

TagRunners Communication Protocol

©Baracoda [™] – Sept. 2009



SUMMARY

<u>SUMMARY</u>	<u>2</u>
<u>REVISION HISTORY</u>	<u>3</u>
<u>1. INTRODUCTION</u>	<u>4</u>
1.1. GENERALITIES	4
1.2. GENERIC PACKET	4
<u>2. COMMUNICATION PROTOCOL</u>	<u>5</u>
2.1. BIDIRECTIONAL PACKETS	5
2.1.1. CONTROL MESSAGES	5
2.2. SCANNER TO HOST MESSAGES	6
2.2.1. ENCAPSULATION SCHEME	6
2.2.2. DATA STRING FORMAT	6
2.3. HOST TO SCANNER MESSAGES	7
2.3.1. COMMUNICATION MESSAGES	7
2.3.2. SCANNER MESSAGES	9
2.3.3. USER INTERFACE MESSAGES	11
2.3.4. MISCELLANEOUS MESSAGES	14
2.3.5. CAPTURE MESSAGES	18
<u>APPENDIX 1: BLUETOOTH PROTOCOL</u>	<u>20</u>
<u>APPENDIX 2: RFID COMMUNICATION PROTOCOL</u>	<u>29</u>
<u>APPENDIX 3: RFID TAG DATA READ/WRITE EXAMPLES ... ERREUR ! SIGNET NON DEFINI.</u>	

Revision History

Changes to the original manual are listed below.

Document	Date	Description
1.0	23 Dec. 08	Initial release
1.2	23 Dec. 08	Modified §2.2.1, 2.2.2. Updated command 0x76 and 0xE0 Added command 0xDE
1.3	23 Dec. 08	Added Annex RFID protocol
1.4	23 Dec. 08	Remove ProxRunner info
1.5	20 Mar.09	Modified §2.2.2 Remove command 0x5A & 0x5B ; 0x54 & 0x55 ; 0xF8 & 0xF9 ; 0xFC & 0xFD Add commands 0xD0 & 0xD1 Annex RFID protocol : add 0x07
1.6	18 Sept.09	Add Appendix 3 (RFID tag read/write examples)

Introduction

Generalities

TagRunners is a wireless RFID data capture product (13.56MHz).

This document is detailing the protocol of communication between the Baracoda TagRunners and its foreign environment through Radio Frequency link (ie: Bluetooth).

Wireless communication is based on the Bluetooth protocol, thanks to the embedded Baracoda Equinox Bluetooth Stack.

HF Tag reading / encoding capabilities are enabled thanks to a RFID antenna & decoder.

The messages described in this document can be:

- Host to scanner messages: the packet is sent only by the host to the scanner
- Scanner to host messages: the packet is sent only by the scanner to the host
- Bidirectional messages: the packet format is the same whether it is sent by the host or the scanner

<http://www.baracoda.com>

Generic packet

All the frames described in this document are formatted as shown:

Code ID	Length	Payload
1 Byte	2 Bytes	N Bytes

- 1 byte for code ID

Bits 7:5 is the logical device

Bits 4:1 is the command

Bit 0: when set, the message must be acknowledged

- 2 bytes for the size of the payload (big-endian), including the sequence number byte which is considered as part of the payload

- Payload (including 1 byte for sequence number when applicable).

The response will have the same code ID as the command.

Communication protocol

Bidirectional packets

I.1.1. Control messages

I.1.1.1. Specific packets

Code ID	Description	Frame
0x01	Legacy	0x01 0x01 0x01 Or 0x01 0x02 0x01

These two (2) sequences will be recognized and purged for backward compatibility with older Baracoda products.

I.1.1.2. Acknowledgment packets

Code ID	Description	Frame
0x06	ACK	0x06 0x01 0xYY
0x15	NACK	0x15 0x01 0xYY

These messages acknowledge the reception of a valid message with the expected sequence number 0xYY, before processing it.

For captured data from the scanner, ACK and NAK have the same meaning but will trigger a different event on the scanner.

I.1.1.3. Synchronization packet

Code ID	Description	Frame
0x16	SYN	0x16 0x01 0xYY

This message acknowledges the reception of a message to acknowledge with an unexpected sequence number. 0xYY is the expected sequence number.

The device will resynchronize its remote sequence number when receiving this message.

Scanner to host messages

I.1.2. Encapsulation scheme

Code ID	Description	Payload
0x34–0x35	RFID data (TagID)	DATA string

I.1.3. Data string format

Timestamp	Data Prefix	Capture Prefix	Protocol Prefix	Protocol Identifier	RFID data	Protocol Suffix	Capture suffix	Data suffix
12 bytes	0-32 bytes	0-32 bytes	0-4 bytes	0-3 bytes	-	0-4 bytes	0-32 bytes	0-32 bytes

Host to scanner messages

I.1.4. Communication messages

Code ID	0x40-0x41
Description	Get Communication Descriptor
Payload	None
Response	2 bytes: {Wireless link: (Bit 0: Bluetooth)} {Wired link: (Bit 0: Serial Dock)}

Code ID	0x42-0x43
Description	Get Retransmission Parameters
Payload	None
Response	2 bytes: {Max number of retransmission, 1 to 0xFE, 0xFF = infinity} {Delay between transmission, 1 to 0xFF, in tenth of seconds}

Code ID	0x44-0x45
Description	Set Retransmission Parameters
Payload	2 bytes: {Max number of retransmissions, 1 to 0xFE, 0xFF = infinity} {Delay between transmissions, 1 to 0xFF, in tenth of seconds}
Response	1 byte: {(Bit 0: Success)}

Code ID	0x46-0x47
Description	Get Capture Frame Format
Payload	None
Response	1 byte {0 = Baracoda, 1 = Baracoda + ACK, 2 = Raw}

Code ID	0x48-0x49
Description	Set Capture Frame Format
Payload	1 byte {0 = Baracoda, 1 = Baracoda + ACK, 2 = Raw}
Response	1 byte: {Bit0: Success}

Code ID	0x50-0x51
Description	Lock/Unlock Data capture module (ie : RFIDdecoder board)
Payload	1 byte : {0 = Unlock, 1 = Lock}
Response	1 byte : {Bit 0 : Success}

Code ID	0x56-0x57
Description	Get/Set In charge behavior
Payload	Get : None Set : 1 byte : {shutdown timeout (0 = leave current timeouts (default), 255 = infinity)}
Response	Get : 1 byte : {shutdown timeout (0 = leave current timeouts (default), 255 = infinity)} Set : 1 byte : {Bit 0 : Success}
Remarks	When scanner in charge, the shutdown timers can be modified

Code ID	0x5E-0x5F
Description	Bluetooth Commands
Payload	{Code ID} "Parameters"
Response	If the device responds: {Code ID} "Response" Else: {0}

Bluetooth specific commands from the Platform2 Bluetooth communication protocol are to be framed within the payload of this message.

1.1.5. Scanner messages

Code ID	0x60-0x61
Description	Get Scanner Status
Payload	None
Response	2 bytes: {(Bit 7: Upgrading) (Bit 1: Docked) (Bit 0: Charging)} {Battery level, 0 to 100}

Code ID	0x62-0x63
Description	Get Operating Mode
Payload	None
Response	1 byte: {Bit 0 = 0:real time, Bit 0 = 1: batch} {(Bit 7: limited)}
Remarks	“limited” means barcode buffer = 0 when in real time, no data loss mode and disconnected

Code ID	0x64-0x65
Description	Set Operating Mode
Payload	1 byte: {Bit 0 = 0: real time, Bit 0 = 1: batch} If real time mode is set : {(Bit 7: limited)(Bit 6: ACK beep) (Bit 5: no ACK beep)} NOTE : the ACK beep enable / disable is only effective when Capture Frame Format is “Baracoda + ACK”
Response	1 byte: {Bit 0: Success}
Remarks	Batch mode is not available for D-Fly scanner

Code ID	0x66-0x67
Description	Get Shutdown Timers
Payload	None
Response	2 bytes: {Number of minutes before shutdown when connected, 1 to 0xFE, 0xFF = infinity} {Number of minutes before shutdown when disconnected, 1 to 0xFE, 0xFF = infinity}

Code ID	0x68-0x69
Description	Set Shutdown Timers
Payload	2 bytes: {Number of minutes before shutdown when connected, 1 to 0xFE, 0xFF = infinity}

	{Number of minutes before shutdown when disconnected, 1 to 0xFE, 0xFF = infinity}
Response	1 byte: {Bit 0: Success}

Code ID	0x6A-0x6B
Description	Get RTC time
Payload	None
Response	6 bytes: {YY}{MM}{DD}{HH}{MM}{SS}

Code ID	0x6C-0x6D
Description	Set RTC time
Payload	6 bytes: {YY}{MM}{DD}{HH}{MM}{SS}
Response	1 byte: {Bit 0: Success}

Code ID	0x74-0x75
Description	Restore defaults settings
Payload	None
Response	1 byte: {Bit 0: Success}
Remarks	External Flash memory is also erased

Code ID	0x76-0x77
Description	Get Product Version
Payload	None
Response	x bytes : «Baracoda RRT...» for TagRunners product

Code ID	0x78-0x79
Description	Get Switching On Delay
Payload	None
Response	1 byte : {1 = 0 second, 2 = 1 second, 3 = 2 seconds}

Code ID	0x7A-0x7B
Description	Set Switching On Delay
Payload	1 byte : {1 = 0 second, 2 = 1 second, 3 = 2 seconds}
Response	1 byte : {Bit 0 :Success}

1.1.6. User Interface messages

LED 1 : left LED

LED 0 : right LED

Code ID	0x80-0x81
Description	Get MMI Descriptor
Payload	None
Response	2 bytes: {(Bit 6: Blue LED 1) (Bit 5: Red LED 1) (Bit 4: Green LED 1) (Bit 2: Blue LED 0) (Bit 1: Red LED 0) (Bit 0: Green LED 0)} {(Bit 0: Buzzer)}

Code ID	0x82-0x83
Description	Get MMI Mode
Payload	None
Response	1 byte: {(Bit 1: Buzzer Enabled) (Bit 0: LEDs enabled)}

Code ID	0x84-0x85
Description	Set MMI Mode
Payload	1 byte: {(Bit 1: Buzzer Enabled) (Bit 0: LEDs enabled) (Bit 7 = 0: Buzzer config select, =1: Buzzer config deselect)(Bit 6 = 0: leds config select, =1: leds config deselect)}
Response	1 byte: {(Bit 0: Success)}

Code ID	0x86-0x87
Description	Get MMI Signal (User interface)
Payload	1 byte: {Signal number, 0 - 3}
Response	(1 + 3n) bytes: {Number of steps, 0 - 4} For each step: {(Bit 6: Blue LED 1) (Bit 5: Red LED 1) (Bit 4: Green LED 1) (Bit 2: Blue LED 0) (Bit 1: Red LED 0) (Bit 0: Green LED 0)} {Buzzer frequency, 0 – 0xFF * 50Hz = 0 – 12750Hz} {Delay until next step, in tenth of seconds}

Code ID	0x88-0x89
Description	Set MMI Signal
Payload	(2 + 3n) bytes {Signal number, 0 - 3} {Number of steps, 0 - 4} For each step: {{Bit 6: Blue LED 1} (Bit 5: Red LED 1) (Bit 4: Green LED 1) (Bit 2: Blue LED 0) (Bit 1: Red LED 0) (Bit 0: Green LED 0)} {Buzzer frequency, 0 – 0xFF * 50Hz = 0 – 12750Hz} {Delay until next step, in tenth of seconds}
Response	1 byte: {{Bit 0: Success}}

Code ID	0x8A-0x8B
Description	Play Signal
Payload	2 bytes: {Signal number, 0 - 3} {Number of loops, 0 – 0xFE, 0xFF = infinity}
Response	1 byte: {{Bit 0: Success}}

Code ID	0x8C-0x8D
Description	Stop Signal
Payload	1 byte: {Signal number, 0 – 3}
Response	1 byte: {{Bit 0: Success}}

The list of MMI signals is:

