
zrc_cmd_disc_response	Sends back the response to originator	0x4f 0x00 0x1f 0x06 0x00 0xe0 0xff 0x03 0x13 0x00 0x0f 0x00 0x00 0x00 0x00 0x00 0x1e 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
zrc_cmd_disc_confirm	The status of the command discovery request	0x4e 0x00 0x00 0x1f 0x06 0x00 0xe0 0xff 0x03 0x13 0x00 0x0f 0x00 0x00 0x00 0x00 0x00 0x1e 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

3.5.3.4 Step 4 – RC command handling



Command / Message	Description	Byte stream over serial interface (message payload)
zrc_cmd_request	Sends the RC command from controller to target	0x49 0x00 0xfa 0xff 0x01 0x0c 0x01 0x30
zrc_cmd_indication	Indication on the target for the RC command	0x4a 0x00 0x2e 0x02 0x02 0x01 0x30
zrc_cmd_confirm	The status of the previous RC command	0x4b 0x00 0x00 0x30

4 Serial bootloader support

The serial bootloader firmware is capable of programming (flashing) the device program memory with a new program application image without using a device programmer (e.g. JTAGICEII).

4.1 Functionality Overview

This feature is supported on all the extension boards of Xmega-a3u and Xmega256RFR2.. Please refer to “Supported boards” section 11.4 in the MAC user guide [7].

The bootloader program is also available at **thirdparty\wireless\addons\bootloader.**

Note: In case, if bootloader firmware needs to be programmed using device programmer, then following fuse settings needs to be used

Table 4-1. Recommended fuse settings for applications using serial bootloader

Parameter	Value for RCB
BODLEVEL	Brown-out detection at VCC = 1.8V
OCDEN	Disabled
JTAGEN	Enabled
SPIEN	Enabled
WDTON	Disabled
EESAVE	Enabled
BOOTSZ	Boot flash size = 2048 words; start address = $F800
BOOTRST	Enabled
CKDIV8	Enabled
CKOUT	Disabled
SUT_CKSEL	Internal RC oscillator start-up time = 6CK + 0ms

Fuse settings can also be specified in terms of bytes as given below -

Extended : 0xFE

High : 0x92

Low : 0x42

Serial bootloader consists of two parts: embedded bootstrap code that should be loaded to the flash memory of ATmega128RFA1 and PC based application that sends data to the embedded bootstrap over serial link. Embedded bootstrap code uses the received data to program the internal flash memory of the MCU. A simple communication protocol is used to ensure proper programming. Motorola S-record (SREC) format files are supported as source images for the serial bootloader PC part.

To upload (flash) the new image (.srec extension) into device program application memory (flash), a dedicated serial bootloader PC application (either a GUI or console) is executed on the host. This application is part of BitCloud SDK for megaRF and available at [22].

For more details on serial bootloader programming to flash the image through serial bootloader application, please refer AVR2054 – Serial Bootloader User Guide [22].

To upload the application image using the programmer, we need to merge the boot loader image with the application image (srec_cat tool can be used). Then we have to upload the merged image to the board.

For example, To upload the serial interface application image to Atmega256RFR2 zigbit modules, first we need to run the following command to merge the application image with the bootloader image after generating the application image.

srec_cat Serial_Interface_Platform.hex -intel bootloader.hex -intel -o Serial_Interface_Platform_BT.hex –intel

Then we can use Atmel studio programmer to flash the merged image to the target board.

5 Network and ZRC APIs

This section describes the APIs provided for RF4Control network layer and ZRC profile.Application can access the profile/Network layer using these APIs.These APIs cover the data service and management service primitives as mentioned in the RF4CE specification.

5.1 Network layer APIs

This section explains the APIs provided by the network layer to the application/profile. In all the request APIs, the application needs to provide the callback for the confirmation so that the network layer will call the same after processing the request.

