NIVIS™

Smart Object API Integration Manual

Version 1.5

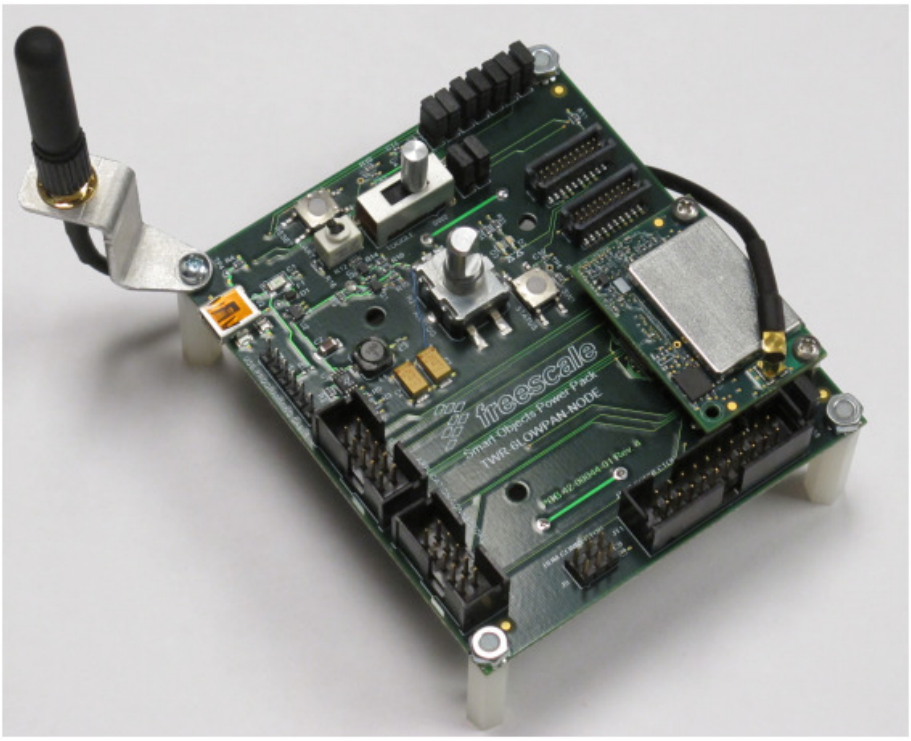

Table of Contents

1 Introduction

1.1 Document purpose

Thismanual provides information on how to integrate the user's board with the Power Pack/Smart Object Radio Module. The document defines a communication solution based on the UART interface andspecifies the API through which an application processor can communicate with the Smart Object in order to send and receive application- and network-related status data. With minimal firmware development effort, the user can add another layer of intelligence to the devices responsible for data harvesting, allowing them to communicate interactively over the Internet. Because the representation or transfer of data to the monitoring system is not of concern to the user, the user can focus on the application layer.

1.2 Audience

This document is intended for firmware developers who are involved in the development of the firmware on the user's application processor.

1.3 Acronyms and abbreviations

ACK	Acknowledgement
API	Application Programming Interface
AP	Application Processor
APDU	Application Protocol Data Unit
CRC	Cyclic Redundancy Check
DL	Data Link Layer
GPIO	General Purpose Input Output
FW	Firmware
HW	Hardware
LQI	Link Quality Indicator
NACK	Not Acknowledgement
REST	Representational State Transfer
RM	Radio Module (i.e., K60)
RSSI	Received Signal Strength Indication
Rx/RX	Reception
SPI	Serial Peripheral Interface
TAI	International Atomic Time
TL	Transport Layer
Tx/TX	Transmission
UART	Universal Asynchronous Receiver/Transmitter

2 Hardware Integration

2.1 Overview

The Power Pack board was designed to allow easy interfacing with the Nivis Smart Object radio module, offering integrators access to the main HW interfaces.Figure 1 shows the main components on the Power Pack board.

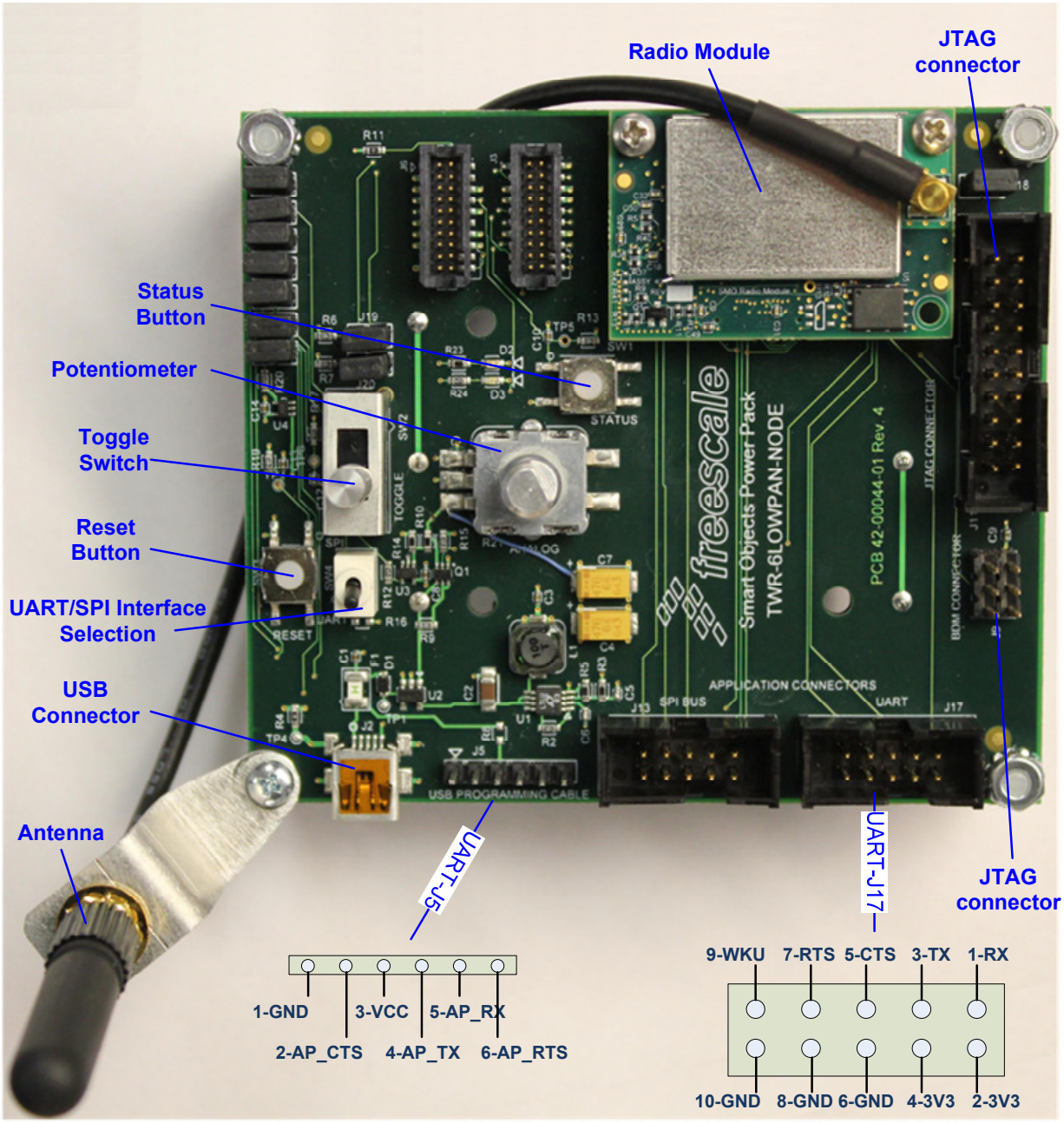

Figure 1 – Power Pack board components

The connector used for UART communication with the application processor is J17 (the arrow in Figure 1 indicates pin 1). The pins on UART-J5 header are connected to the corresponding pins on UART-J17 headerand can be easily probed during integration (e.g., logic analyzer). The RADIO_WKU signal can also be probed using pin 9 on the SPI connector.

The connector used for SPI communication with the application processor is J13 (the arrow in Figure 1 indicates pin 1).

2.2 UART integration

In order to use UART communication, the SW4switch (UART1_RTS pin on the radio module) needs to be in the UART position at device power-on.

The following settings are used for the UART interface:

- Default baud rate: 115200
- Bits: 8
- Parity: None
- Stop Bits: 1
- Flow Control: None

Applications utilizing the API interface will be provided with a means of adjusting the UART communication speed/baud rate (API Command API_UPDATE_UART_SPEED). The change in baud rate by the application processor will be 'approved' by the RM processor to ensure that it can still communicatereliably.