IHM_SIGNAL_CAPTURE_READ : 0
 IHM_SIGNAL_CAPTURE_ACK : 1
 IHM_SIGNAL_CAPTURE_NAK : 2
 IHM_SIGNAL_CAPTURE_LOST : 3
 IHM_SIGNAL_USER_DEFINED : 13

Code ID	0x92-0x93
Description	Get Beeps mode
Payload	
Response	1 byte: {{Bit 0: Beep level 0=low, 1=high} (Bit 1: Read beep) (Bit 2: ACK beep)}

Code ID	0x94-0x95
Description	Set Beeps mode
Payload	<p>1 byte:</p> <p>{{(Bit 0: Beep level; 0=low, 1=high)</p> <p>(Bit 1: Read beep)</p> <p>(Bit 2: ACK beep)</p> <p>(Bit 7 = 0: ACK beep config select, =1: ACK beep config deselect)</p> <p>(Bit 6 = 0: Read beep config select, =1: Read beep config deselect)</p> <p>(Bit 5 = 0: Beep level config select, =1: Beep level config deselect)}}}</p>
Response	<p>1 byte:</p> <p>{{(Bit 0: Success)}}}</p>

I.1.7. Miscellaneous messages

Code ID	0xC2-0xC3
Description	Get/Set DataPrefix
Payload	Get : None Set : 1-33 bytes: { DataPrefix length} "DataPrefix String"
Response	Get : 1-33 bytes: { DataPrefix length} "DataPrefix String" Set : 1 byte: {Success?}
Remarks	Configure value of 'DataPrefix'

Code ID	0xC4-0xC5
Description	Get/Set DataSuffix
Payload	Get : None Set : 1-33 bytes: { DataSuffix length} "DataSuffix String"
Response	Get : 1-33 bytes: { DataSuffix length} "DataSuffix String" Set : 1 byte: {Success?}
Remarks	Configure value of 'DataSuffix'

Code ID	0xC6-0xC7
Description	Get/Set Data Format 2
Payload	Get : None Set : 1 byte: {{Bit 5 = 0: DataPrefix config select, =1: DataPrefix config deselect} (Bit 4 = 0: DataSuffix config select, =1: DataSuffix config deselect) (Bit 1: DataPrefix) (Bit 0: DataSuffix)
Response	Get : 1 byte: {(Bit 1: DataPrefix) (Bit 0: DataSuffix)} Set : 1 byte: {(Bit 0:Success)}
Remarks	Enable, disable the adding of 'Data Prefix' and 'Data Suffix'

Code ID	0xC8-0xC9
Description	Get / Set Captured data (TagID) length
Payload	Get : None Set : 2 bytes {authorized RFID data length. 0 = disabled}
Response	Get : 2 bytes {authorized RFID data length. 0 = disabled} Set : 1 byte {{Bit 0:Success}}

Code ID	0xCA-0xCB
Description	Get and erase stored Captured data with no data loss mode
Payload	None
Response	1 byte {{Bit 0:Success}}

Code ID	0xCC-0xCD
Description	Reset modes
Payload	None = restore defaults, keep link keys, reboot scanner 1byte : 0 = restore defaults, keep link keys, reboot scanner 1 = switch off scanner (no restoring defaults) 2 = reboot scanner (no restoring defaults)
Response	1 byte {{Bit 0:Success}}

Code ID	0xCE-0xCF
Description	Batch upload commands
Payload	{Code ID} "Parameters" (cf. below)
Response	{Code ID} "Response"

The UPLOAD Code IDs are:

Code ID	0
Description	Launch upload process (typically used only for the upload barcode)
Payload	1 byte: {0 mandatory}
Response	1 byte: {Bit 0: Success}

Code ID	1
Description	Ready to start upload (Scanner to host message)
Payload	3 bytes : {0 mandatory} { number of elements to be uploaded MSB }

	{ number of elements to be uploaded LSB }
Response	None

Code ID	2
Description	Start uploading data
Payload	1 byte: {0 mandatory}
Response	1 byte: {Bit 0: Success}

Code ID	3
Description	RESERVED
Payload	N/A
Response	N/A

Code ID	4
Description	Set upload status and end process
Payload	2 bytes : {0 mandatory} {1 : upload successful, data can be erased from the scanner 0 : upload failed, do not erase data}
Response	1 byte: {Bit 0: Success}

Code ID	0xD0-0xD1
Description	Get Serial Number
Payload	Get: None
Response	Get : 2-15 bytes: { Serial Number string length } [S/N (1-14 bytes)]

Code ID	0xD2-0xD3
Description	Get/Set Anti duplicate scans
Payload	Get : None Set : 1 byte {0 = disabled 1 = no consecutive duplicate scans + error signal 2 = no consecutive duplicate scans + no decoding}
Response	Get : 1 byte {0 = disabled 1 = no consecutive duplicate scans + error signal 2 = no consecutive duplicate scans + no decoding } Set : 1 byte {{Bit 0:Success}}

Comments	The comparison will be made over the 32 first characters of the barcodes only.
----------	--

Code ID	0xD4-0xD5
Description	Restore last batch
Payload	None or 1 byte (optional): {1 = upload data after retrieving}
Response	1 byte: {{Bit 0:Success}}
Comments	This is only available if no new data capture has been made.

Code ID	0xD8-0xD9
Description	Enable remote trigger
Payload	None : use default 5s timeout 1 byte : {timeout (s)}
Response	1 byte {{Bit 0:Success}}

Code ID	0xDE-0xDF
Description	RFID commands
Payload	{Code ID} "Parameters"
Response	{Code ID} "Response"

RFID specific commands from the Platform2 RFID communication protocol are to be framed within the payload of this message.

1.1.8. Capture messages

Code ID	0xE2-0xE3
Description	Get Mode
Payload	None
Response	1 byte: {0 = trigger, 1 = smart autoscan, 2 = disabled, 3= autoscan }

Code ID	0xE4-0xE5
Description	Set Mode
Payload	1 byte OR 2 bytes if aiming trigger scan mode {0 = trigger, 1 = smart autoscan, 2 = disabled, 3= autoscan, 4=aiming trigger scan} {aiming trigger scan mode timeout value in second}
Response	1 byte: {{(Bit 0: Success)}}

Code ID	0xE6-0xE7
Description	Get Data Format
Payload	None
Response	1 byte: {{(Bit 2:Timestamp) (Bit 1: Capture Prefix) (Bit 0: Capture Suffix)}}

Code ID	0xE8-0xE9
Description	Set Data Format
Payload	1 byte: {{(Bit 7 = 0: Timestamp config select, =1: Timestamp config deselect) (Bit 6 = 0: Capture Prefix config select, =1: Capture Prefix config deselect) (Bit 5 = 0: Capture Suffix config select, =1: Capture Suffix config deselect) (Bit 2:Timestamp) (Bit 1: Capture Prefix) (Bit 0: Capture Suffix) }
Response	1 byte: {{(Bit 0:Success)}}

Code ID	0xEA-0xEB
Description	Get Capture Prefix
Payload	None
Response	1-33 bytes: { Capture Prefix length} "Capture Prefix String"

Code ID	0xEC-0xED
Description	Set Capture Prefix
Payload	1-33 bytes: { Capture Prefix length} "Capture Prefix String"
Response	1 byte: {{Bit 0:Success}}

Code ID	0xEE-0xEF
Description	Get Capture Suffix
Payload	None
Response	1-33 bytes: { Capture Suffix length} "Capture Suffix String"

Code ID	0xF0-0xF1
Description	Set Capture Suffix
Payload	1-33 bytes: { Capture Suffix length} "Capture Suffix String"
Response	1 byte: {{Bit 0:Success}}

Code ID	0xF4-0xF5
Description	Set Timestamp
Payload	6 bytes: {YY}{MM}{DD}{HH}{MM}{SS}
Response	1 byte: {Bit 0: Success}

Code ID	0xF8-0xF9
Description	Get stored Captured data count
Payload	None
Response	2 bytes: {Stored captured data (ie TagID) count [15:8]} {Stored captured data (ie TagID) count [7:0]}

Code ID	0xFC-0xFD
Description	Erase stored captured data (ie :TagID)
Payload	None
Response	1 byte: {{Bit 0:Success}}

APPENDIX 1: Bluetooth Protocol

The configuration frames are as follows:

Header: 1 Byte	Length: 2 Bytes (MSB, LSB)	Payload: 0 to 65535 Bytes.
----------------	----------------------------	----------------------------

Commands

Command	Set Pin Code
Header	0x01 (flash only)
Length	xx xx (new pin size)
Payload	N digits PIN. (Default "0000")
Response	0x01 00 01 01 if done 0x01 00 01 00 if not
Remark	Max Pin length=16

Command	Get Pin Code
Header	0x07
Length	00 00
Payload	N digits PIN. (Défaut "0000")
Response	0x07 {PinCode size} {Pincode}
Remark	

Command	Set Name
Header	0x02 (flash only)
Length	xx xx
Payload	(new name size)
Response	New name 0x02 00 01 01 if done 0x02 00 01 00 if not
Remark	(Names up to 248 Bytes)

Command	Get Name
Header	0x08
Length	00 00
Payload	
Response	0x08 {name size} {name}
Remark	Name size: 2 Bytes MSB, LSB Names up to 248 Bytes

Command	Set Mode
Header	0x03 (flash only)
Length	00 01
Payload	0x01 if MASTER, 0x00 if SLAVE
Response	0x03 00 01 01 if done

	0x03 00 01 00 if not
Command	Set Mode
Header	0x03 (flash only)
Length	00 02
Payload	0x01 if MASTER, 0x00 if SLAVE, [Role switch]
Response	0x03 00 01 01 if done 0x03 00 01 00 if not

When in Master, the Module connects to the address specified by Set REMOTE BDA or to the last paired device.

The real MASTER in a Bluetooth piconet is the device which manages the clock used for the frequency hopping. We used to speak about MASTER too for devices which create the connection (that's true if you do not switch the clock role)

A device with a slave BT clock role is unable to synchronize more than one master clock. If more than one SmartModule needs to connect to the same other device (PC, Access Point...) you will need to switch the clock role to allow the slave to be connected to more than one master. Note that most of the BT access point already generates the BT clock role switch when a master device creates a connection.

Command	Get Mode
Header	0x04
Length	00 00
Payload	
Response	0x04 00 02 {Mode (1byte) Switch role (1byte)}
Remark	0x01 if MASTER, 0x00 if SLAVE 0x01 if want automatic switch role, 0x00 otherwise

Command	Set Remote BDA (Used by Master Mode of the SM)
Header	0x05
Length	00 06
Payload	BDA(ex:0x00,0x02,0xC3,0x21, 0xDE,0xFA)
Response	0x05 00 01 01 if done 0x05 00 01 00 if not
Remark	If The SM is set to Master (using Set MODE command), the SM use this Address to connect to.

Command	Get Remote BDA
Header	0x06
Length	00 00
Payload	
Response	0x06 00 06 {6 bytes of BDA}

Remark	
--------	--

Command	Get Bluetooth Version
Header	0x76
Length	00 00
Payload	
Response	0x76, x, x, {version string }

Command	Restore Factory Settings
Header	
Length	
Payload	('R', 's', 't')
Response	

Command	Get inquiry scan timeout
Header	0x27
Length	00 00
Payload	
Response	0x27 00 04 [Inquiry Interval (MSB) Inquiry Interval (LSB) Inquiry Window (MSB) Inquiry Window (LSB)]
Remark	Inquiry Interval and Inquiry Window are in number of Bluetooth slots) (1 slot = 0.625 ms)

Command	Set inquiry scan timeout
Header	0x26
Length	00 04
Payload	Inquiry Interval (MSB) Inquiry Interval (LSB) Inquiry Window (MSB) Inquiry Window (LSB)] (default 0xC80, 0x18)
Response	0x26 00 01 01 if done 0x26 00 01 00 if not
Remarks	Inquiry Scan TimeOuts are used by the Module to answer to Inquiries. So, if you set both values to 0, the Module will not be discoverable.