5.1.1 nlde_data_request/confirm

To initiate the data request from the application, the following API should be called.

```
bool nlde_data_request(uint8_t PairingRef, profile_id_t ProfileId,
                uint16_t VendorId, uint8_t nsduLength, uint8_t *nsdu,
                uint8_t TxOptions, uint8_t Handle, FUNC_PTR confirm_cb );
```

Handle parameter can be used by the application to track the data request for the retry handling

The confirmation proto type is shown below.

```
void nlde_data_confirm(nwk_enum_t Status, uint8_t PairingRef,
                        profile_id_t ProfileId, uint8_t Handle );
```

5.1.2 nlme_set_request

This API allows the application to change the NIB attributes.

```
bool nlme_set_request(nib_attribute_t NIBAttribute, uint8_t NIBAttributeIndex,
                        uint8_t *NIBAttributeValue , FUNC_PTR confirm_cb );
```

The confirmation proto type is shown below.

```
void nlme_set_confirm(nwk_enum_t Status, nib_attribute_t NIBAttribute,
```

uint8_t NIBAttributeIndex);

5.1.3 nlme_get_request

This API allows the application to get the NIB attribute value.

```
bool nlme_get_request(nib_attribute_t NIBAttribute, uint8_t NIBAttributeIndex
                                  , FUNC_PTR confirm_cb);
```

The confirmation proto type is shown below.

```
void nlme_get_confirm(nwk_enum_t Status, nib_attribute_t NIBAttribute,
                      uint8_t NIBAttributeIndex, void *NIBAttributeValue);
```

5.1.4 nlme_reset_request

This API allows the application to request a reset of the NWK layer

```
bool nlme_reset_request(bool SetDefaultNIB,FUNC_PTR confirm_cb) ;
```

SetDefaultNIB – true for cold reset

False for warm reset

The confirmation proto type is shown below.

```
void nlme_reset_confirm(nwk_enum_t Status);
```

5.1.5 nlme_start_request

This API allows the application to request the NLME to start a network

```
bool nlme_start_request(FUNC_PTR confirm_cb);
```

The confirmation proto type is shown below.

```
void nlme_start_confirm(nwk_enum_t Status);
```

5.1.6 nlme_rx_enable_request

This API allows the application to request the network layer to either enable (for a finite period or until further notice) or disable the receiver

```
bool nlme_rx_enable_request(uint32_t RxOnDuration, FUNC_PTR confirm_cb);
```

The confirmation proto type is shown below.

```
void nlme_rx_enable_confirm(nwk_enum_t Status);
```

5.1.7 nlme_discovery_request

This API allows the application to request the network layer to discover other devices of interest operating in the POS of the device

```
bool nlme_discovery_request(uint16_t DstPANId, uint16_t DstNwkAddr,
                        uint8_t OrgAppCapabilities,
                        dev_type_t OrgDevTypeList[DEVICE_TYPE_LIST_SIZE],
                        profile_id_t OrgProfileIdList[PROFILE_ID_LIST_SIZE],
                        dev_type_t SearchDevType, uint8_t DiscProfileIdListSize,
                        profile_id_t DiscProfileIdList[PROFILE_ID_LIST_SIZE],
                        uint32_t DiscDuration, FUNC_PTR confirm_cb) ;
```

The confirmation proto type is shown below.

```
void nlme_discovery_confirm(nwk_enum_t Status, uint8_t NumNodes,
                            node_desc_t *NodeDescList);
```

5.1.8 nlme_discovery_indication

This API allows the application to receive the notification that a discovery request command has been received.

```
void nlme_discovery_indication(nwk_enum_t Status, uint64_t SrcIEEEAddr,
                               uint8_t OrgNodeCapabilities, uint16_t OrgVendorId,
                               uint8_t OrgVendorString[7], uint8_t OrgAppCapabilities,
                               uint8_t OrgUserString[15], dev_type_t OrgDevTypeList[3],
                               profile_id_t OrgProfileIdList[7],
                               dev_type_t SearchDevType, uint8_t RxLinkQuality);
```

5.1.9 nlme_discovery_response

This API allows the application to respond to the discovery indication command received from the network layer

```
bool nlme_discovery_response(nwk_enum_t Status, uint64_t DstIEEEAddr,
                             uint8_t RecAppCapabilities, dev_type_t RecDevTypeList[3],
                             profile_id_t RecProfileIdList[7], uint8_t DiscReqLQI);
```