NOTE Lower UART baud rates are likely to be the bottleneck in the system.

Figure 2 shows the pins for the UARTinterface from the Smart Objects Radio Moduleto their equivalent signals on the application processor.

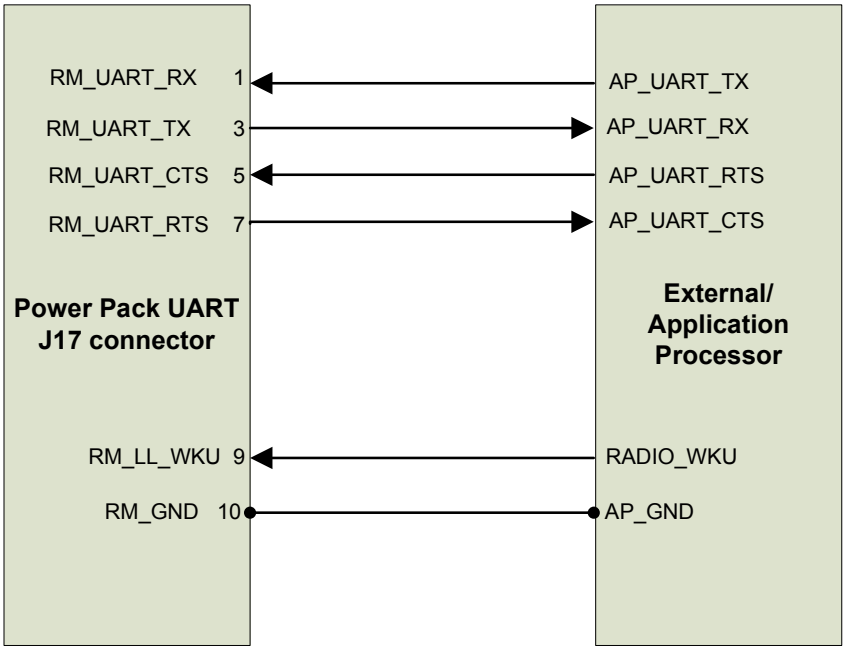

Figure 2 – UART interface on Power Pack board

Table 1 summarizes the use of signals for UART integration.

Table 1 – UART integration: Signal name, direction, and use

Signal name	Direction	Use
RM_UART1_TX	Output	As in UART communication,TX and RX are used for data transfer.
RM_UART1_RX	Input	
RM_UART1_RTS (AP_UART1_CTS)	Output	This signal is active low and is used by the modem to: - indicate a want-to-send state. The signal will return to high state after a high to low transition of the RM_UART1_CTS (AP_UART1_RTS)signal. -indicate ready-to-receive state. It will be generated as a response to the AP CPU RADIO_WKU signal.
RM_UART1_CTS (AP_UART1_RTS)	Input	This signal is active low and is used by the application CPU to indicate a ready-to-receive state. It will Implicitly confirm the acquisition of RM_UART1_RTS low signal to the modem.
RM_LL_WKU	Input	This signal is active low and is used by the application CPU to signal the intention to communicate with the Radio Module. For a low power device, it will wake up the Radio Module CPU from sleep mode.

Figure 3 shows the signal transition diagram.

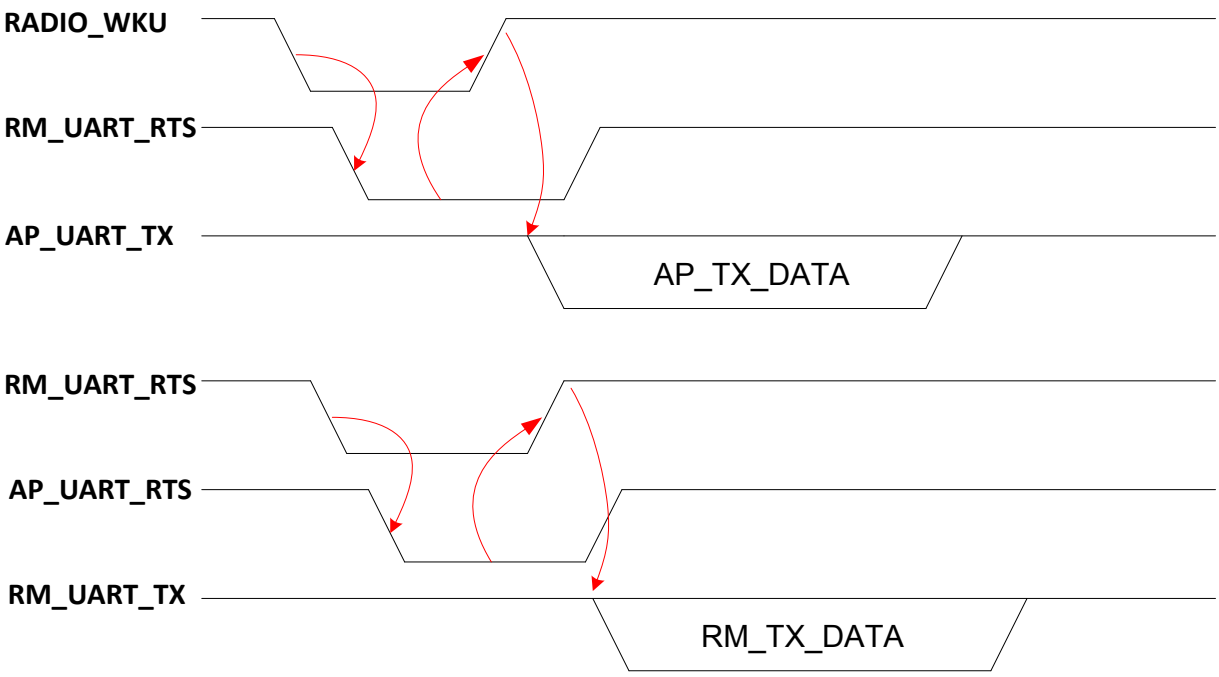

Figure 3 – Signal transition diagram

- There is no flow control for transmitting. Once message transmission has started, it will flow without restrictions and without awareness of the receiving capability of the destination. The destination should be capable of receiving bytes at the arriving speed.
- After activating the RM_UART1_RTS signal, the radio module waits for a maximum of 250ms for the application processor to become ready to receive, while checking the RM_UART1_CTS (AP_UART1_RTS) signal. If RM_UART1_CTS is low, the radio module (RM) starts transmitting the data. If250ms has elapsed without any confirmation from the application processor that it can receive the message, RM_UART1_RTS becomes high, and the process is repeated as follows:
    - If the message that needs to be transmitted is a response, the RM processor will try two times to send it, each time activating the RM_UART1_RTS signal and waiting 250 ms for the RM_UART1_CTS signal to become low. If response cannot be sent after two attempts, the request will be removed and the application processor will need to resend the request in order to receivea response from the RM.
    - If the message that needs to be transmitted is a request, the RM processor will try three times to send it, each time activating the RM_UART1_RTS signal and waiting 250 ms for the RM_UART1_CTS signal to become low.If request cannot be sent after three attempts, the process is repeated infinitely for a GET_RESOURCES_LIST request orfour times for other types of requests. The timeout between requests is 1 second for line-powered devices and 30 seconds for battery-powered devices.
    - If the message is a request andis successfully sent, the RM will wait for a response/ACK/NACK from the application processor.

2.3 SPI integration

In order to use SPI communication, the SW4 switch (UART1_RTS pin on the radio module) needs to be in the SPI position at device power-on.

Table 2 summarizes the use of signals for SPI integration.

Table 2 – SPI integration: Signal name, direction, and use

Signal name	Direction	Use
SPI1_SCLK	Output	Serial clock
SPI1_MOSI	Output	Master output, slave input
SPI1_MISO	Input	Master input, slave output
SPI1_CS	Output	Chip select (active low)
RM_LL_WKU	Input	This signal is active low and is used by the application CPU to signal the intention to communicate with the radio module. For a low-power device, this will wake up the radio module CPU from sleep mode.

The radio module processor is the master of communication. The radio modulehas the SPI clockset to the default frequency of 100KHz.To begin a communication, the RM transmits the logic 0 on the CS pin. The active state for the CS pin is Low. The RM will then wait for the application processor to become ready to receive (this is useful in case the application processor needs extra time to wake up before being able to receive an SPI message).The application processor must indicate the “ready to receive” state by putting the WKU line in low. After the WKU line becomes low, the RM processor starts sending the data on the SPI1_MOSI line.

If the application processor has any message to be read by the radio moduleprocessor, it can signal the master by using the WKU signal. The active state for WKU is Low. The radio module will query the application processor when it is ready to receive data.