Command	Set page scan timeout
Header	0x24
Length	00 04
Payload	[Page Interval (MSB) Page Interval (LSB) Page Window (MSB) Page Window (LSB)] (default 0x320, 0xb0)
Response	0x24 00 01 01 if done 0x24 00 01 00 if not
Remark	Page Scan TimeOuts are used by the Module to answer to Connect Inquiries. So, if you set both values to 0, the Module will not be Connectable.

Command	Get page scan timeout
Header	0x25
Length	00 00
Payload	
Response	0x25 00 04 [Page Interval (MSB) Page Interval (LSB) Page Window (MSB) Page Window (LSB)]
Remark	Page Scan Interval and Page Scan Window are in number of Bluetooth slots) (1 slot = 0.625 ms)

Typical values are:

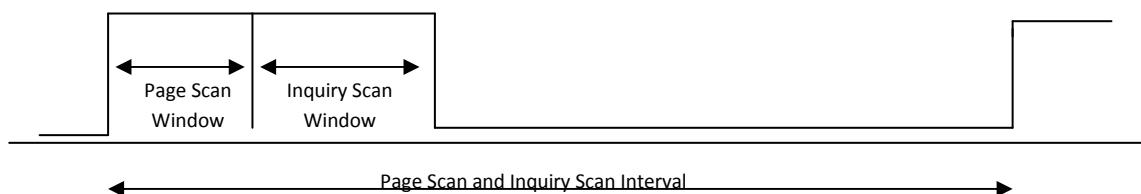
Full power:

- Inquiry Interval = 0x400
- Inquiry Window = 0x200
- Page Scan Interval = 0x400
- PageScan Window = 0x200

Low power:

- Inquiry Interval = 0x320
- Inquiry Window = 0x80
- Page Scan Interval = 0x320
- PageScan Window = 0x80

Here is how these values change the consumption of the Module:



Command	Set sniff
Header	0x09
Length	00 04
Payload	[MSB of MinSniff interval, LSB of MinSniff interval, MSB of MaxSniff interval, LSB of MaxSniff interval]
Response	0x09 00 01 01 if done 0x09 00 01 00 if not
Remark	

Command	Set sniff (advanced)
Header	0x09
Length	00 08
Payload	[MSB of MinSniff interval, LSB of MinSniff interval, MSB of MaxSniff interval, LSB of MaxSniff interval, Sniff Attempts MSB, Sniff attempts LSB, Sniff timeout MSB, Sniff timeout LSB]
Response	0x09 00 01 01 if done 0x09 00 01 00 if not
Remark	

Command	Get Sniff
Header	0x10
Length	00 00
Payload	
Response	0x10 00 08 [MSB of MinSniff interval, LSB of MinSniff interval, MSB of MaxSniff interval, LSB of MaxSniff interval, Sniff Attempts MSB, Sniff attempts LSB, Sniff timeout MSB, Sniff timeout LSB]
Remark	When setting only MinSniff and MaxSniff values, the default value 0x08 will be used for Sniff attempts and Sniff timeout.

Typical values are:

Full speed (full power)

MinSniff = 0

MaxSniff = 0

Very Low Power (low speed): (sniff of 500ms Only are accepted. If the remote device does not support sniffs of 500ms, no sniff will be used)

MinSniff = 0x0320

MaxSniff = 0x0320

Very Low Power (low speed): (sniff between 250ms to 500ms are accepted. No sniff will be used if the remote device does not support any sniff values in this specified range)

MinSniff = 0x0160

MaxSniff = 0x0320

Low Power (medium speed):

MinSniff = 0x0050

MaxSniff = 0x00F0

Attempt = 0x0008

Timeout = 0x0030

MaxSniff and MinSniff are only used for sniff negotiation between the Smart Module and the other BT device. If both sides allow sniff value MaxSniff, then MaxSniff will be used. If the other side does not accept Sniff values MinSniff to MaxSniff, no sniff will be used.

Values are in number of Bluetooth slots (1 slot = 625µs)

Set MinSniff and MaxSniff to 0 to disable Sniff.

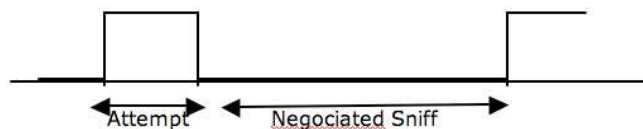
MinSniff must be inferior to MaxSniff.

Possible values for MinSniff and MaxSniff are 0x12 to 0xFFFF.

Sniff attempts of 0 is not allowed.

Warning: Setting MaxSniff to 0xFF means a sniff period of 40s! You will have very very low data rate.

Note: This setting takes effect immediately.



For further details on Sniff values, see the Bluetooth spec 1.1, chapter 10.8.2

Command	Get link timeout
Header	0x18
Length	00 00
Payload	
Response	0x18 00 02 [MSB of link Tmo, LSB of link Tmo]
Remark	

Command	Set link timeout
Header	0x19
Length	00 02
Payload	[MSB of link Tmo, LSB of link Tmo]
Response	0x19 00 01 01 if done
Remark	The link Time Out is a multiple of 625µsec (625µs = 1 Bluetooth slot) (default 0x7D00 (=20s))

This Timeout is use by the Link Manager to monitor the Bluetooth Link. If there is no answer from the other device after this timeout, the Link Manager assumes that we are disconnected. By default, this value is set to 20 seconds. You can go down to 1s, but then you can have disconnection even if it's only a temporary perturbation.

This value will take effect at the next connection.

Command	Get Security Mode
Header	0x20
Length	00 00
Payload	
Response	0x20 00 01 01 if secured 0x20 00 01 00 if non secured
Remark	

Command	Set Security Mode
Header	0x21
Length	00 {size}
Payload	{00 non secured, 01 secured} {PIN CODE (default 01)}
Response	0x21 00 01 01 if done, 0x21 00 01 00 if not
Remark	Size=PINCODE size + 1 For example : 0x21 00 05 00 30 30 30 30 to disable security

Command	Get Bluetooth class device
Header	0x30
Length	00 00
Payload	
Response	0x30 00 04 [Class of device]
Remark	See the Bluetooth specification for more details

Command	Set Bluetooth class device
Header	0x31
Length	00 04
Payload	[Class of Device (4 bytes, MSB->LSB)] (default 0x500)
Response	0x31 00 01 01 if done 0x31 00 01 00 if not

Typical Bluetooth class of device:

Peripheral	0x000500 (default)
Undefined	0x001F00
Phone	0x502204
Computer	0x120104
PDA	0x100114
Access Point	0x120320

Command	Set Remote rfcmm channel
Header	0x36
Length	00 01
Payload	[channel (1byte)]
Response	0x36 00 01 01 if done 0x36 00 01 00 if not
Remark	

Command	Get Remote rfcmm channel
Header	0x37
Length	00 00
Payload	
Response	0x37 00 01 [channel]
Remark	

If “channel” is not zero, the Module will directly try to connect (if in master mode) to the specified rfcmm channel.

Setting the channel to zero will force the Module to connect (if in master mode) to the first specified Remote Service UUID (by default SPP).

The services in the Module are all set to channel 1.

Command	Set Target Service UUID
Header	0x38
Length	00 02
Payload	[UUID (2 Bytes)] (default 0x1101)
Response	0x38 00 01 01 if done 0x38 00 01 00 if not
Remark	Try to connect to this remote service.

Command	Get Target Service UUID
Header	0x39
Length	00 00

Payload	
Response	0x39 00 02 [UUID]
Remark	Try to connect to this remote service.

Here are some service UUID:

SPP	0x1101
DUN	0x1103
FAX	0x1102

You can get more UUIDs by reading the Bluetooth spec.

Command	Get Encryption Mode
Header	0x40
Length	00 00
Payload	
Response	0x40 00 01 [encryption]
Remark	

Command	Set Encryption Mode
Header	0x41 (flash only)
Length	00 01
Payload	[Encryption (1 byte)]
Response	0x41 00 01 01 if done 0x41 00 01 00 if not
Remark	Argument is: 0x01 to enable encryption, 0x00 to disable.

Command	Get local Bluetooth Address
Header	0x43
Length	00 00
Payload	
Response	0x43 00 06 {6 Bytes (BD_address MSB, ..., LSB)}
Remark	

APPENDIX 2: RFID communication protocol

Introduction

Generalities

Platform2 is supporting a communication interface with an RFID daughter board. Using a UART link, the communication is possible thanks to a communication protocol described in this document.

Generic packet

All the frames described in this document are formatted as shown:

Code ID	Length	Payload
1 Byte	2 Bytes	N Bytes

Commands with codeIDs between 0x00 and 0x7F are configuration commands

Commands with codeIDs between 0x80 and 0xFF are communication commands

Communication protocol

Configuration messages

Code ID	0x02
Description	Get/Set active protocols (Tagrunners only)
Payload	Get : None Set : 4 bytes (MSB first): [PROTOCOL_DESELECT_MASK_MSB] [PROTOCOL_DESELECT_MASK_LSB] [PROTOCOL_MASK_MSB] [PROTOCOL_MASK_LSB]
Response	Get : 2 bytes: Cf. below for bits significance Set : 1 = success

MASK format :

- Bit 0 (LSB): ISO/IEC 14443-A (or NXP Mifare)
- Bit 1 : ISO/IEC 14443-B
- Bit 2 : ISO/IEC 15693 (e.g. TI Tag-it or NXP ICODE-SLI)
- Bit 3 : NXP ICODE-1
- Bit 4 : Inside Contactless PicoTAG
- Bit 5 : S.T. MicroElectronics SR
- Bit 6 : ASK CTS256B/CTS512B
- Bit 7 : Calypso (Innovatron protocol)
- Bit 8 : EPC HF Version 2 (future feature)

- Bit 9 : ISO/IEC 15693 in FAST mode
- Bit 10 : RFU
- Bit 11 : RFU
- Bit 12 : RFU
- Bit 13 : RFU
- Bit 14 : RFU
- Bit 15 : RFU
- Bit 16 : RFU

For the deselect mask, a set bit masks its configuration.

Example :

The current active protocols mask value is 0xFFFF (all protocols enabled, default value)

We want to disable 14443-A and 14443-B, we also want to be sure 15693 is enabled. We want to keep the other settings as they are and we do not want to have to make a "get" before the "set".

- ➔ The 4 bytes for the set command should be : 0x00 0xF8 0x00 0x04
- ➔ The new active protocols mask value is 0xFFF4

Code ID	0x03
Description	Get/Set protocol ID transmission (Tagrunners only)
Payload	Get : None Set : 1 byte: 1 = enable, 0 = disable
Response	Get : 1 byte: 1 = enabled, 0 = disabled Set : 1 = success

If enabled, IDs are transmitted before the tag UID data:

ID string	Associated protocol
[A]	ISO/IEC 14443-A (or NXP Mifare)
[B]	ISO/IEC 14443-B
[C]	ISO/IEC 15693 (e.g. TI Tag-it or NXP ICODE-SLI)
[D]	NXP ICODE-1
[E]	Inside Contactless PicoTAG
[F]	S.T. MicroElectronics SR
[G]	ASK CTS256B/CTS512B
[H]	Calypso (Innovatron protocol)
[I]	EPC HF Version 2
[Z]	unknown

Code ID	0x04
Description	Get Mifare keys status (Mask) (Tagrunners only)
Payload	None
Response	4 bytes: Bits 0-15 : the A keys (bits 0-3 reserved)

Remarks	Bits 0-15 : the B keys (bits 0-3 reserved) A set bit means that a key has been loaded at this emplacement.
	For Mifare cards, a 6-byte security key is needed to access the data. The scanner is able to handle 16 keys for each of the 2 key sets (A and B defined in Mifare spec.) for each key set, the 4 first keys are reserved for default keys.