5.1.10 nlme_auto_discovery_request

This API allows the application to request the network layer to handle the receipt of discovery request command frames automatically.

```
bool nlme_auto_discovery_request(uint8_t RecAppCapabilities,
                             dev_type_t RecDevTypeList[DEVICE_TYPE_LIST_SIZE],
                             profile_id_t RecProfileIdList[PROFILE_ID_LIST_SIZE],
                             uint32_t AutoDiscDuration,FUNC_PTR confirm_cb);
```

5.1.11 nlme_pair_request

This API allows the application to request the network layer to pair with another device. This primitive would normally be issued following a discovery operation.

```
bool nlme_pair_request(uint8_t LogicalChannel, uint16_t DstPANId,
                       uint64_t DstIEEEAddr,uint8_t OrgAppCapabilities,
                       dev_type_t OrgDevTypeList[DEVICE_TYPE_LIST_SIZE],
                       profile_id_t OrgProfileIdList[PROFILE_ID_LIST_SIZE],
                       uint8_t KeyExTransferCount, FUNC_PTR confirm_cb );
```

The network layer provides the confirmation to the application after receiving the response from the other node.If no response, it provides the corresponding error code in the status parameter. The confirmation proto type is shown below.

```
void nlme_pair_confirm(nwk_enum_t Status, uint8_t PairingRef,
                  uint16_t RecVendorId, uint8_t RecVendorString[7],
                  uint8_t RecAppCapabilities, uint8_t RecUserString[15],
                  dev_type_t RecDevTypeList[3],
                 profile_id_t RecProfileIdList[7] );
```

5.1.12 nlme_pair_indication

This API allows the application to receive the notification of the reception of a pairing request command

```
void nlme_pair_indication(nwk_enum_t Status, uint16_t SrcPANId,
                   uint64_t SrcIEEEAddr, uint8_t OrgNodeCapabilities,
                   uint16_t OrgVendorId, uint8_t OrgVendorString[7],
                   uint8_t OrgAppCapabilities, uint8_t OrgUserString[15],
                   dev_type_t OrgDevTypeList[3],profile_id_t OrgProfileIdList[7],
                   uint8_t KeyExTransferCount, uint8_t ProvPairingRef);
```

5.1.13 nlme_pair_response

This API allows the application to respond to the pairing request command received via nlme_pair_indication API

```
bool nlme_pair_response(nwk_enum_t Status, uint16_t DstPANId,
                  uint64_t DstIEEEAddr, uint8_t RecAppCapabilities,
                 dev_type_t RecDevTypeList[3],profile_id_t RecProfileIdList[7],
                 uint8_t ProvPairingRef);
```

5.1.14 nlme_unpair_request

This API allows the application to request the network layer to remove a pairing link with another device both in the local and remote pairing tables.

```
bool nlme_unpair_request(uint8_t PairingRef, FUNC_PTR confirm_cb) ;
```

The confirmation proto type is shown below.

```
void nlme_unpair_confirm(uint8_t Status, uint8_t PairingRef);
```

5.1.15 nlme_unpair_indication

This API allows the application to get the notification of the removal of a pairing link by another device

```
void nlme_unpair_indication(uint8_t PairingRef);
```

5.1.16 nlme_unpair_response

This API allows the application to notify the network layer to remove the pairing link indicated via the NLME-UNPAIR.indication primitive from the pairing table

```
bool nlme_unpair_response(uint8_t PairingRef);
```

5.1.17 nlme_update_key_request

This API allows the application to request the network layer to change the security link key of an entry in the pairing table.

```
bool nlme_update_key_request(uint8_t PairingRef, uint8_t NewLinkKey[16]
                             , FUNC_PTR confirm_cb );
```

The confirmation proto type is shown below.

```
void nlme_update_key_confirm(nwk_enum_t Status, uint8_t PairingRef);
```