Applications utilizing the SPI interface are provided with a means of adjusting the SPI communications speed using the command API_UPDATE_SPI_SPEED. The change in speed by the application processor must be ‘approved’ by the radio module processor.

The SPI signals have a “HIGH” logic level of +3Vdc and a “LOW” logic level of 0Vdc.

Figure 4 shows the SPI signal transition diagram.

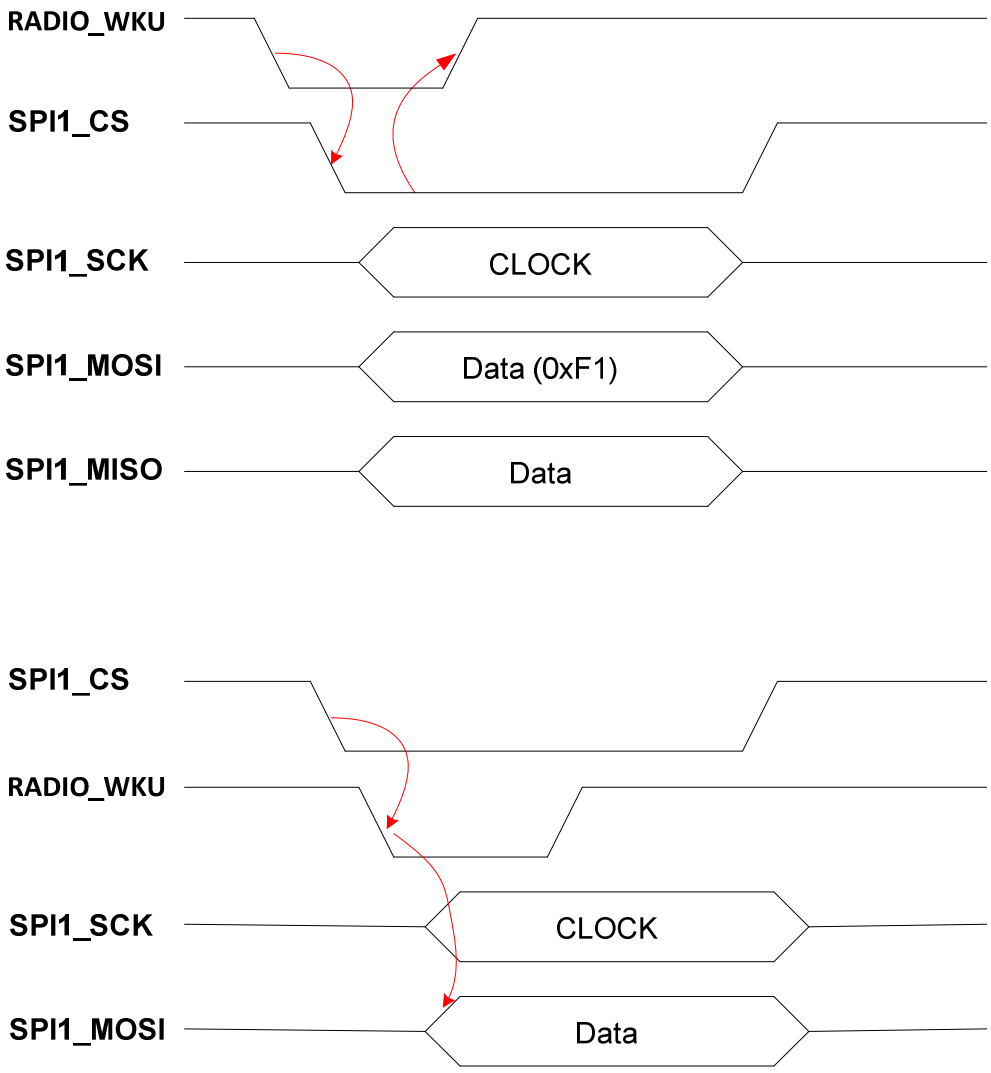

Figure 4 – SPI signal transition diagram

The following settings are used for SPI communication:

- Default Speed: 100 KHz
- Max Speed: 2 MHz
- Clock polarity: 0
- Clock phase: 0 - data is captured on the clock's rising edge (low→high transition) and data is propagated on a falling edge (high→low clock transition)

3 Software Integration

3.1 Overview

The Smart Objects API was designed around the architectural elements of REST, in order to permit transfer of representations of resources between a server and a client. A resource can be any source of specific information (e.g.,sensor data temperature and pressure data, metering data, list of events on a home security controller, lamp status, or lamp schedule profile) that can be referenced with a global identifier (URI).The application processor can define its own list of resources that can be accessed later for reading or writing.

3.2 Communication flow

Communication between the application processor and the radio moduleprocessor is based on two main kinds of packets: requests and responses. All request packets must be followed by a response/ACK/NACK. If the RM processor does not receive a response/ACK/NACK within 250 ms, the request will be repeatedas follows:

- Infinitely for a GET_RESOURCES_LIST request until an ACK/NACK is received.
- Four timesfor all other request types or until a response/ACK/NACK is received.

The timeout between retries is 1 second for line-powered devices and 30 seconds for battery-powered devices.

A message is considered as response to a request if it meets all of the following criteria:

- It is a response (the Req/Rsp Flag from the Message Header must be 1; see Message header_Message_Header_1).
- The Message ID field is the same as in the request message.

3.3 API message format

3.3.1 Overview

Table 3 describes the API message format.

Table 3 – API message format

Field	Size (Bytes)	Values	Comments
STX	1	0xF0	This is the start character for every message. When it is received, the receiver discards any other Rx message in progress and starts receiving this new message.
Message header	1	Bit field	See Message header.
Message type	1		Depends on Message Class in the Message Header.
Message ID	1	0-255	Used to match up requests and responses.
Data size	2		The number of bytes in the Data field.
Data	0…X		The request/response message data.
CRC	2		CRC is based on a standard CRC algorithm (CCITT-CRC, 0x1021 as the polynomial) and includes everything between, but not including, the STX, CRC, and ETX. The initial value is 0xFFFF.
ETX	1	0xF1	This is the end character for every message.

3.3.2 Message header

Table 4 describes the message header.

Table 4 – Message header

Bit	7	6	5	4	3	2	1	0
Description	Message Class				Req/ Rsp Flag	Reserved		

The **Request/Response Flag** indicates whether a message is a request/response (0 = Request, 1=Response).

Table 5 lists available message classes.

Table 5 – Message classes

Field	Size (Bits)	Values	Comments
Message Class	4	1 = API_COMMANDS 2 = STACK SPECIFIC 3 = UDP TRANSFER 4 = ACK 5 = NACK 0,6-15 = RESERVED	

3.3.3 Special characters

Table 6 lists special characters.

Table 6 – Special characters

Char	Values	Comments
STX	0xF0	Start of packet
ETX	0xF1	End of packet
CHX	0xF2	Escape character

The STX is a special character that indicates the start of a new packet. The sender is allowed to abort the current packet and start a new packet by sending the STX in the middle of a packet. The ETX is a special character that indicates the end of a message. Therefore, the packet data needs to be protected if it contains any STX or ERX characters. The escape character CHX is used for this purpose.

NOTE The CRC field is calculated based on the un-escaped packet data.

All packet characters including the CRC field (excluding STX and ETX fields) will be escaped with the escape character CHX if any of the characters in the packet is:

- STX (0xF0): It will be replaced with two characters: 0xF2 (CHX) and 0x0F (1's complement of 0xF0)
- ETX (0xF1): It will be replaced with two characters: 0xF2 (CHX) and 0x0E (1's complement of 0xF1)
- CHX (0xF2): It will be replaced with two characters: CHX (0xF2) and 0x0D (1's complement of 0xF2)

In other words, whenever the receiver receives a CHX character, it should discard it and the next character is 1's complement.

3.4 API commands

3.4.1 Summary of message types

Table 7 lists the sub-commands/message types for message class API_COMMANDS and a short description for each API command.

Table 7 – Message types for API_COMMANDS

Message Type	Values	Description
API_GET_AP_PLATFORM_INFO	1	(AP←RM) Reads platform information from AP.
API_GET_RM_PLATFORM_INFO	2	(AP→RM) Reads platform information structure (HW version, FW version, API protocol version, low power support, battery) from Radio Module.
API_UPDATE_SPI_SPEED	3	(AP→RM) Sets SPI speed.
API_UPDATE_UART_SPEED	4	(AP→RM) Sets UART speed.
API_GET_CURRENT_TIME	5	(AP→RM) Gets current time from Radio Module.