Code ID	0x05
Description	Load Mifare key(s) (Tagrunners only)
Payload	Restore default keys : None (erases all user defined keys) Load a key : 8 bytes 1B : {key set : 0 = A key, 1 = B key} 1B : {key number : 4 to 15} 6B : {the 6 bytes of the key}
Response	1 = success

Code ID	0x07
Description	Get/Set Protocol ID Prefix Suffix
Payload	Byte 1 : Protocol ID (0 – 9) (see below) Following bytes : List of commands and there parameters (see below)
Response	Byte 1 : 1 = success, 0 = failed If Get: Byte 2 : Protocol ID Following bytes : list of command and there data
Remarks	See command 0x83 for more details about the transparent mode

Command List:

RFID_CONFIG_TRAME_SET_PREFIX = 0x14
RFID_CONFIG_TRAME_SET_SUFFIX = 0x15
RFID_CONFIG_TRAME_GET_PREFIX = 0x16
RFID_CONFIG_TRAME_GET_SUFFIX = 0x17

Protocol ID:

ID	Associated protocol
0	ISO/IEC 14443-A (or NXP Mifare)
1	ISO/IEC 14443-B
2	ISO/IEC 15693 (e.g. TI Tag-it or NXP ICODE-SLI)
3	NXP ICODE-1
4	Inside Contactless PicoTAG
5	S.T. MicroElectronics SR
6	ASK CTS256B/CTS512B
7	Calypso (Innovatron protocol)
8	EPC HF Version 2
0xFF	All Protocol

Code ID	0x76
Description	Get version
Payload	None
Response	x bytes: “ProxRunners vx.xx” or “TagRunners vx.xx”

Communication messages

Code ID	0x80
Description	Find TAG and get ID (Tagrunners only)
Payload	0-3bytes: [optional 2 bytes] : {protocol mask} [optional 1 byte] : {timeout (s)}
Response	None if no tag detected Else x bytes: {Tag ID}

Code ID	0x81
Description	Read data from a TAG (Tagrunners only)
Payload	6 bytes: 2B : { protocol mask the mask format is the same as the one used in command 0x02. if 00 00 is given, then the default mask will be used (the one defined with command 0x02) } 1B : { timeout Timeout for the response. if 00 is given, then the default timeout will be used (5s) } 1B : { expected card type 00 : try any supported card 01 : only try to read mifare ultralight cards 02 : only try to read mifare 1K cards 03 : only try to read mifare 4K cards ETC. the complete list of card types codes is shown below } 2B : { start/stop sectors (pages) Start sector must be smaller or equal to stop sector number For all cards (tags), the first sector (or page, or block) is 0. For the stop sector, if the specified number is greater than the last sector available on the tag, the reading will occur until the last sector Examples : 00 00 : read first sector/page 00 FF : read the whole memory }
Response	x bytes: ⚠ only the reading status is encapsulated into the data frame (code ID, length, payload), before receiving this, the response can be sent : 2B : { protocol type RF protocol used by the read card (see mask format) Example : 00 01 = mifare tag } 1B : { UID (tag ID) length } xB : { UID } 1B : {Card type

00 : try any supported card
 01 : mifare ultralight card
 02 : mifare 1K card
 03 : mifare 4K card
 ETC. the complete list of card types codes is shown below }

For each read sector/page:

If Mifare Ultralight :

{1B : page number
 4B : data}

If Mifare 1K or 4K :

{1B : sector number
 1B : block number
 16B : data}

If ISO15693 card from NXP or TI :

{1B : block number
 4B : data}

If ISO15693 card from STMicroelectronics :

{1B : block number
 1B : data}

Code ID	0x82
Description	Write data into a TAG (Tagrunners only)
Payload	<p>7+ bytes:</p> <p>2B : { protocol mask the mask format is the same as the one used in command 0x02. if 00 00 is given, then the default mask will be used (the one defined with command 0x02) }</p> <p>1B : { timeout Timeout for the response. if 00 is given, then the default timeout will be used (5s) }</p> <p>1B : { card type 00 : try any supported card 01 : write in mifare ultralight cards 02 : write in mifare 1K cards 03 : write in mifare 4K cards ETC. the complete list of card types codes is shown below }</p> <p>1B : { start sector Number of first sector/page to start writing at}</p> <p>2B : { length (MAX = 256) Number of bytes to write /!\ whole sectors will be written. If the specified number of bytes does not complete one sector, the remaining space will be erased. }</p> <p>xB : { data }</p>
Response	1 byte: { 1 = success }

For the two above commands, the complete list of card types codes for read/write compatibility:

Code	Protocol	Manufacturer	Card type	Compatibility
0x01	ISO 14443-A	NXP	Mifare Ultralight	YES
0x02	ISO 14443-A	NXP	Mifare 1K	YES
0x03	ISO 14443-A	NXP	Mifare 4K	YES
0x04	ISO 14443-A	NXP	Desfire	NO
0x05	ISO 14443-A	Others (e.g. ASK)	_	NO
0x06	ISO 14443-B	Any	_	NO
0x07	ISO 15693	NXP	I-CODE SLI	YES
0x08	ISO 15693	NXP	I-CODE SLI-L	YES
0x09	ISO 15693	NXP	I-CODE SLI-S	YES
0x0a	ISO 15693	NXP	Others	NO
0x0b	ISO 15693	Texas Instruments	Tag-it HF-I Plus inlay	YES
0x0c	ISO 15693	Texas Instruments	Tag-it HF-I Plus chip	YES
0x0d	ISO 15693	Texas Instruments	Tag-it HF-I Standard chip/inlay	YES
0x0e	ISO 15693	Texas Instruments	Tag-it HF-I Pro chip/inlay	YES
0x0f	ISO 15693	Texas Instruments	Others	NO
0x10	ISO 15693	STMicroelectronics	LRI64	YES
0x11	ISO 15693	STMicroelectronics	Others	NO
0x12	ISO 15693	Legic	Any	NO
0x13	ISO 15693	Others	_	NO

About the known cards memory constitution :

Card type	Memory constitution
Mifare Ultralight	64 Bytes: 16 4-byte pages
Mifare 1K	768 Bytes: 16 48-byte sectors each constituted of 3 16-byte blocks
Mifare 4K	3456 Bytes: 32 48-byte sectors each constituted of 3 16-byte blocks 8 240-byte sectors each constituted of 15 16-byte blocks
I-CODE SLI	112 Bytes: 28 4-byte blocks
I-CODE SLI-L	32 Bytes: 8 4-byte blocks
I-CODE SLI-S	160 Bytes: 40 4-byte blocks
Tag-it HF-I Plus inlay	256 Bytes: 64 4-byte blocks
Tag-it HF-I Plus chip	256 Bytes: 64 4-byte blocks
Tag-it HF-I Standard chip/inlay	44 Bytes: 11 4-byte blocks

	(blocks 8-9 → UID, block 10 → AFI)
Tag-it HF-I Pro chip/inlay	48 Bytes: 12 4-byte blocks (blocks 8-9 → UID, block 10 → AFI, block 11 → password)
LRI64	14 Bytes: 14 1-byte blocks (blocks 0-7 → UID, block 8 → AFI, block 9 → DSFID)

APPENDIX 3: RFID tag data read/write examples

(for BlackBerry, Symbian & Java platforms)

1. Introduction

1.1 Generalities

This document describes how developers can read and write RFID tag data with the Baracoda TagRunners (BRRT) and Baracoda DualRunners (BDR) readers. It can be used on the following platforms (SDKs):

- Blackberry
- Java J2ME phones
- Symbian

Please note that the Full SDK for PC and PDA has a set of commands (ReadTagData, WriteTagData) that can be used directly.

<http://www.baracoda.com>

2. Mifare Ultralight tags

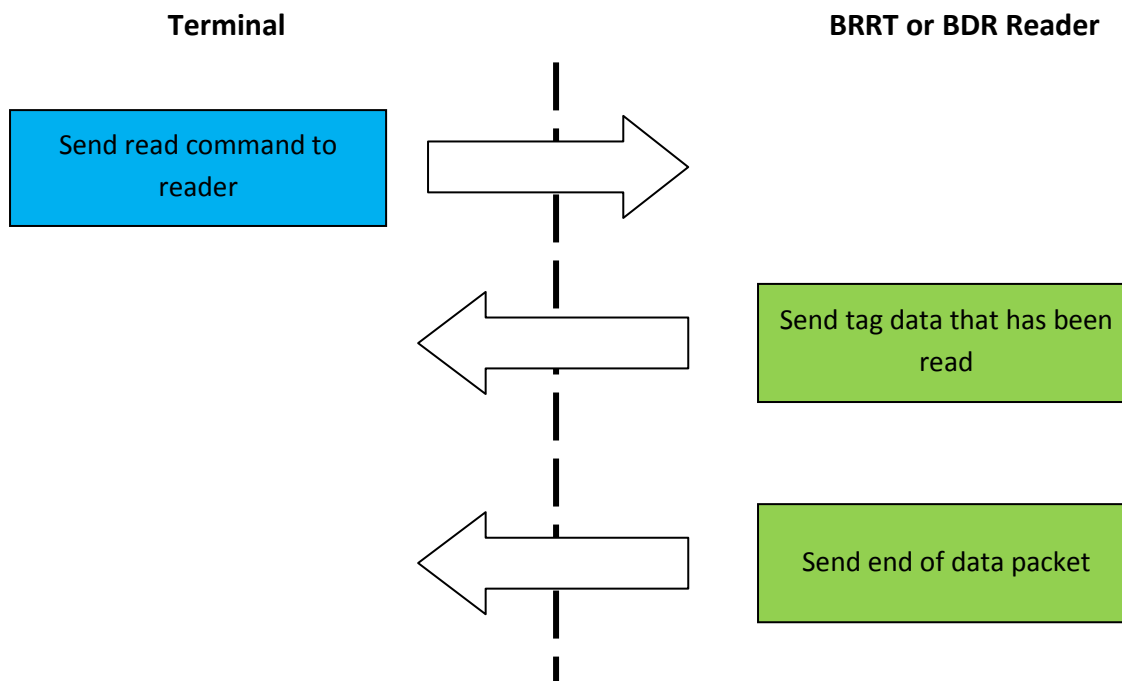
2.1 Tag memory structure

Mifare Ultralight tags have 64 bytes of memory divided into 16 (sixteen) 4-bytes long pages.

2.2 Reading tag data

a) Communication diagram

The following diagram shows what needs to be done in order to read the contents of a Mifare Ultralight tag:



b) Data sent from the terminal to the reader

First of all, the terminal needs to send to the reader a Read data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x81. In this case, the format of the command will be as follows (in hexadecimal):

DE 00 07 81 00 00 1E 01 00 FF

where:

- **DE**: prefix
- **00 07**: data length (7 bytes will follow)
- **81**: command ID
- **00 00**: protocol mask (it's best to specify 00 00 for Mifare cards)
- **1E**: timeout for the response (30s)
- **01**: card type (Mifare Ultralight)
- **00 FF**: start/stop sectors/pages (**00 FF** means read the whole memory)

c) Reader's response (successful read)

If the read operation has been successful, the reader will send the tag ID, the tag data and then an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

00 01 07 xx xx xx xx xx xx xx 01 00 aa bb cc dd 01 ee ff gg hh 02 ... 0F ii jj kk ll DE 00 02 81 01,
where

- **00 01:** tag protocol (Mifare Ultralight)
- **07:** tag ID length (7 bytes)
- **xx xx xx xx xx xx xx:** tag ID (tag ID is read-only)
- **01:** card type (01 means Mifare Ultralight)
- **00:** page id (0)
- **aa bb cc dd:** 4 bytes of page 0
- **01:** page id (1)
- **ee ff gg hh:** 4 bytes of page 1
- etc...
- **0F:** page id (15)
- **ii jj kk ll:** 4 bytes of page 15
- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

Please note that the dump does not show all 16 pages of the tag data. They all have an identical structure: page id followed by 4 (four) bytes of data. The last byte of the end of data packet is equal to 1 (one), which means that the read operation has been successful.

d) Reader's response (read failure)

If the reader cannot read the tag data, the reader's response will be the following:

[xx xx yy zz zz zz zz vv] DE 00 02 81 00,

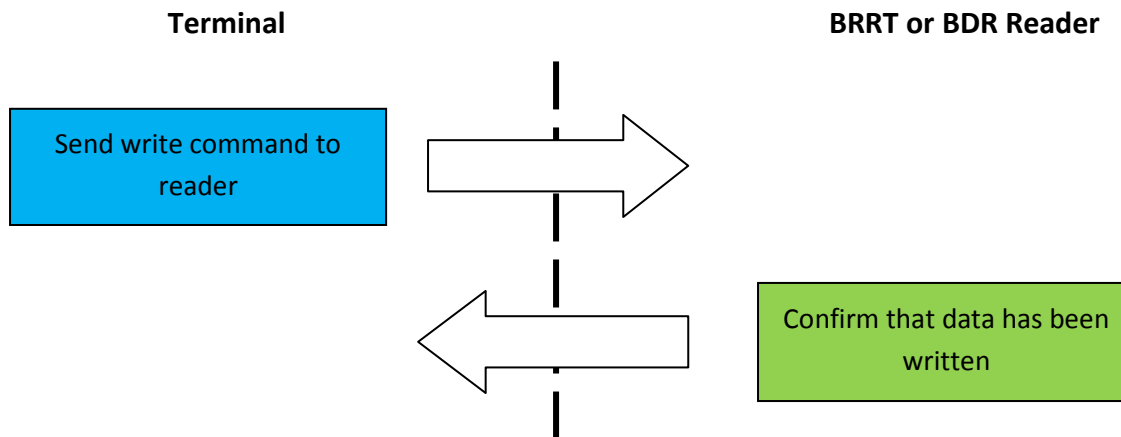
where:

- the first part of the response is optional and will be sent only if a tag of a different protocol was read
 - o **xx xx:** tag protocol
 - o **yy:** tag ID length
 - o **zz zz zz zz:** tag ID (**yy** bytes)
 - o **vv:** card type
- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

2.3 Writing tag data

a) Communication diagram

The following diagram shows what needs to be done in order to write the contents of a Mifare Ultralight tag:



b) Data sent from the terminal to the reader

For every page that needs to be modified, the terminal needs to send a write command to the reader. It is recommended to verify what parts of the tag data have been modified and send only the differing pages.