5.1.18 nlme_ch_agility_request

This API allows the application to configure the channel agility mode.

```
bool nwk_ch_agility_request(nwk_agility_mode_t AgilityMode
                            , FUNC_PTR confirm_cb );
```

The confirmation proto type is shown below.

```
void nwk_ch_agility_confirm(nwk_enum_t Status, bool ChannelChanged,
                            uint8_t LogicalChannel);
```

5.1.19 nlme_ch_agility_indication

This API allows the application to receive the indications when channel agility event has occured, i.e. the base channel has been changed automatically. The new channel is indicated by the parameter LogicalChannel

```
void nwk_ch_agility_indication(uint8_t LogicalChannel);
```

5.2 ZRC profile APIs

5.2.1 zrc_cmd_request

This API allows the application to send the zrc command.The profile will call the confirmation callback provided in the request after processing the request.

```
bool zrc_cmd_request(uint8_t PairingRef, uint16_t VendorId,
                     zrc_cmd_code_t CmdCode, uint8_t CmdLength,
                     uint8_t *Cmd, uint8_t TxOptions, FUNC_PTR confirm_cb);
```

The confirmation proto type is shown below.

```
void zrc_cmd_confirm(nwk_enum_t Status, uint8_t PairingRef,
                     cec_code_t RcCmd);
```

5.2.2 zrc_cmd_indication

This API allows the application to receive the indication for the zrc command from the other node.

```
void zrc_cmd_indication(uint8_t PairingRef, uint8_t nsduLength, uint8_t *nsdu,
          uint8_t RxLinkQuality, uint8_t RxFlags);
```

5.2.3 zrc_cmd_disc_request

This API allows the application to send the zrc command discovery request to the other node.

```
bool zrc_cmd_disc_request(uint8_t PairingRef, FUNC_PTR confirm_cb );
```

The confirmation proto type is shown below.

```
void zrc_cmd_disc_confirm(nwk_enum_t Status, uint8_t PairingRef,
                          uint8_t *SupportedCmd);
```

5.2.4 zrc_cmd_disc_indication

This API allows the application to receive the indication for the command discovery request from the other node.

```
void zrc_cmd_disc_indication(uint8_t PairingRef);
```

5.2.5 zrc_cmd_disc_response

This API allows the application to send the response for the the command discovery request from the other node.

```
bool zrc_cmd_disc_response(uint8_t PairingRef, uint8_t *SupportedCmd);
```

5.3 Registering indication callbacks

The application needs to register the indication callbacks for network layer/ZRC profile at the network startup. The following APIs are provided by Network layer/ZRC profile to the application for registering the indication callbacks.

```
void register_zrc_indication_callback(zrc_indication_callback_t *zrc_ind_callback);
void register_nwk_indication_callback(nwk_indication_callback_t *nwk_ind_cb);
```

The application needs to define the structure of the corresponding indication callback (zrc/network) and fill it with the required callbacks. In case of partially filled indication structure, we have to initialize other callbacks to NULL to ignore them. Then it should call the corresponding API passing the structure as an argument.

5.3.1 Indication structure

The structure used for registering the network indication callbacks is shown below.

```
typedef struct nwk_indication_callback
{
    nwk_ch_agility_indication_cb_t nwk_ch_agility_indication_cb;
    nlme_unpair_indication_cb_t nlme_unpair_indication_cb;
    nlme_pair_indication_cb_t nlme_pair_indication_cb;
    nlme_discovery_indication_cb_t nlme_discovery_indication_cb;
    nlme_comm_status_indication_cb_t nlme_comm_status_indication_cb;
    zrc_data_indication_cb_t zrc_data_indication_cb;
    nlde_data_indication_cb_t nlde_data_indication_cb;
} nwk_indication_callback_t;
```

The structure used for registering the ZRC indication callbacks is shown below.

```
typedef struct zrc_indication_callback
{
    zrc_cmd_indication_cb_t zrc_cmd_indication_cb;
    zrc_cmd_disc_indication_cb_t zrc_cmd_disc_indication_cb;
    vendor_data_ind_cb_t vendor_data_ind_cb;
} zrc_indication_callback_t;
```