Command Direction:

AP→RM Application Processor (AP) to Radio Module (RM)

AP←RM Radio Module (RM) to Application Processor (AP)

3.4.2 API_GET_AP_PLATFORM_INFO (1) (AP←RM)

3.4.2.1 Overview

This command is used by the RM to read platform related informationfromthe AP.The implementation of this command is optional.

3.4.2.2 Request

Table 8 describes the request format for API_GET_AP_PLATFORM_INFO.

Table 8 – Request format for API_GET_AP_PLATFORM_INFO

Field	Size (Bytes)	Values	Comments
Data size	2	0x00	

3.4.2.3 Response

Table 9 describes the response format for API_GET_AP_PLATFORM_INFO.

Table 9 – Response format for API_GET_AP_PLATFORM_INFO

Field	Size (Bytes)	Values	Comments
Data size	2	16	
HW version	8		ASCII
FW version	8		ASCII

3.4.3 API_GET_RM_PLATFORM_INFO (2) (AP→RM)

3.4.3.1 Overview

This command is used by AP to read platform related information from RM.

3.4.3.2 Request

Table 10 describes the request format for API_GET_RM_PLATFORM_INFO.

Table 10 – Request format for API_GET_RM_PLATFORM_INFO

Field	Size (Bytes)	Values	Comments
Data size	2	0x00	

3.4.3.3 Response

Table 11 describes the response format for API_GET_RM_PLATFORM_INFO.

Table 11 – Response format for API_GET_RM_PLATFORM_INFO

Field	Size (Bytes)	Values	Comments
Data size	2	20	
HW version	8		ASCII
FW version	8		ASCII
API protocol version	2		Unsigned integer, MSB first
Low power support	1	0 = device is line powered 1 = device is battery powered	
Battery lifetime	1	Estimated battery lifetime in percent [0-100] or 101 if line powered.	

3.4.4 API_UPDATE_SPI_SPEED (3) (AP→RM)

3.4.4.1 Overview

This command is used by the AP to update the communication speed for SPIinterface.

3.4.4.2 Request

Table 12describes the request format for API_UPDATE_SPI_SPEED.

Table 12 – Request format for API_UPDATE_SPI_SPEED

Field	Size (Bytes)	Values	Comments
Data size	2	1	
Data	1	1 – 25000 2 – 50000 3 – 75000 4 - 100000 5 - 200000 6 – 250000 7 – 500000 8 - 1000000 9 - 2000000	4 - 100000 is default

3.4.4.3 Response

The response format for API_UPDATE_SPI_SPEED is:

- ACK – if the request is accepted.
- NACK – if the request is not accepted.

3.4.5 API_UPDATE_UART_SPEED (4) (AP→RM)

3.4.5.1 Overview

This command is used by the AP to update the communication speed for UARTinterface.

3.4.5.2 Request

Table 13describes the request format for API_UPDATE_UART_SPEED.

Table 13 – Request format forAPI_UPDATE_UART_SPEED

Field	Size (Bytes)	Values	Comments
Data size	2	1	
Data	1	1 – 9600 2 – 19200 3 – 38400 4 - 115200 5 - 230000	4 - 115200 is default

3.4.5.3 Response

The response format forAPI_UPDATE_UART_SPEED is:

- ACK – if the request is accepted. The UART speed is changed immediately and the ACK is sent at the new UART speed.
- NACK – if the request is not accepted.

3.4.6 API_GET_CURRENT_TIME (5) (AP→RM)

3.4.6.1 Overview

Sent by the AP, this command retrieves the current time from the RM.

3.4.6.2 Request

Table 14describes the request format for API_GET_CURRENT_TIME.

Table 14 – Request format forAPI_GET_CURRENT_TIME

Field	Size (Bytes)	Values	Comments
Data size	2	0	

3.4.6.3 Response

Table 15describes the response format forAPI_GET_CURRENT_TIME.

Table 15 – Response format for API_GET_CURRENT_TIME

Field	Size (Bytes)	Values	Comments
Data size	2	4	
Current time	4		Unsigned integer MSB

3.4.6.4 Stack-specific commands

Table 16lists stack-specific commands for API_GET_CURRENT_TIME.

Table 16 – Stack-specific commands for API GET_CURRENT_TIME

Message Type	Type ID	Comments
GET_JOIN_STATUS	1	(AP→RM) Requests network join status (4E registration).
NOTIFY_JOIN	2	(AP←RM)Network join status change notification (4E registration).
GET_RPL_INFO	3	(AP→RM) Requests RPL related information.
GET_LINK_QUALITY	4	(AP→RM) Requests radio link quality information.
GET_RESOURCES_LIST	20	(AP←RM) Requests list of resources.
RESOURCE_LIST_INDICATION	21	(AP→RM) Sends resource list to the RM. Resources will be appended to the resource directory defined by the RM and uploaded to the proxy server at boot time.
READ_RESOURCE	22	(AP←RM) Reads current representation of a resource.
WRITE_RESOURCE	23	(AP←RM) Writesthe specified resource.
CHANGE_RESOURCE	24	(AP -> RM) Force a read resource request

3.4.7 GET_JOIN_STATUS(1) (AP→RM)

3.4.7.1 Overview

This command is used by the AP to request network join status information from the RM.

3.4.7.2 Request

Table 17describes the request format for GET_JOIN_STATUS.

Table 17 – Request format for GET_JOIN_STATUS

Field	Size (Bytes)	Values	Comments
Data size	2	0	

3.4.7.3 Response

Table 18describes the response format for GET_JOIN_STATUS.

Table 18 – Response format for GET_JOIN_STATUS

Field	Size (Bytes)	Values	Comments
Data size	2	1	
Data	1	0 – Discovery Mode 1 – Joining Mode 2– Joined	

3.4.8 NOTIFY_JOIN (2) (AP←RM)

3.4.8.1 Overview

This command is used by the radio moduleprocessor to inform the application processor when the device changed join status.

3.4.8.2 Request

Table 19describes the request format for NOTIFY_JOIN.

Table 19 – Request format for NOTIFY_JOIN

Field	Size (Bytes)	Values	Comments
Data size	2	1	
Data	1	0 – Discovery Mode 1 – Joining Mode 2 – Joined 3 – Resource discovery confirmed	

3.4.8.3 Response

The response format for NOTIFY_JOIN is:

- ACK – if there are no errors in the request packet.
- NACK – if the request is not accepted.

3.4.9 GET_RPL_INFO (3) (AP→RM)

3.4.9.1 Overview

This command is used by the application processor to read RPL related information from the radio module processor.

3.4.9.2 Request

Table 20describes the request format for GET_RPL_INFO.

Table 20 – Request format for GET_RPL_INFO

Field	Size (Bytes)	Values	Comments
Data size	2	0	

3.4.9.3 Response

Table 21describes the response format for GET_RPL_INFO.

Table 21 – Response format for GET_RPL_INFO

Field	Size (Bytes)	Values	Comments
Data size	2	1	
RPL RANK	2	0..65535	Unsigned integer, MSB

3.4.10 GET_LINK_QUALITY (4) (AP→RM)

3.4.10.1 Overview

This command is used by the application processor to read radio link quality information from the radio module processor.

3.4.10.2 Request

Table 22describes the request format for GET_LINK_QUALITY.

Table 22 – Request format for GET_LINK_QUALITY

Field	Size (Bytes)	Values	Comments
Data size	2	0	

3.4.10.3 Response

Table 23describes the response format for GET_LINK_QUALITY.

Table 23 – Response format for GET_LINK_QUALITY

Field	Size (Bytes)	Values	Comments
Data size	2	2	
LQI	1	255	*(to be added in the future)*
RSSI	1	0..255	

3.4.11 GET_RESOURCES_LIST (20)(AP←RM)

3.4.11.1 Overview

This command is used by the radio moduleprocessor to inform theapplication processorthat it should start sending RESOURCE_LIST_INDICATION commands.It is sent at boot time every 1 second for line power devices and every 30 seconds for low power devices, until the AP responds with an ACK or a NACK. After receiving this request, the AP must send a confirmation response in the form of an ACK or NACK message. If the confirmation is not received by the RM processor, the message is retransmitted infinitely until AP sends a response.

3.4.11.2 Request

Table 24describes the request format for GET_RESOURCES_LIST.

Table 24 – Request format for GET_RESOURCES_LIST

Field	Size (Bytes)	Values	Comments
Data size	2	0	

3.4.11.3 Response

The response format for GET_RESOURCES_LIST is:

- ACK – if there are no errors in the request packet.
- NACK – if the request is not accepted.