In order to write a page, the terminal needs to send to the reader a Write data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x82. In the example the user intends to modify page 7 (seven), so the format of the command will be as follows (in hexadecimal):

DE 00 0C 82 00 00 00 01 07 00 04 xx xx xx xx

where:

- **DE**: prefix
- **00 0C**: data length (12 bytes will follow)
- **82**: command ID (write tag data)
- **00 00**: protocol mask (it's best to specify 00 00 for Mifare cards)
- **00**: timeout for the response (0 means use default = 5s)
- **01**: card type (Mifare Ultralight)
- **07**: start page (7)
- **00 04**: data length (4 bytes)
- **xx xx xx xx**: data to be written in page 7

c) Reader's response (successful write)

If the write operation has been successful, the reader will send an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

DE 00 02 81 01,

where

- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

d) Reader's response (write failure)

If the reader cannot write the tag data, the reader's response will be the following:

DE 00 02 81 00,

where:

- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

Please note that most Mifare Ultralight cards have pages that are read-only. The developer should not try to modify those fields as the results are unpredictable (e.g. the reader may report a success even though the write operation failed). Please refer to the used RFID tag manufacturer specification for more details.

3. Mifare 1K tags

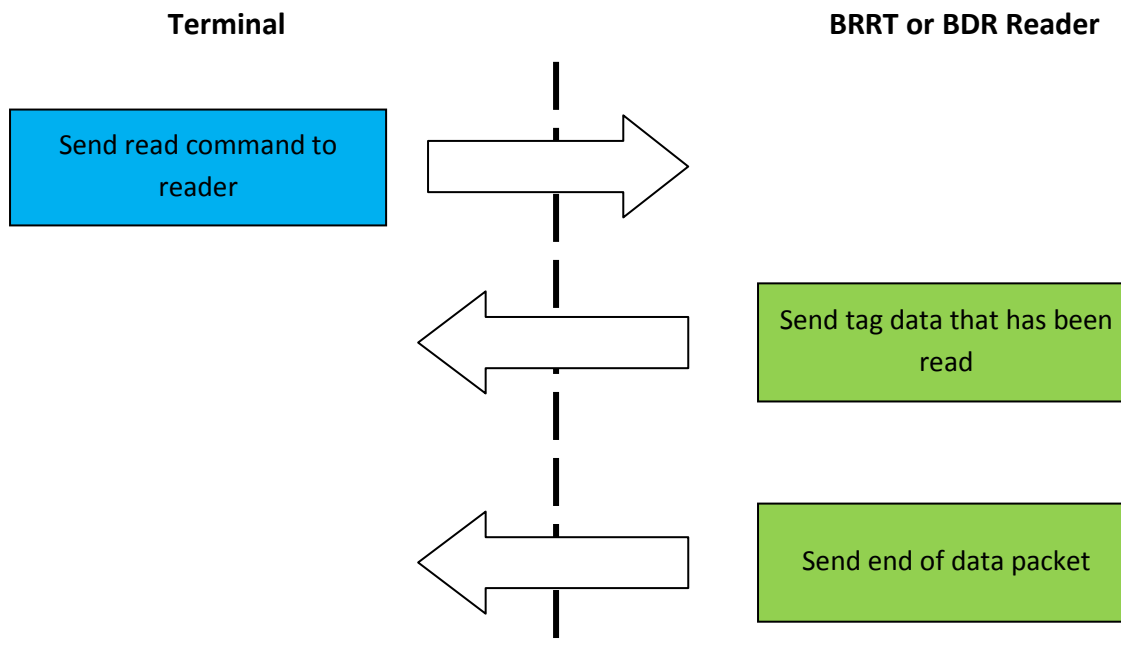
3.1 Tag memory structure

Mifare 1K tags have 768 bytes of memory divided into 16 (sixteen) 48-bytes long sectors. Every sector is divided into 3 (three) 16-bytes long blocks

3.2 Reading tag data

a) Communication diagram

The following diagram shows what needs to be done in order to read the contents of a Mifare 1K tag:



b) Data sent from the terminal to the reader

First of all, the terminal needs to send to the reader a Read data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x81. In this case, the format of the command will be as follows (in hexadecimal):

DE 00 07 81 00 00 1E 02 00 FF

where:

- **DE**: prefix
- **00 07**: data length (7 bytes will follow)
- **81**: command ID
- **00 00**: protocol mask (it's best to specify 00 00 for Mifare cards)
- **1E**: timeout for the response (30s)
- **02**: card type (02 means Mifare 1K)
- **00 FF**: start/stop sectors/pages (**00 FF** means read the whole memory)

c) Reader's response (successful read)

If the read operation has been successful, the reader will send the tag ID, the tag data and then an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

00 01 04 xx xx xx xx 02 00 00 aa aa .. aa 00 01 bb bb .. bb 00 02 cc cc .. cc 01 00 dd .. 0F 02 ee ee .. ee DE 00 02 81 01,

where

- **00 01:** tag protocol (Mifare)
- **04:** tag ID length (4 bytes)
- **xx xx xx xx:** tag ID (tag ID is read-only)
- **02:** card type (02 means Mifare 1K)
- **00:** sector id (0)
- **00:** block id (block 0 of sector 0)
- **aa aa .. aa:** 16 bytes of block 0, sector 0
- **00:** sector id (0)
- **01:** block id (block 1 of sector 0)
- **bb bb .. bb:** 16 bytes of block 1, sector 0
- **00:** sector id (0)
- **02:** block id (block 2 of sector 0)
- **cc cc .. cc:** 16 bytes of block 2, sector 0
- **01:** sector id (1)
- **00:** block id (block 0 of sector 1)
- **cc cc .. cc:** 16 bytes of block 0, sector 1
- etc...
- **0F:** sector id (15)
- **02:** block id (block 2 of sector 15)
- **ee ee .. ee:** 16 bytes of block 2, sector 15
- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

Please note that the dump does not show all 16 sectors of the tag data. They all have an identical structure: (sector id, block id followed by 16 (sixteen) bytes of data) x (3) three times per sector. The last byte of the end of data packet is equal to 1 (one), which means that the read operation has been successful.

d) Reader's response (read failure)

If the reader cannot read the tag data, the reader's response will be the following:

[xx xx yy zz zz zz zz vv] DE 00 02 81 00,

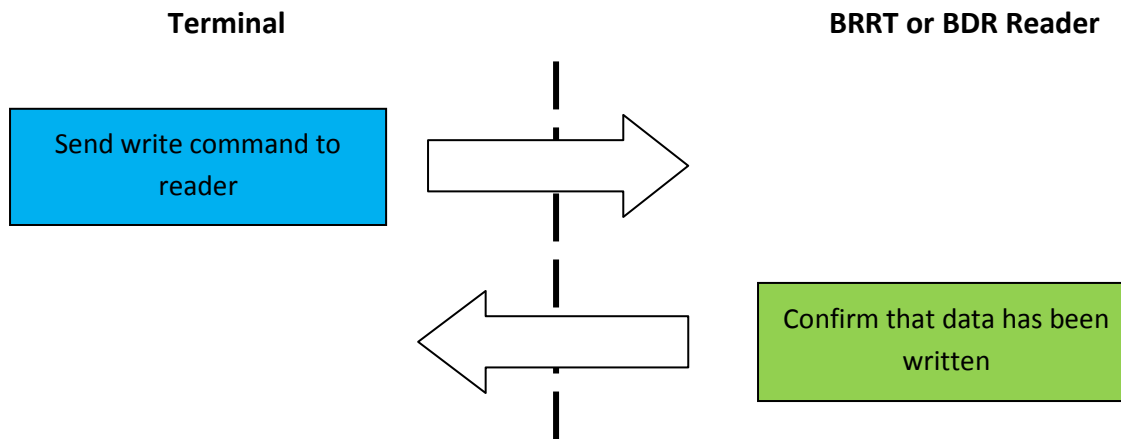
where:

- the first part of the response is optional and will be sent only if a tag of a different protocol was read
 - o **xx xx:** tag protocol
 - o **yy:** tag ID length
 - o **zz zz zz zz:** tag ID (**yy** bytes)
 - o **vv:** card type
- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

3.3 Writing tag data

a) Communication diagram

The following diagram shows what needs to be done in order to write the contents of a Mifare 1K tag:



b) Data sent from the terminal to the reader

For every sector that needs to be modified, the terminal needs to send a write command to the reader. It is recommended to verify what parts of the tag data have been modified and send only the differing sectors. In order to write a sector, the terminal needs to send to the reader a Write data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x82. In the example the user intends to modify sector 0 (zero), so the format of the command will be as follows (in hexadecimal):

DE 00 38 82 00 00 00 02 00 00 30 xx .. xx

where:

- **DE**: prefix
- **00 38**: data length (56 bytes will follow)
- **82**: command ID (write tag data)
- **00 00**: protocol mask (it's best to specify 00 00 for Mifare cards)
- **00**: timeout for the response (0 means use default = 5s)
- **02**: card type (Mifare 1K)
- **00**: start sector (0)
- **00 30**: data length (48 bytes)
- **xx .. xx**: 48 bytes of data to be written in sector 0

c) Reader's response (successful write)

If the write operation has been successful, the reader will send an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

DE 00 02 81 01,

where

- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

d) Reader's response (write failure)

If the reader cannot write the tag data, the reader's response will be the following:

DE 00 02 81 00,

where:

- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

Please note that most Mifare 1K cards have pages that are read-only. The developer should not try to modify those fields as the results are unpredictable (e.g. the reader may report a success even though the write operation failed). Please refer to the used RFID tag manufacturer specification for more details.