6 Appendix

6.1 Applications along with the supported platforms

S.No	Application	Supported Platforms
1	NWK – Serial Interface	Host – SAM4L_XPLAINED_PRO NCP – AT32UC3A256S_RZ600_AT86RF212 AT32UC3A256S_RZ600_AT86RF231 ATMEGA256RFR2_XPLAINED_PRO ATXMEGA256A3U_RF212_ZIGBIT_EXT ATXMEGA256A3U_RF212_ZIGBIT_USB ATXMEGA256A3U_RF233_ZIGBIT_USB XMEGA_A3BU_XPLAINED_RZ600_RF212 XMEGA_A3BU_XPLAINED_RZ600_RF231
2	ZRC - Serial Interface – Controller and Target	Host – SAM4L_XPLAINED_PRO NCP – AT32UC3A256S_RZ600_AT86RF212 AT32UC3A256S_RZ600_AT86RF231 ATMEGA256RFR2_XPLAINED_PRO ATXMEGA256A3U_RF212_ZIGBIT_EXT ATXMEGA256A3U_RF212_ZIGBIT_USB ATXMEGA256A3U_RF233_ZIGBIT_USB XMEGA_A3BU_XPLAINED_RZ600_RF212 XMEGA_A3BU_XPLAINED_RZ600_RF231
3	ZRC - Button Controller	XMEGA_A3BU_XPLAINED_RZ600_RF212 XMEGA_A3BU_XPLAINED_RZ600_RF231
4	ZRC - Single button Controller	Host – SAM4L_XPLAINED_PRO NCP – ATMEGA256RFR2_XPLAINED_PRO
5	ZRC - Terminal Target	AT32UC3A256S_RZ600_AT86RF212 AT32UC3A256S_RZ600_AT86RF231 ATXMEGA256A3U_RF212B_ZIGBIT_USB ATXMEGA256A3U_RF233_ZIGBIT_USB

7 Abbreviations

API	Application Programming Interface
BMM	Buffer Management Module

CRC	Cyclic Redundancy Check
CEC	Consumer Electronics Control
CSMA	Carrier Sense Multiple Access
EOT	End Of Text
ED	Energy Detection
EEPROM	Electircally Erasable Programmable Read only memory
FOTA	Firmware Over The Air
I2C	Inter-Integrated Circuit
LED	Light-Emitting Diode
MCU	Micro Controller Uint
MAC	Medium Access Control
NVM	Non-Volatile Memory
NCP	Network Co-processor
PBP	Push Button Pairing
PAL	Platform Abstraction Layer
QMM	Queue Management Module
ORG	Originator
REC	Recipient
RCB	Radio Controller Board
RF4CE	Radio Frequency For Consumer Electronics
RC	Remote Control
SAL	Security Abstraction Layer
STB	Security Tool Box
SPI	Serial Programming Interface
SBC	Single Button Controller
SOC	System On Chip

8 References

[1] IEEE Standard 802.15.4TM-2006: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)
http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf

[2] Atmel RF4Control – ZigBee RF4CE Certified Platform
http://www.atmel.com/dyn/products/tools_card_mcu.asp?tool_id=4712

[3] Atmel AT86RF212, 700/800/900 MHz transceiver for IEEE 802.15.4
http://www.atmel.com/dyn/products/product_card.asp?part_id=4349

[4] Atmel AT86RF231; Low Power 2.4 GHz Transceiver for IEEE 802.15.4
http://www.atmel.com/dyn/products/product_card.asp?part_id=4338

[5] Atmel Atmega256RFR2; Microcontroller with Low Power 2.4GHz Transceiver for ZigBee™ and IEEE 802.15.4™
http://www.atmel.com/devices/ATMEGA256RFR2.aspx

[6] Atmel AVR Atxmega56a3
http://www.atmel.com/devices/atxmega256a3.aspx

[7] AVR2025: IEEE 802.15.4 MAC Software Package under the asf link
thirdparty\wireless\avr2025_mac

[8] Atmel ATAVRRZ541 AVR Z-Link 2.4 GHz Packet Sniffer Kit
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4187