3.4.12 RESOURCE_LIST_INDICATION (21)(AP→RM)

3.4.12.1 Overview

This command is used by the application processor to send its list of resources to the radio module. Multiple RESOURCE_LIST_INDICATION messages can be sent to the RM, each containing definitions of more than one resource, such that the total size of the message is less than 255 bytes (maximum API buffer size) and each resource definition is complete. To indicate that there are no more resources that need to be defined, the application processor will terminate the list of resources with a dummy resource with ID 0xFF and empty body. The maximum number of resources that can be defined by the application processor is 4.API Use Cases contains an example describing this command.

3.4.12.2 Request

Table 25describes the request format for RESOURCE_LIST_INDICATION.

Table 25 – Request format for RESOURCE_LIST_INDICATION

Field	Size (Bytes)	Values	Comments
Data size	2		
Array of Resource Definition			

3.4.12.3 Resourcedefinition

Table 26describes the resource definition for RESOURCE_LIST_INDICATION.

Table 26 – Resource definition for RESOURCE_LIST_INDICATION

Field	Size (Bytes)	Values	Comments
Resource ID	1	1 -0xFF	
URI String Size	1		URI MUST be limited to 18 characters.
URI String			URI String MUST not start with '/' character but may contain multiple URI segments (separated by '/' character). The total size for the URI String MUST be less than 18 characters. E.g.: *power/instant*
Resource Type String Size	1		Resource Type MUST be limited to 19 characters.
Resource Type String			
Interface String Size	1		Interface MUST be limited to 9 characters.
Interface String			Interfaces "Adm", "App0" and "Ema" are reserved for the Radio Module.
Resource Size	2		Maximum size estimate: gives an indication of the maximum size of the resource indicated by the target URI.
Content Type	1		0 = text/plain; charset=utf-8 42 = application/octet-stream 47 = application/exi
Number Of VariableDefinitions	1		
Array of Variables Definition			See Variable Definition

3.4.12.4 Variable definition

Table 27describes the variable definition for RESOURCE_LIST_INDICATION.

Table 27 – Variable definition for RESOURCE_LIST_INDICATION

Field	Size (Bytes)	Values	Comments
Variable ID	1		
Name String Size	1		
Name String			Name string must be maximum 15 characters long
Type ID	1		

3.4.12.5 Predefined types

Table 28lists predefined types for RESOURCE_LIST_INDICATION.

Table 28 – Predefined types for RESOURCE_LIST_INDICATION

Type	Type ID	Comments
Uint8	1	MSB first
Uint16	2	MSB first
Uint32	3	MSB first
Int8	4	MSB first
Int16	5	MSB first
Int32	6	MSB first
ShortOctetStream	8	[1Bytes Len][Data]

3.4.12.6 Response

The response format for RESOURCE_LIST_INDICATION is:

- ACK – if there are no errors in the request packet.
- NACK – if the request is not accepted.

3.4.13 READ_RESOURCE (22)(AP←RM)

3.4.13.1 Overview

The radio modulesends this message to application processor to request the most recent representation of the resource.

3.4.13.2 Request

Table 29describes the request format for READ_RESOURCE.

Table 29 – Request format for READ_RESOURCE

Field	Size (Bytes)	Values	Comments
Data size	2	1	
Resource ID	1	1-0xFE	
Parameter size	0/2		Big endian order
Parameter value	Parameter size		Received via on demand COAP request parameter (**od=***Parameter Value*)

3.4.13.3 Response

Table 30describes the response format for READ_RESOURCE.

Table 30 – Response format for READ_RESOURCE

Field	Size (Bytes)	Values	Comments
Data size	2		
Resource ID	1	1 -0xFE	
Array of Variable Values			

3.4.13.4 Variable values

Table 31describes variable values for READ_RESOURCE.

Table 31 – Variable values for READ_RESOURCE

Field	Size (Bytes)	Values	Comments
Variable ID	1		
Type ID	1		
Variable Value			

3.4.14 WRITE_RESOURCE (23)(AP←RM)

3.4.14.1 Overview

The radio modulesends this message to application processor to update the representation of the resource.

3.4.14.2 Request

Table 32describes the request format for WRITE_RESOURCE.

Table 32 – Request format for WRITE_RESOURCE

Field	Size (Bytes)	Values	Comments
Data size	2	1	
Resource ID	1	1 -0xFE	
Array of Variable Values			

3.4.14.3 Variable values

Table 33describes variable values for WRITE_RESOURCE.

Table 33 – Variable values for WRITE_RESOURCE

Field	Size (Bytes)	Values	Comments
Variable ID	1		
Type ID	1		
Variable Value			

3.4.14.4 Response

Table 34describes the response format for WRITE_RESOURCE.

Table 34 – Response format for WRITE_RESOURCE

Field	Size (Bytes)	Values	Comments
Data size	2	1	
Resource ID	1	1 -0xFE	
Response Code	1	0 or 1	0 = success 1 = failure

3.4.15 CHANGE_RESOURCE (24)(AP-> RM)

3.4.15.1 Overview

The radio module sends this message to announce a value change on a specific resource.

3.4.15.2 Request

Table 35describes the request format for CHANGE_RESOURCE.

Table 35 – Request format for CHANGE_RESOURCE

Field	Size (Bytes)	Values	Comments
Data size	2	1	
Resource ID	1	1 -0xFE	
Action	1	1	Notify RM about change of value

3.4.15.3 Response

The response format for CHANGE_RESOURCE is:

- ACK–if thereareno errorsin the requestpacket.
- NACK–ifthe requestis notaccepted.

3.5 UDP-specific

3.5.1 UDP_CREATE_SOCKET (1) (AP→RM)

3.5.1.1 Overview

The application processor sends this messageto the radio module to request the creation of a UDP socket. Maximum one UDP socket designated to AP can exist at one point in time.

3.5.1.2 Request

Table 36describes the request format for UDP_CREATE_SOCKET.

Table 36 – Request format for UDP_CREATE_SOCKET

Field	Size (Bytes)	Values	Comments
Data size	2	2	
UDP_PORT	2	0 …65535	UDP ports 32000, 32001, 44965, 5683 are used by Radio Module.

3.5.1.3 Response

The response format for UDP_CREATE_SOCKET is:

- ACK – if there are no errors in the request packet, the request is accepted and a listener is created.
- NACK – if there are errors in the request packet or the request is not accepted (i.e., restricted UDP port).

3.5.2 UDP_DELETE_SOCKET (2) (AP→RM)

3.5.2.1 Overview

The application processor sends this message to the radio module to request the deletion of the UDP socket previously created. Since there is only one UDP socket associated with the AP, this command accepts no parameters.

3.5.2.2 Request

Table 37describes the request format for UDP_DELETE_SOCKET.

Table 37 – Request format for UDP_DELETE_SOCKET

Field	Size (Bytes)	Values	Comments
Data size	2	2	

3.5.2.3 Response

The response format for UDP_DELETE_SOCKET is:

- ACK – if there are no errors in the request packet, the request is accepted and the socketis deleted, or there is not any listener on the port associated with the AP.
- NACK – if the request is not accepted.

3.5.3 UDP_SEND_DATAGRAM (3) (AP→RM)

3.5.3.1 Overview

The application processor sends this message to the radio module processor requesting the transfer of data to the IPv6 destination address and destination UDP port specified in the payload.

3.5.3.2 Request

Table 38describes the request format for UDP_SEND_DATAGRAM.

Table 38 – Request format for UDP_SEND_DATAGRAM

Field	Size (Bytes)	Values	Comments
Data size	2	2	
UDP_PORT destination	2	0 ...65535	
UPD_IPv6 destination	16		
PAYLOAD			

3.5.3.3 Response

The response format for UDP_SEND_DATAGRAM is:

- ACK – if there are no errors in the request packet and the request is accepted.
- NACK – if the request is not accepted.

3.5.4 UDP_RECEIVE_DATAGRAM (4) (AP←RM)

3.5.4.1 Overview

This command is used by the RM to deliver a UDP datagram to the AP.

3.5.4.2 Request

Table 39describes the request format for UDP_RECEIVE_DATAGRAM.

Table 39 – Request format for UDP_RECEIVE_DATAGRAM

Field	Size (Bytes)	Values	Comments
Data size	2	2	
UDP_PORT source	2	0 ...65535	
UPD_IPv6 source	16		
PAYLOAD			

3.5.4.3 Response

The response format for UDP_RECEIVE_DATAGRAM is:

- ACK – if there are no errors in the request packet.
- NACK – if the request is not accepted.

3.6 Response ACK

3.6.1 Overview

The ACK response confirms that an API request message (Request/Response field = 0) identified by the Message ID field has been received properly (CRC was successfully validated) and interpreted (the message was validated successfully based on the Message Class and Message Type fields).

Based on the received ACK response, the receiver should remove the already-sent API request message identified by the ACK Message ID field and no retry mechanism should be applied for this request.