4. Mifare 4K tags

4.1 Tag memory structure

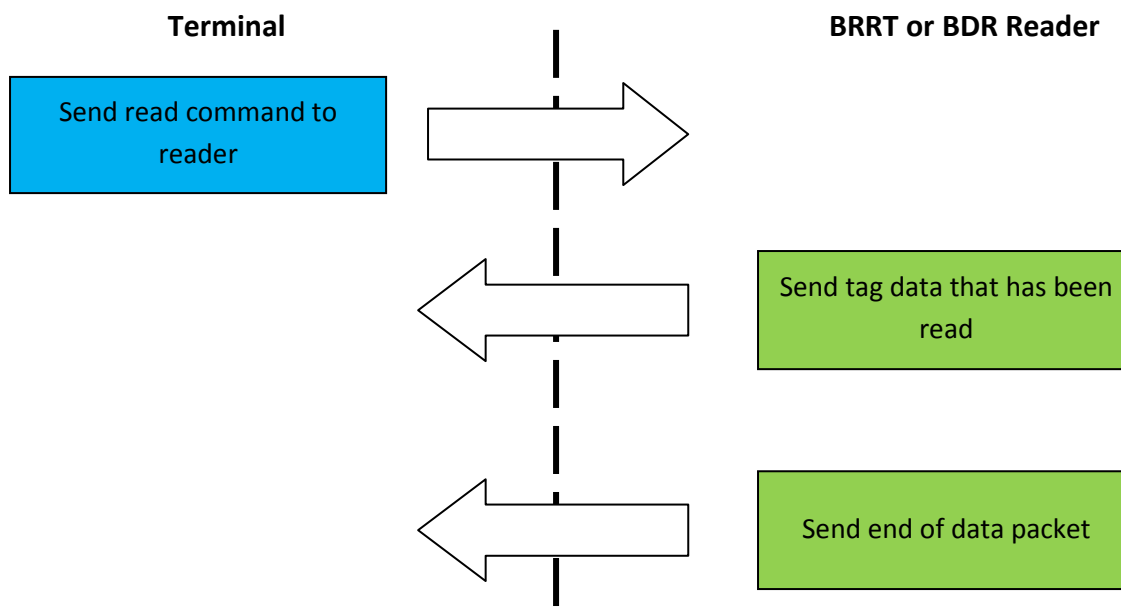
Mifare 4K tags have 3456 bytes of memory divided into:

- 32 (thirty-two) 48-bytes long sectors. Every such sector is divided into 3 (three) 16-bytes long blocks
- 8 (eight) 240-bytes long sectors. Every such sector is divided into 15 (fifteen) 16-bytes long blocks.

4.2 Reading tag data

a) Communication diagram

The following diagram shows what needs to be done in order to read the contents of a Mifare 4K tag:



b) Data sent from the terminal to the reader

First of all, the terminal needs to send to the reader a Read data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x81. In this case, the format of the command will be as follows (in hexadecimal):

DE 00 07 81 00 00 1E 03 00 FF

where:

- **DE**: prefix
- **00 07**: data length (7 bytes will follow)
- **81**: command ID
- **00 00**: protocol mask (it's best to specify 00 00 for Mifare cards)
- **1E**: timeout for the response (30s)
- **03**: card type (03 means Mifare 4K)
- **00 FF**: start/stop sectors/pages (**00 FF** means read the whole memory)

c) Reader's response (successful read)

If the read operation has been successful, the reader will send the tag ID, the tag data and then an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

00 01 04 xx xx xx xx 02 00 00 aa aa .. aa 00 01 bb bb .. bb 00 02 cc cc .. cc 01 00 dd .. 27 0E ee ee .. ee DE 00 02 81 01,

where

- **00 01**: tag protocol (Mifare)
- **04**: tag ID length (4 bytes)
- **xx xx xx xx**: tag ID (tag ID is read-only)
- **03**: card type (03 means Mifare 4K)
- **00**: sector id (0)
- **00**: block id (block 0 of sector 0)
- **aa aa .. aa**: 16 bytes of block 0, sector 0
- **00**: sector id (0)
- **01**: block id (block 1 of sector 0)
- **bb bb .. bb**: 16 bytes of block 1, sector 0
- **00**: sector id (0)
- **02**: block id (block 2 of sector 0)
- **cc cc .. cc**: 16 bytes of block 2, sector 0
- **01**: sector id (1)
- **00**: block id (block 0 of sector 1)
- **cc cc .. cc**: 16 bytes of block 0, sector 1
- etc...
- **27**: sector id (39)
- **0E**: block id (block 14 of sector 39)
- **ee ee .. ee**: 16 bytes of block 14, sector 39
- **DE 00 02 81 01**: end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

Please note that the dump does not show all 40 sectors of the tag data. The first 32 (thirty-two) sectors all have an identical structure: (sector id, block id followed by 16 (sixteen) bytes of data) x (3) three times per sector. The next eight (8) sectors' structure is: (sector id, block id followed by 16 (sixteen) bytes of data) x (15) fifteen times per sector. The last byte of the end of data packet is equal to 1 (one), which means that the read operation has been successful.

d) Reader's response (read failure)

If the reader cannot read the tag data, the reader's response will be the following:

[xx xx yy zz zz zz zz vv] DE 00 02 81 00,

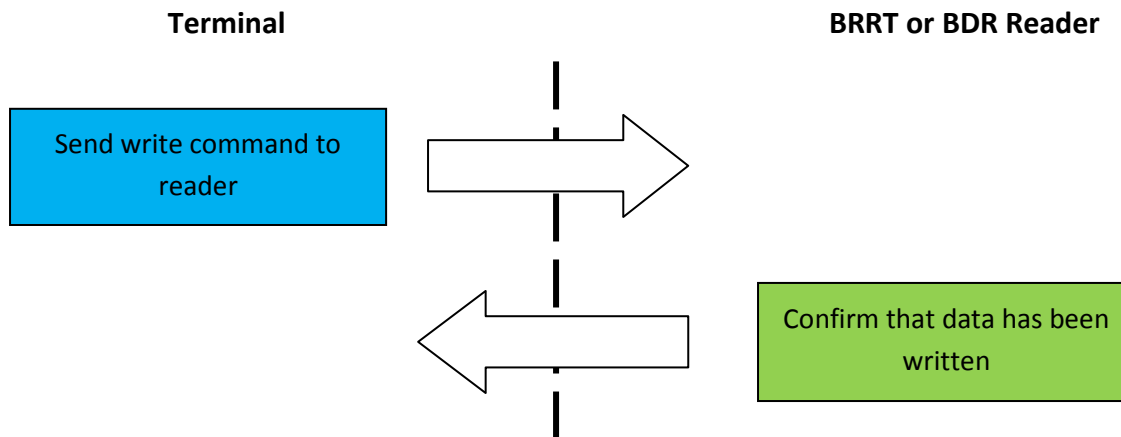
where:

- the first part of the response is optional and will be sent only if a tag of a different protocol was read
 - o **xx xx**: tag protocol
 - o **yy**: tag ID length
 - o **zz zz zz zz**: tag ID (**yy** bytes)
 - o **vv**: card type
- **DE 00 02 81 00**: end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

4.3 Writing tag data

a) Communication diagram

The following diagram shows what needs to be done in order to write the contents of a Mifare 4K tag:



b) Data sent from the terminal to the reader

For every sector that needs to be modified, the terminal needs to send a write command to the reader. It is recommended to verify what parts of the tag data have been modified and send only the differing sectors. In order to write a sector, the terminal needs to send to the reader a Write data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x82. In the example the user intends to modify sector 0 (zero) and then 39 (thirty-nine), so the format of the command will be as follows (in hexadecimal):

DE 00 38 82 00 00 00 03 00 00 30 xx .. xx

where:

- **DE**: prefix
- **00 38**: data length (56 bytes will follow)
- **82**: command ID (write tag data)
- **00 00**: protocol mask (it's best to specify 00 00 for Mifare cards)
- **00**: timeout for the response (0 means use default = 5s)
- **03**: card type (Mifare 4K)
- **00**: start sector (0)
- **00 30**: data length (48 bytes)
- **xx .. xx**: 48 bytes of data to be written in sector 0

As soon as the reader has confirmed that the data has been written (please check the next section), the developer can proceed to writing sector 39 (next page):

DE 00 F8 82 00 00 00 03 27 00 F0 yy .. yy

where:

- **DE**: prefix
- **00 F8**: data length (248 bytes will follow)
- **82**: command ID (write tag data)
- **00 00**: protocol mask (it's best to specify 00 00 for Mifare cards)
- **00**: timeout for the response (0 means use default = 5s)
- **03**: card type (Mifare 4K)
- **27**: start sector (39)
- **00 F0**: data length (240 bytes)
- **yy .. yy**: 240 bytes of data to be written in sector 39

To sum up, the developer should remember that the first 32 (thirty-two sectors) are 48-bytes long and the last 8 (eight) have a length of 240 (two hundred and forty) bytes.

c) Reader's response (successful write)

If the write operation has been successful, the reader will send an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

DE 00 02 81 01,

where

- **DE 00 02 81 01**: end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

d) Reader's response (write failure)

If the reader cannot write the tag data, the reader's response will be the following:

DE 00 02 81 00,

where:

- **DE 00 02 81 00**: end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

Please note that most Mifare 4K cards have pages that are read-only. The developer should not try to modify those fields as the results are unpredictable (e.g. the reader may report a success even though the write operation failed). Please refer to the used RFID tag manufacturer specification for more details.

5. Tag-it HF-I Plus inlay tags

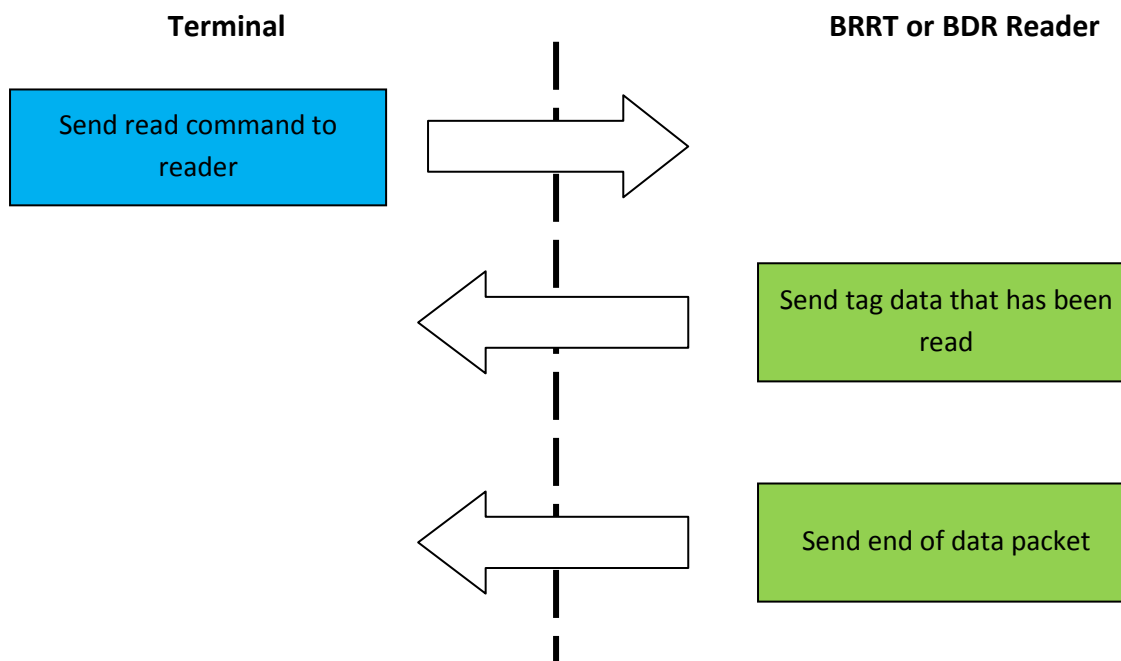
5.1 Tag memory structure

Tag-it HF-I Plus inlay tags have 256 bytes of memory divided into 64 (sixty-four) 4-bytes long blocks.