[9] Atmel Studio 6, IDE for writing and debugging AVR applications
http://www.atmel.in/microsite/atmel_studio6/

[10] Atmel AVR JTAGICE mkII
http://www.atmel.com/tools/AVRJTAGICEMKII.aspx

[11] ZigBee RF4CE
http://www.zigbee.org/rf4ce

[12] ZigBee RF4CE Specification Version 1.00; http://www.zigbee.org
094945r00ZB_RF4CE-Specification.pdf

[13] ZigBee RF4CE ZRC Profile Specification; http://www.zigbee.org
094946r00ZB_RF4CE-CERC-Profile-Specification.pdf

[14] High-Definition Multimedia Interface, Specification Version 1.3a
http://www.hdmi.org

[15] Sensor Terminal Board, Dresden Elektronik GmbH
http://www.dresden-elektronik.de/shop/prod75.html

[16] RCB Breakout Board, Dresden Elektronik GmbH
http://www.dresden-elektronik.de/shop/prod84.html

[17] ATmega128RFA1 Evaluation Kit (EK1)
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4677

[18] AVR2044: RCB128RFA1 – Hardware User Manual
http://www.atmel.com/dyn/products/app_notes.asp?family_id=676

[19] Minimalist GNU for Windows
http://www.mingw.org or http://sourceforge.net/projects/mingw/

[20] Srec cat tool for windows

http://www.srecord.sourceforge.net/

[21] Atmel Software Framework
http://www.atmel.com/tools/avrsoftwareframework.aspx
http://asf.atmel.com/docs/latest/

[22] BitCloud SDK for megaRF and AVR2054 Serial Bootloader User Guide
http://www.atmel.com/bitcloud

9 Document revision history

Please note that the referring page numbers in this section are referring to this document. The referring revisions in this section are referring to the document revision.

9.1 Rev. 8357E-MCU Wireless-02/13

Released with version AVR2102_RF4Control_v_2_0_0

This release version is a re-architecture for the NWK-MAC/TAL Interface layer to have the Microcontroller specific library approach.

The stack is ported into Atmel Software Framework [21]

9.2 Rev. 8357D-MCU Wireless-06/12

Released with version AVR2102_RF4Control_v_1_4_0

Section 5 – Network and ZRC APIs added

Section 3.5.2 – ZRC serial interface message structure added

Section 3.4.4 – Serial interface message structure added

Section 6.1 – Updated with the newly added platforms

Table 2.7 – Updated with newly added build switches

Section 3.1.4.3 – Procedure to initiate push button pairing updated

9.3 Rev. 8357C-MCU Wireless-08/11

Update section “Omitting 32 kHz crystal”

9.4 Rev. 8357B-MCU Wireless-08/11

Released with version AVR2102_RF4Control_v_1_3_0

Section ‘serial bootloader support’ added.

Section ‘special stack configuration’ added

9.5 Rev. 8357A-MCU Wireless-01/11

Released with version AVR2102_RF4Control_v_1_0_1-1_2.1

Editorial changes of the RF4Control user guide

User guide document number changed

9.6 Rev. 2102C-MCU Wireless-11/10

Released with version AVR2102_RF4Control_v_1_0_1-1_2

ZRC profile layer introduced including sections describing ZRC features and handling

Push button pairing added as separate layer

Vendor data handling added

ZRC Target application added

ZRC Serial Interface application added

ATmega128RFA1-EK1 support added

9.7 Rev. 2102B-MCU Wireless-04/10

Released with version AVR2102_RF4Control_v_1_0_1-1_1_Lib.zip

Section “Serial interface usage” added

Table “Message codes and message lengths for the ZRC API.” updated

Section “Channel agility” added

Section “Single Button Controller example application” added

9.8 Rev. 2102A-MCU Wireless-12/09

Initial Version: Internal hex file release

Released with version AVR2102_RF4Control_v_1_0_1-1_0.zip

10 Table of Contents

Enabling Unlimited Possibilities®

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
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 Building
1-6-4 Osaki
Shinagawa-ku, Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370