3.6.2 Message format

Table 40 describes the message format for the response ACK.

Table 40 – Message format for response ACK

Field	Size (Bytes)	Values	Comments
STX	1	0xF0	When this character is received, the receiver discards any other Rx message in progress and starts receiving this new message.
Message Header	1	0x48	ACK
Message Type	1		1 = Data received properly 2= Message processed and sent via RF channel. 3= Change to API accepted (i.e. UART Speed Change)
Message ID	1	0xNN	NN = Message ID used in referencing this message transaction.
Data size	2	0x0	The number of data bytes in the buffer
CRC	2		
ETX		0xF1	End of message character

3.7 Response NACK

3.7.1 Overview

The NACK response confirms that the API request message (Request/Response field = 0) identified by the Message ID field has been properly received (CRC was successfully validated) but there is inconsistent data inside the API message (e.g., Message Type or/and Class unknown, or the requested operation(s) fail(s)).

3.7.2 Message format

Table 41 describes the message format for the response NACK.

Table 41 – Message format for response NACK

Field	Size (Bytes)	Values	Comments
STX	1	0xF0	When this character is received, the receiver discards any other Rx message in progress and starts receiving this new message.
Message Header	1	0x58	NACK
Message Type	1	0xXX	1 = NACK_CRC_FAIL = 1 (used internally) 2 = NACK_BAD_UART_BAUDRATE, 3 = NACK_UNKNOWN_COMMAND, 4 = NACK_UNKNOWN_API_TYPE, 5 = NACK_UNKNOWN_COAP_TYPE, 6 = NACK_UNKNOWN_UDP_TYPE, 7 = NACK_UNKNOWN_NACK_TYPE, 8 = NACK_UNKNOWN_ACK_TYPE, 9 = NACK_FAILURE, 10 = NACK_INVALID_SIZE, 11 = NACK_RES_ID_NOT_FOUND, 12 = NACK_DUPLICATE_RES_ID, 13 = NACK_VAR_DEF_NOT_FOUND, 14 = NACK_BAD_TYPE_ID, 15 = NACK_RES_LIST_ALREADY_DEFINED, 16 = NACK_MAX_APP_RES_NO_REACHED, 17 = NACK_BAD_URI, 18 = NACK_BAD_RT, 19 = NACK_BAD_IF, 20 = NACK_BAD_CONTENT_TYPE, 21 = NACK_BAD_VAR_NAME, 22 = NACK_BAD_VAR_ID, 23 = NACK_REQ_NOT_EXPECTED, 24 = NACK_MSG_NOT_FOUND, 25 = NACK_UNSUPPORTED_FEATURE
Message ID	1	0xNN	NN = Message ID used in referencing this message transaction.
Data size	2	0x0	The number of data bytes in the buffer
CRC	2		
ETX	1	0xF1	End of message character

4 API Use Cases

4.1 Overview

This section describes the command flows between the application processor and the radio module processor with sequence diagrams in different scenarios.

4.2 Resource discovery

This section describes the sequence of messages exchanged between the radio module and the application processor from device start until it uploads the list of resources on the proxy server.

Figure 5 shows an example in which the resource content type is EXI.

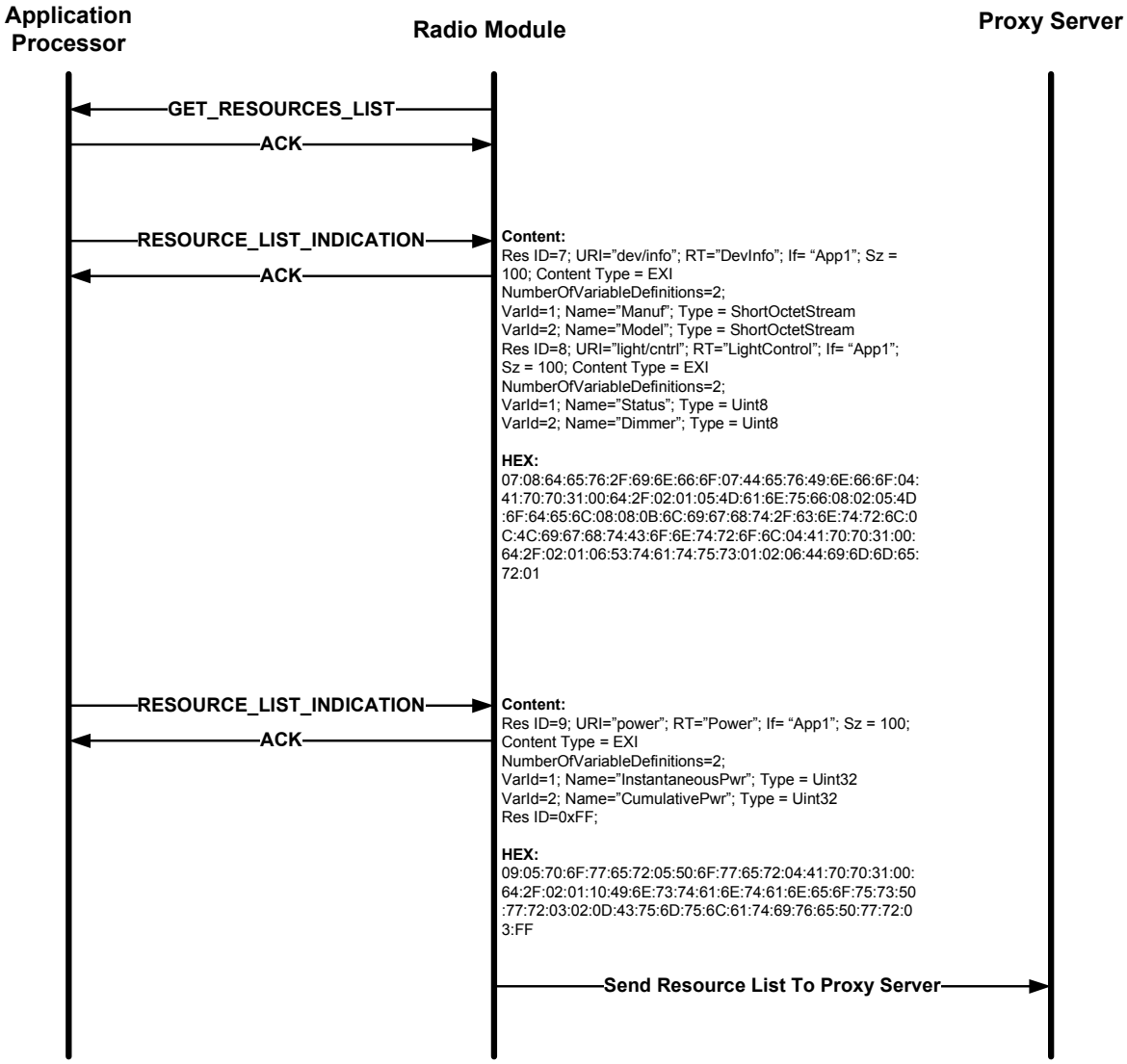

Figure 5 – Example 1: Content type of resource is EXI

As shown in Figure 5:

1. After power ON, the radio module processor sends a GET_RESOURCE_LIST command to the application processor. It will keep sending this command until it receives an ACK/NACK from the AP.
2. After sending the response to the previous command (in the form of either an ACK or a NACK), the AP starts sending RESOURCE_LIST_INDICATION commands to the RM. Each of these commands is acknowledged by the RM.
3. The RM receives the list of resources defined by the AP, terminated by a resource with ID 0xFF. After it receives resource ID 0xFF, anysubsequentresource definition contained in a RESOURCE_LIST_INDICATION message will be rejected with a NACK message.

Figure 6 shows an example in which the resource content type is Text/Plain. The steps described above also apply in this example.