5.2 Reading tag data

a) Communication diagram

The following diagram shows what needs to be done in order to read the contents of a Tag-it HF-I Plus inlay tag:



b) Data sent from the terminal to the reader

First of all, the terminal needs to send to the reader a Read data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x81. In this case, the format of the command will be as follows (in hexadecimal):

DE 00 07 81 00 04 1E 0B 00 FF

where:

- **DE**: prefix
- **00 07**: data length (7 bytes will follow)
- **81**: command ID
- **00 04**: protocol mask (ISO 15693 for Tag-it HF-I Plus inlay)
- **1E**: timeout for the response (30s)
- **0B**: card type (Tag-it HF-I Plus inlay)
- **00 FF**: start/stop sectors/pages (**00 FF** means read the whole memory)

c) Reader's response (successful read)

If the read operation has been successful, the reader will send the tag ID, the tag data and then an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

00 04 08 xx xx xx xx xx xx xx xx 0B 00 aa bb cc dd 01 ee ff gg hh 02 ... 3F ii jj kk ll DE 00 02 81 01,
where

- **00 04:** tag protocol (ISO 15693 for Tag-it HF-I Plus inlay)
- **08:** tag ID length (8 bytes)
- **xx xx xx xx xx xx xx xx:** tag ID (tag ID is read-only)
- **0B:** card type (0B means Tag-it HF-I Plus inlay)
- **00:** block id (0)
- **aa bb cc dd:** 4 bytes of block 0
- **01:** block id (1)
- **ee ff gg hh:** 4 bytes of block 1
- etc...
- **3F:** block id (63)
- **ii jj kk ll:** 4 bytes of block 63
- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

Please note that the dump does not show all 64 blocks of the tag data. They all have an identical structure: block id followed by 4 (four) bytes of data. The last byte of the end of data packet is equal to 1 (one), which means that the read operation has been successful.

d) Reader's response (read failure)

If the reader cannot read the tag data, the reader's response will be the following:

[xx xx yy zz zz zz zz vv] DE 00 02 81 00,

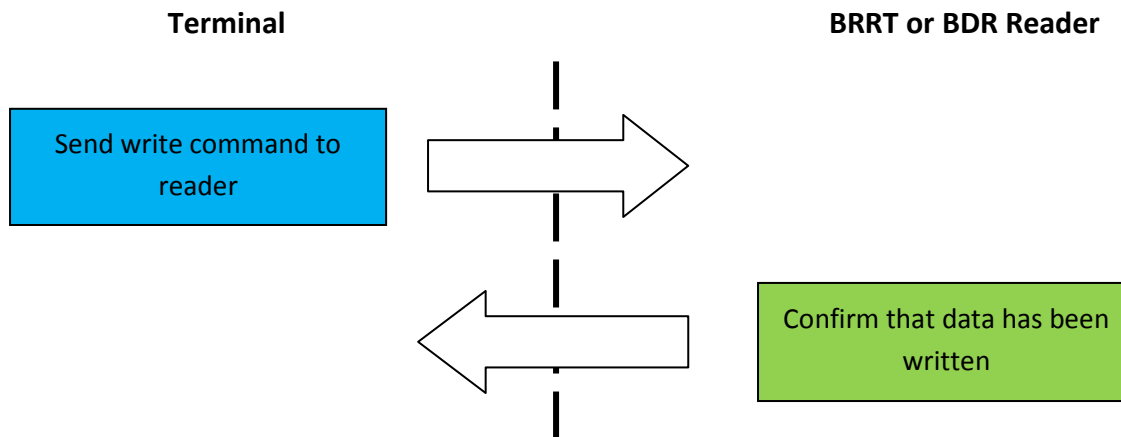
where:

- the first part of the response is optional and will be sent only if a tag of a different protocol was read
 - o **xx xx:** tag protocol
 - o **yy:** tag ID length
 - o **zz zz zz zz:** tag ID (**yy** bytes)
 - o **vv:** card type
- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

5.3 Writing tag data

a) Communication diagram

The following diagram shows what needs to be done in order to write the contents of a Tag-it HF-I Plus inlay tag:



b) Data sent from the terminal to the reader

For every block that needs to be modified, the terminal needs to send a write command to the reader. It is recommended to verify what parts of the tag data have been modified and send only the differing blocks.

In order to write a page, the terminal needs to send to the reader a Write data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x82. In the example the user intends to modify block 3 (three), so the format of the command will be as follows (in hexadecimal):

DE 00 0C 82 00 04 00 0B 03 00 04 xx xx xx xx

where:

- **DE**: prefix
- **00 0C**: data length (12 bytes will follow)
- **82**: command ID (write tag data)
- **00 04**: protocol mask (ISO 15693 for Tag-it HF-I Plus inlay)
- **00**: timeout for the response (0 means use default = 5s)
- **0B**: card type (Tag-it HF-I Plus inlay)
- **03**: start block (3)
- **00 04**: data length (4 bytes)
- **xx xx xx xx**: data to be written in block 3

c) Reader's response (successful write)

If the write operation has been successful, the reader will send an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

DE 00 02 81 01,

where

- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

d) Reader's response (write failure)

If the reader cannot write the tag data, the reader's response will be the following:

DE 00 02 81 00,

where:

- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

6. Tag-it HF-I Plus chip tags

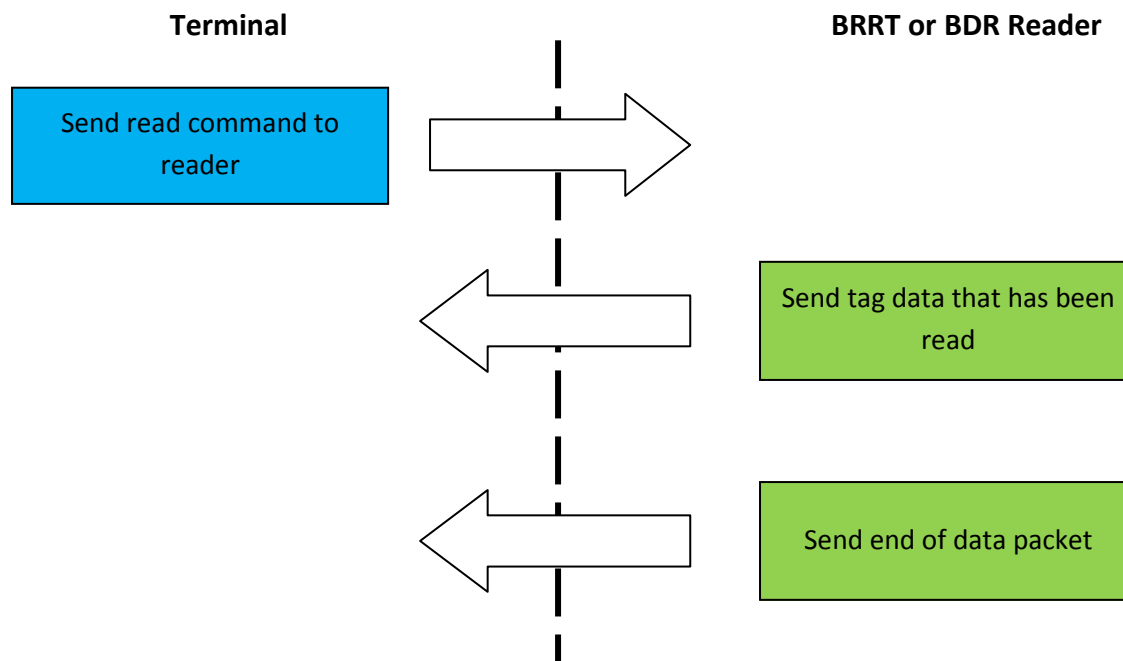
6.1 Tag memory structure

Tag-it HF-I Plus chip tags have 256 bytes of memory divided into 64 (sixty-four) 4-bytes long blocks.

6.2 Reading tag data

a) Communication diagram

The following diagram shows what needs to be done in order to read the contents of a Tag-it HF-I Plus chip tag:



b) Data sent from the terminal to the reader

First of all, the terminal needs to send to the reader a Read data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x81. In this case, the format of the command will be as follows (in hexadecimal):

DE 00 07 81 00 04 1E 0C 00 FF

where:

- **DE**: prefix
- **00 07**: data length (7 bytes will follow)
- **81**: command ID
- **00 04**: protocol mask (ISO 15693 for Tag-it HF-I Plus chip)
- **1E**: timeout for the response (30s)
- **0C**: card type (Tag-it HF-I Plus chip)
- **00 FF**: start/stop sectors/pages (**00 FF** means read the whole memory)

c) Reader's response (successful read)

If the read operation has been successful, the reader will send the tag ID, the tag data and then an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

00 04 08 xx xx xx xx xx xx xx xx 0C 00 aa bb cc dd 01 ee ff gg hh 02 ... 3F ii jj kk ll DE 00 02 81 01,
where

- **00 04:** tag protocol (ISO 15693 for Tag-it HF-I Plus chip)
- **08:** tag ID length (8 bytes)
- **xx xx xx xx xx xx xx xx:** tag ID (tag ID is read-only)
- **0C:** card type (0C means Tag-it HF-I Plus chip)
- **00:** block id (0)
- **aa bb cc dd:** 4 bytes of block 0
- **01:** block id (1)
- **ee ff gg hh:** 4 bytes of block 1
- etc...
- **3F:** block id (63)
- **ii jj kk ll:** 4 bytes of block 63
- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

Please note that the dump does not show all 64 blocks of the tag data. They all have an identical structure: block id followed by 4 (four) bytes of data. The last byte of the end of data packet is equal to 1 (one), which means that the read operation has been successful.

d) Reader's response (read failure)

If the reader cannot read the tag data, the reader's response will be the following:

[xx xx yy zz zz zz zz vv] DE 00 02 81 00,

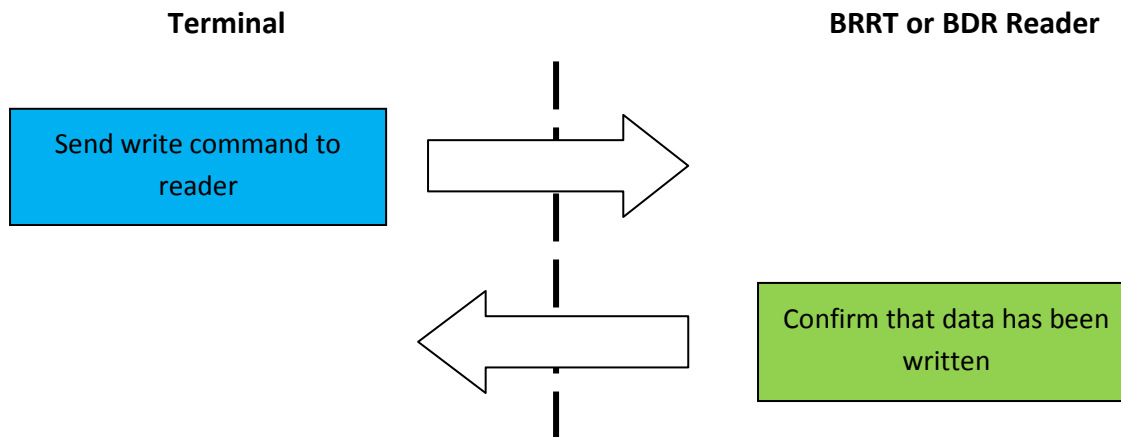
where:

- the first part of the response is optional and will be sent only if a tag of a different protocol was read
 - o **xx xx:** tag protocol
 - o **yy:** tag ID length
 - o **zz zz zz zz:** tag ID (**yy** bytes)
 - o **vv:** card type
- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

6.3 Writing tag data

a) Communication diagram

The following diagram shows what needs to be done in order to write the contents of a Tag-it HF-I Plus chip tag:



b) Data sent from the terminal to the reader

For every block that needs to be modified, the terminal needs to send a write command to the reader. It is recommended to verify what parts of the tag data have been modified and send only the differing blocks.