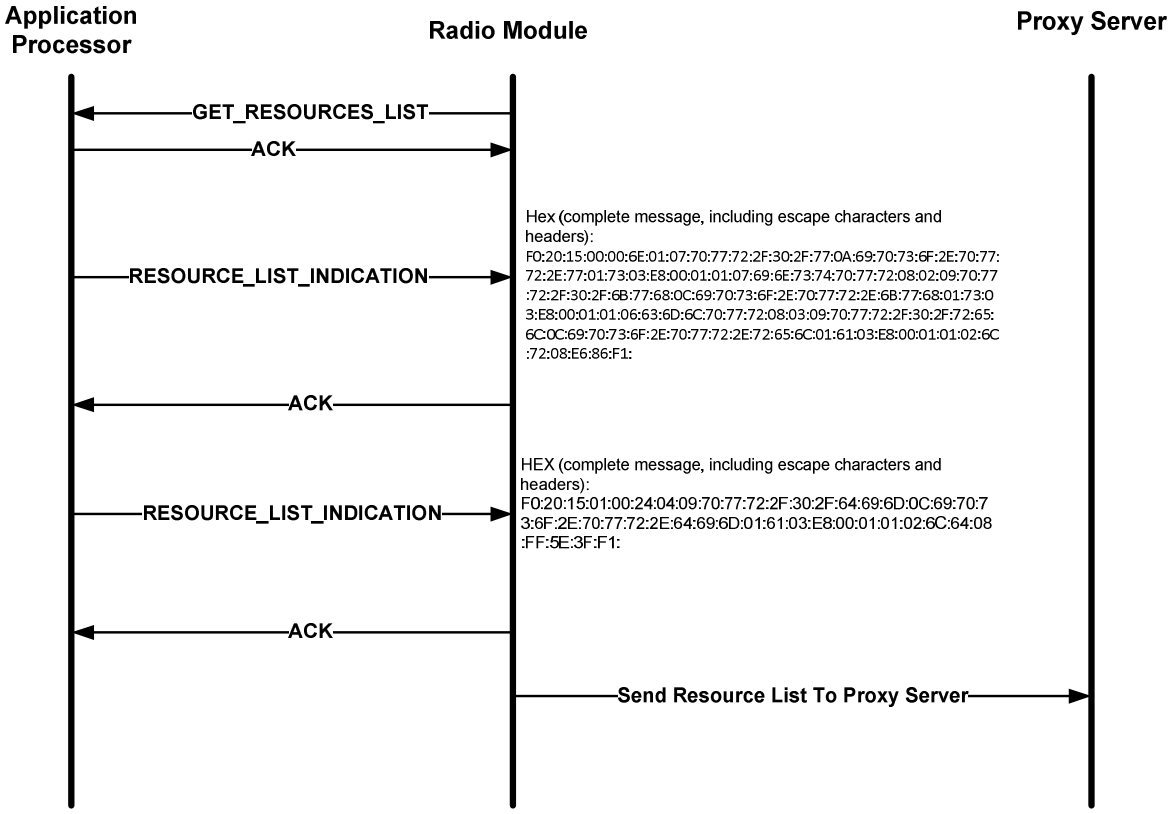

Figure 6 – Example 2: Content type of resource is Text/Plain
(implementation of IPSO Application Framework Power set (http://www.ipso-alliance.org/wp-content/media/draft-ipso-app-framework-04.pdf))

Messages exchanged between AP and RM:

1/7/2013 12:49:54 PM,568 AP Received: **F0:20:14:01:00:00:FF:DE:F1:**

F0 = STX

20 = Message Header (2 = Stack Specific; 0 = Request Flag; 0 = Reserved)

14 = Message Type = GET_RESOURCE_LIST

01 = Message ID

00:00 = Data size

FF:DE = CRC

F1 = ETX

1/7/2013 12:49:54 PM,604 AP Sent: **F0:48:01:01:00:00:43:CD:F1:**

F0 = STX

48 = Message Header (4 =API; 1 = Response Flag; 0 = Reserved)

01 = Message Type = Data Received Properly

01 = Message ID (must be equal to the message ID in the request)

00:00 = Data size

43:CD = CRC

F1 =ETX

1/7/2013 12:49:54 PM,616 AP Sent:
F0:20:15:00:00:6E:01:07:70:77:72:2F:30:2F:77:0A:69:70:73:6F:2E:70:77:72:2E:77:01:73:03:E8:00:01:01:
07:69:6E:73:74:70:77:72:08:02:09:70:77:72:2F:30:2F:6B:77:68:0C:69:70:73:6F:2E:70:77:72:2E:6B:77:68:
01:73:03:E8:00:01:01:06:63:6D:6C:70:77:72:08:03:09:70:77:72:2F:30:2F:72:65:6C:0C:69:70:73:6F:2E:70:
77:72:2E:72:65:6C:01:61:03:E8:00:01:01:02:6C:72:08:E6:86:F1:

F0 = STX

20 = Message Header (2 = Stack Specific; 0 = Request Flag; 0 = Reserved)

15 = Message Type = RESOURCE_LIST_INDICATION

00 = Message ID

00:6E = Data size

01 = Resource ID

07 = URI String Size

70:77:72:2F:30:2F:77 = URI String (ASCII “pwr/0/w”)

0A = Resource Type String Size

69:70:73:6F:2E:70:77:72:2E:77 = Resource Type String (ASCII “ipso.pwr.w”)

01 = Interface Type String Size

73 = Interface Type String (ASCII “s”)

03:E8 = Resource Size

00 = Content Type (Text Plain)

01 = Number of Variable Definitions

01 = Variable ID

07 = Name String Size

69:6E:73:74:70:77:72 = Name String (ASCII “instpwr”)

08 = Type ID (ShortOctetStream)

02 =Resource ID

09 = URI String Size

70:77:72:2F:30:2F:6B:77:68: = URI String (ASCII “pwr/0/kwh”)

0C = Resource Type String Size

69:70:73:6F:2E:70:77:72:2E:6B:77:68 = Resource Type String (ASCII “ipso.pwr.kwh”)

01 = Interface String Size

73 = Interface String (ASCII “s”)

03:E8 = Resource Size

00 = Content Type (Text Plain)

01 = Number of Variable Definitions

01 = Variable ID

06 = Name String Size

63:6D:6C:70:77:72:= Name String (ASCII “cmlpwr”)

08 = Type ID (ShortOctetStream)

03 =Resource ID

09 = URI String Size

70:77:72:2F:30:2F:72:65:6C = URI String (ASCII “pwr/0/rel”)

0C = Resource Type String Size

69:70:73:6F:2E:70:77:72:2E:72:65:6C = Resource Type (ASCII “ipso.pwr.rel”)

01 = Interface String Size

61 = Interface String (ASCII “a”)

03:E8 = Resource Size

00 = Content Type (Text Plain)

01 = Number of Variable Definitions

01 = Variable ID

02 = Name String Size

6C:72 = Name String (ASCII “lr”)

08 = Type ID (ShortOctetStream)

E6:86 = CRC

F1 = ETX

1/7/2013 12:49:54 PM,620 AP Received: **F0:48:01:00:00:00:74:FD:F1:**

F0 =STX

48 = Message Header (4 =API; 1 = Response Flag; 0 = Reserved)

01 = Message Type = Data Received Properly

00 = Message ID

00:00 = Data size

74:FD = CRC

F1 = ETX

1/7/2013 12:49:54 PM,627 AP Sent:
F0:20:15:01:00:24:04:09:70:77:72:2F:30:2F:64:69:6D:0C:69:70:73:6F:2E:70:77:72:2E:64:69:6D:01:61:03:
E8:00:01:01:02:6C:64:08:FF:5E:3F:F1:

F0 = STX

20 = Message Header (2 = Stack Specific; 0 = Request Flag; 0 = Reserved)

15 = Message Type = RESOURCE_LIST_INDICATION

01 = Message ID

00:24 = Data size

04 = Resource ID

09 =URI String Size

70:77:72:2F:30:2F:64:69:6D = URI String (ASCII “pwr/0/dim”)

0C = Resource Type String Size

69:70:73:6F:2E:70:77:72:2E:64:69:6D

01 = Interface String Size

61 = Interface String (ASCII “a”)

03:E8 = Resource Size

00 = Content Type (Text Plain)

01 = Number Of Variable Definitions

01 = Variable ID

02 = Variable Name String Size

6C:64 = Variable Name String (ASCII “ld”)

08 = Type ID (ShortOctetStream)

FF = Resource ID (required to indicate that there are no more resources to be defined)

5E:3F = CRC

F1 = ETX

1/7/2013 12:49:54 PM,630AP Received: **F0:48:01:01:00:00:43:CD:F1:**

F0 = STX

48 = Message Header (4 =API; 1 = Response Flag; 0 = Reserved)

01 = Message Type = Data Received Properly

01 = Message ID

00:00 = Data size

43:CD = CRC

F1 = ETX

4.3 Read resource

This section describes the sequence of messages exchanged when the application processor has to respond to a READ_RESOURCErequest command.

Figure 7 shows an example in which the resource content type is EXI.

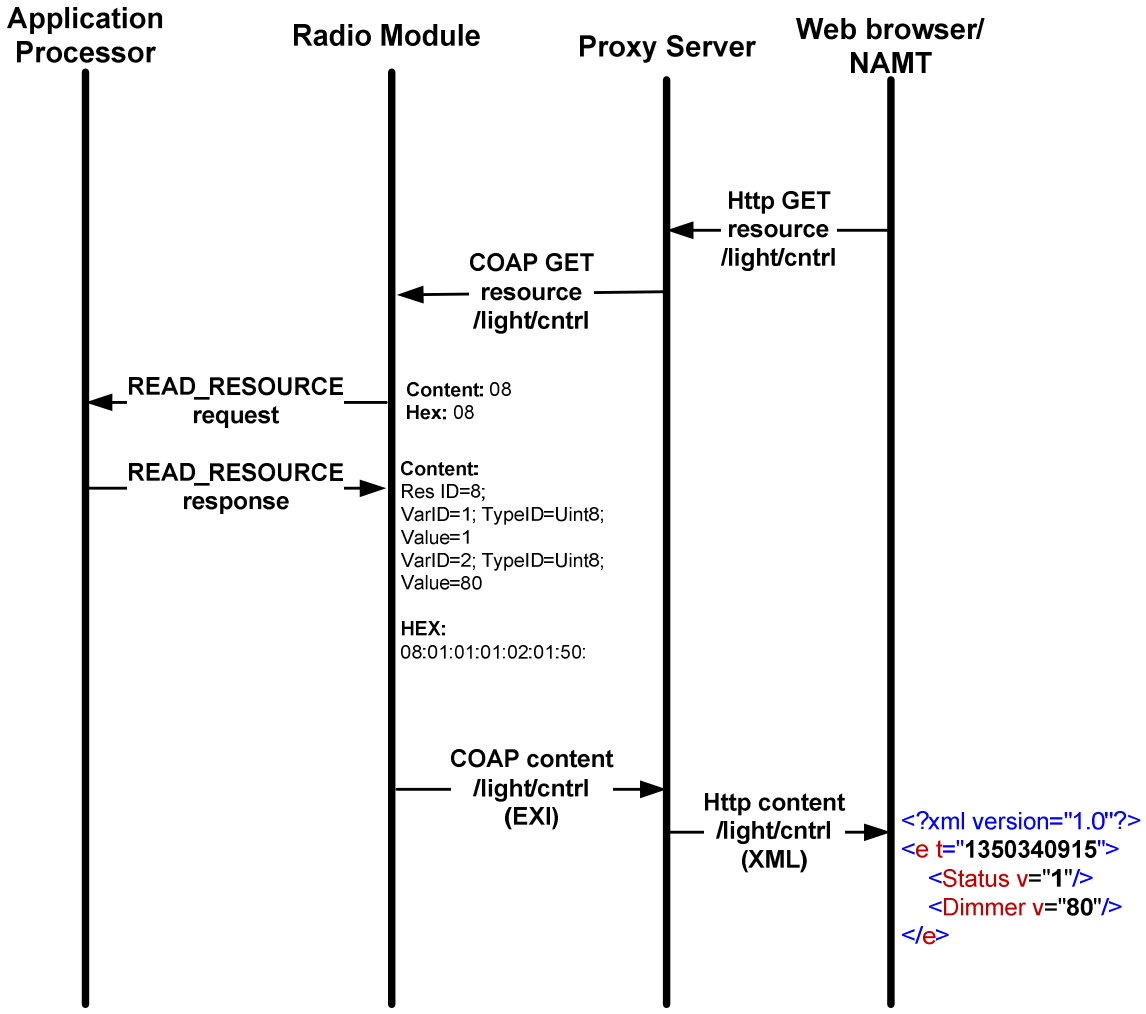

Figure 7 – Example 1: Content Type of resource is EXI

As shown in Figure 7:

1. The radio module sends a READ_RESOURCE request to the application processor, specifying the resource ID.
2. The application processor responds with the resource representation at that moment of time.

Figure 8 shows an example in which the resource content type is Text/Plain. The steps described above also apply in this example.

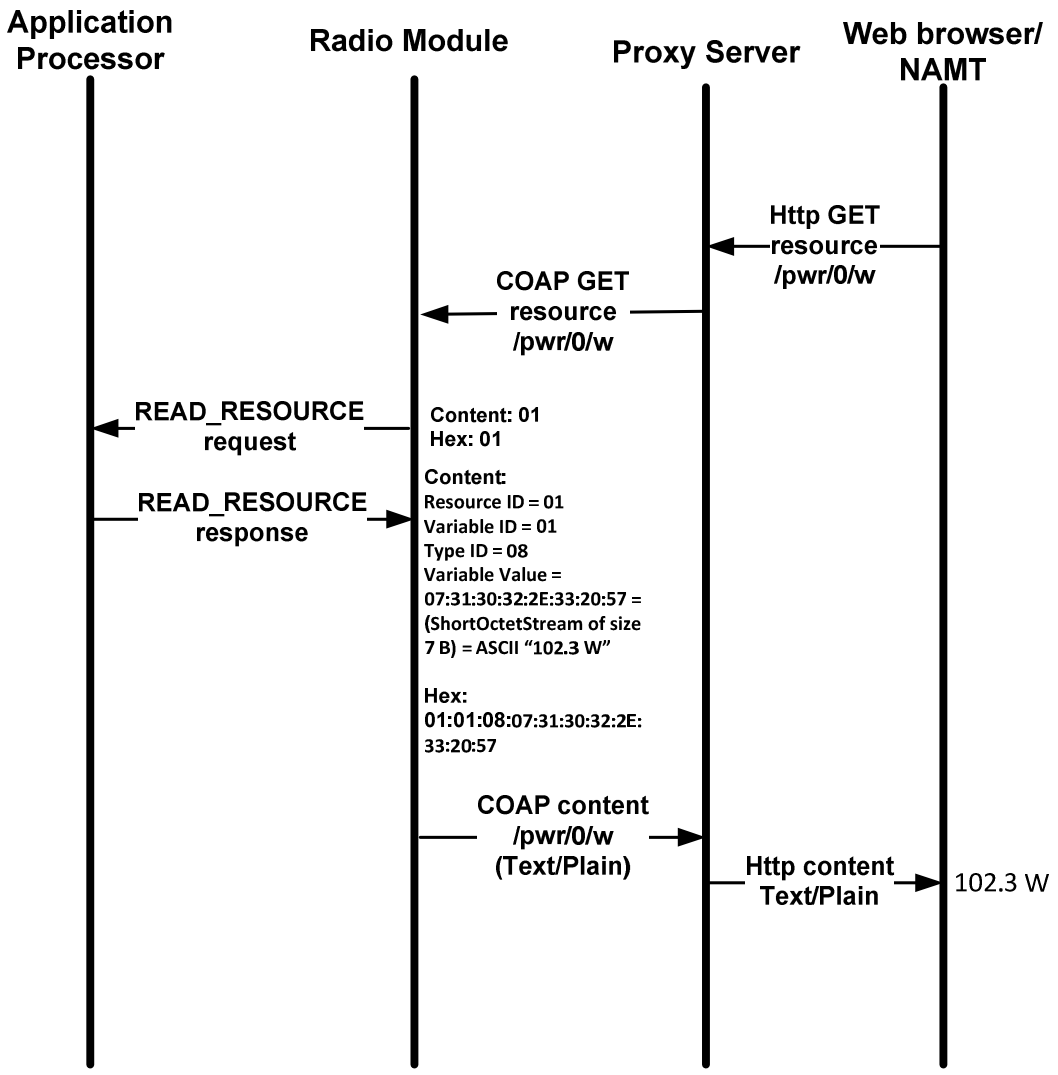

Figure 8 – Example 2: content type of resource is Text/Plain
(implementation of IPSO Application Framework Power set (http://www.ipso-alliance.org/wp-content/media/draft-ipso-app-framework-04.pdf))

Messages exchanged between AP and RM:

1/7/2013 4:46:42 PM,10 AP Received: **F0:20:16:64:00:01:01:42:F4:F1:**

F0 =STX

20 = Message Header (2 = Stack Specific; 0 = Request Flag; 0 = Reserved)

16 = Message Type (READ_RESOURCE)

64 = Message ID

00:01 = Data size

01 = Resource ID

42:F4 = CRC

F1 = ETX

1/7/2013 4:46:42 PM,44 AP Sent: F0:28:16:64:00:0B:01:01:08:07:31:30:32:2E:33:20:57:0A:6B:F1:

F0 =STX

28 = Message Header (2 = Stack Specific; 1 = Request Flag; 0 = Reserved)

16 = Message Type (READ_RESOURCE)

64 = Message ID

00:0B = Data size

01 = Resource ID

01 = Variable ID

08 = Type ID

07:31:30:32:2E:33:20:57 = Variable Value (ShortOctetStream of size 7 B as indicated by the first octet) = ASCII “102.3 W”

0A:6B = CRC

F1 = ETX