In order to write a page, the terminal needs to send to the reader a Write data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x82. In the example the user intends to modify block 3 (three), so the format of the command will be as follows (in hexadecimal):

DE 00 0C 82 00 04 00 0C 03 00 04 xx xx xx xx

where:

- **DE**: prefix
- **00 0C**: data length (12 bytes will follow)
- **82**: command ID (write tag data)
- **00 04**: protocol mask (ISO 15693 for Tag-it HF-I Plus chip)
- **00**: timeout for the response (0 means use default = 5s)
- **0C**: card type (Tag-it HF-I Plus chip)
- **03**: start block (3)
- **00 04**: data length (4 bytes)
- **xx xx xx xx**: data to be written in block 3

c) Reader's response (successful write)

If the write operation has been successful, the reader will send an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

DE 00 02 81 01,

where

- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

d) Reader's response (write failure)

If the reader cannot write the tag data, the reader's response will be the following:

DE 00 02 81 00,

where:

- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

7. I-CODE SLI tags

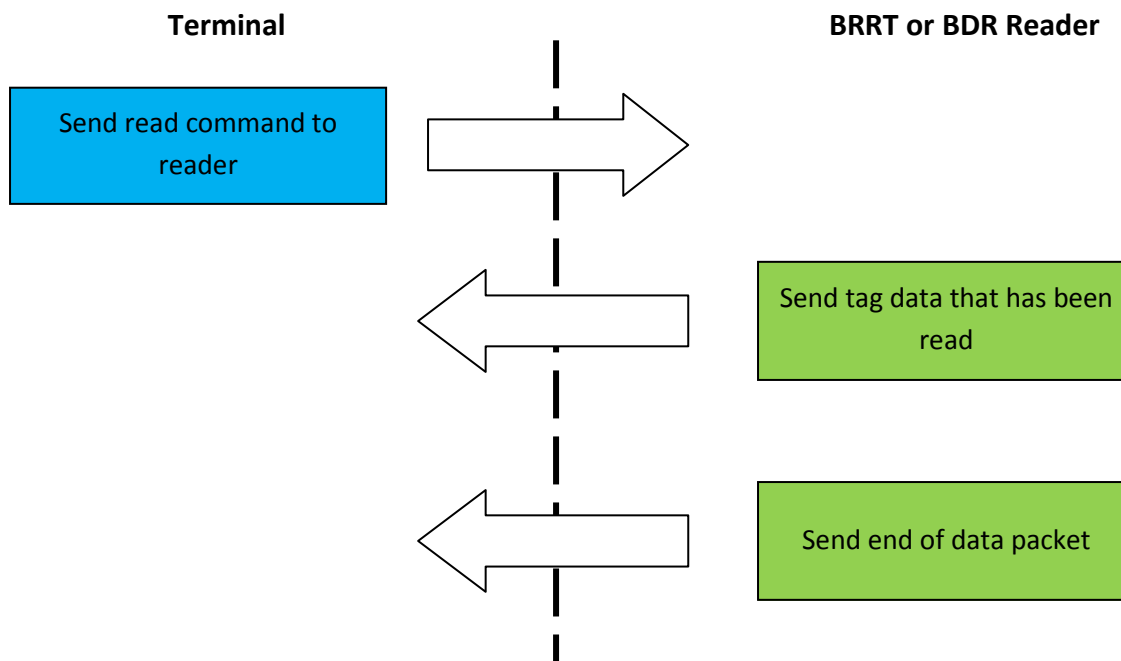
7.1 Tag memory structure

I-CODE SLI tags have 112 bytes of memory divided into 28 (twenty-eight) 4-bytes long blocks.

7.2 Reading tag data

a) Communication diagram

The following diagram shows what needs to be done in order to read the contents of an I-CODE SLI tag:



b) Data sent from the terminal to the reader

First of all, the terminal needs to send to the reader a Read data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x81. In this case, the format of the command will be as follows (in hexadecimal):

DE 00 07 81 00 04 1E 07 00 FF

where:

- **DE**: prefix
- **00 07**: data length (7 bytes will follow)
- **81**: command ID
- **00 04**: protocol mask (ISO 15693 for I-CODE SLI)
- **1E**: timeout for the response (30s)
- **07**: card type (I-CODE SLI)
- **00 FF**: start/stop sectors/pages (**00 FF** means read the whole memory)

c) Reader's response (successful read)

If the read operation has been successful, the reader will send the tag ID, the tag data and then an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

00 04 08 xx xx xx xx xx xx xx xx 07 00 aa bb cc dd 01 ee ff gg hh 02 ... 1B ii jj kk ll DE 00 02 81 01,
where

- **00 04:** tag protocol (ISO 15693 for I-CODE SLI)
- **08:** tag ID length (8 bytes)
- **xx xx xx xx xx xx xx xx:** tag ID (tag ID is read-only)
- **07:** card type (07 means I-CODE SLI)
- **00:** block id (0)
- **aa bb cc dd:** 4 bytes of block 0
- **01:** block id (1)
- **ee ff gg hh:** 4 bytes of block 1
- etc...
- **1B:** block id (27)
- **ii jj kk ll:** 4 bytes of block 27
- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

Please note that the dump does not show all 28 blocks of the tag data. They all have an identical structure: block id followed by 4 (four) bytes of data. The last byte of the end of data packet is equal to 1 (one), which means that the read operation has been successful.

d) Reader's response (read failure)

If the reader cannot read the tag data, the reader's response will be the following:

[xx xx yy zz zz zz zz vv] DE 00 02 81 00,

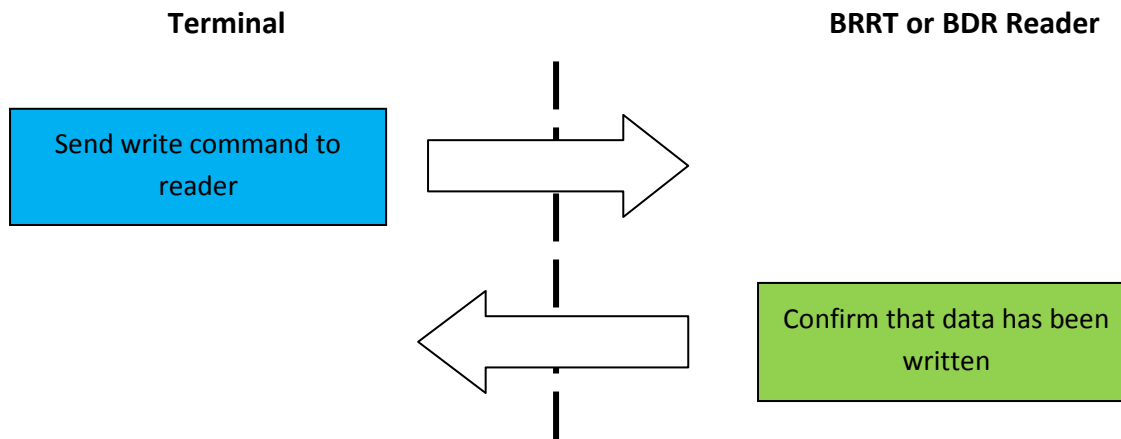
where:

- the first part of the response is optional and will be sent only if a tag of a different protocol was read
 - o **xx xx:** tag protocol
 - o **yy:** tag ID length
 - o **zz zz zz zz:** tag ID (**yy** bytes)
 - o **vv:** card type
- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

7.3 Writing tag data

a) Communication diagram

The following diagram shows what needs to be done in order to write the contents of an I-CODE SLI tag:



b) Data sent from the terminal to the reader

For every block that needs to be modified, the terminal needs to send a write command to the reader. It is recommended to verify what parts of the tag data have been modified and send only the differing blocks.

In order to write a page, the terminal needs to send to the reader a Write data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x82. In the example the user intends to modify block 3 (three), so the format of the command will be as follows (in hexadecimal):

DE 00 0C 82 00 04 00 07 03 00 04 xx xx xx xx

where:

- **DE**: prefix
- **00 0C**: data length (12 bytes will follow)
- **82**: command ID (write tag data)
- **00 04**: protocol mask (ISO 15693 for I-CODE SLI)
- **00**: timeout for the response (0 means use default = 5s)
- **07**: card type (I-CODE SLI)
- **03**: start block (3)
- **00 04**: data length (4 bytes)
- **xx xx xx xx**: data to be written in block 3

c) Reader's response (successful write)

If the write operation has been successful, the reader will send an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

DE 00 02 81 01,

where

- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

d) Reader's response (write failure)

If the reader cannot write the tag data, the reader's response will be the following:

DE 00 02 81 00,

where:

- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

8. I-CODE SLI-S tags

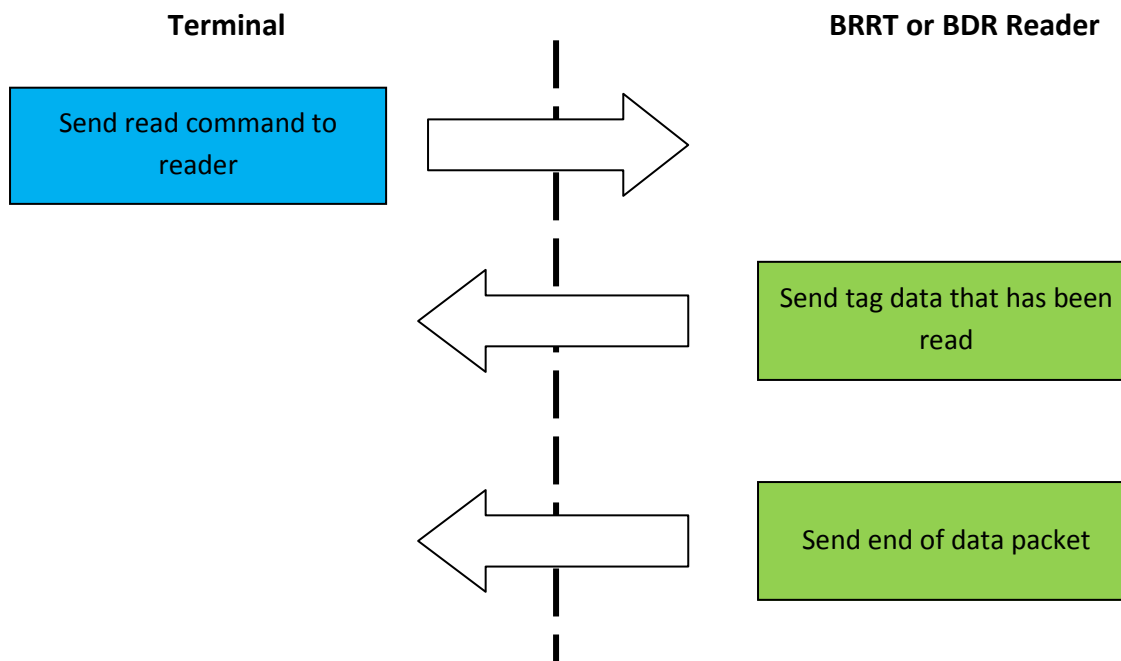
8.1 Tag memory structure

I-CODE SLI-S tags have 160 bytes of memory divided into 40 (forty) 4-bytes long blocks.

8.2 Reading tag data

a) Communication diagram

The following diagram shows what needs to be done in order to read the contents of an I-CODE SLI-S tag:



b) Data sent from the terminal to the reader

First of all, the terminal needs to send to the reader a Read data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x81. In this case, the format of the command will be as follows (in hexadecimal):

DE 00 07 81 00 04 1E 09 00 FF

where:

- **DE**: prefix
- **00 07**: data length (7 bytes will follow)
- **81**: command ID
- **00 04**: protocol mask (ISO 15693 for I-CODE SLI-S)
- **1E**: timeout for the response (30s)
- **09**: card type (I-CODE SLI-S)
- **00 FF**: start/stop sectors/pages (**00 FF** means read the whole memory)

c) Reader's response (successful read)

If the read operation has been successful, the reader will send the tag ID, the tag data and then an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

00 04 08 xx xx xx xx xx xx xx xx 09 00 aa bb cc dd 01 ee ff gg hh 02 ... 27 ii jj kk ll DE 00 02 81 01,
where

- **00 04:** tag protocol (ISO 15693 for I-CODE SLI-S)
- **08:** tag ID length (8 bytes)
- **xx xx xx xx xx xx xx xx:** tag ID (tag ID is read-only)
- **09:** card type (09 means I-CODE SLI-S)
- **00:** block id (0)
- **aa bb cc dd:** 4 bytes of block 0
- **01:** block id (1)
- **ee ff gg hh:** 4 bytes of block 1
- etc...
- **27:** block id (39)
- **ii jj kk ll:** 4 bytes of block 39
- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

Please note that the dump does not show all 40 blocks of the tag data. They all have an identical structure: block id followed by 4 (four) bytes of data. The last byte of the end of data packet is equal to 1 (one), which means that the read operation has been successful.

d) Reader's response (read failure)

If the reader cannot read the tag data, the reader's response will be the following:

[xx xx yy zz zz zz zz vv] DE 00 02 81 00,

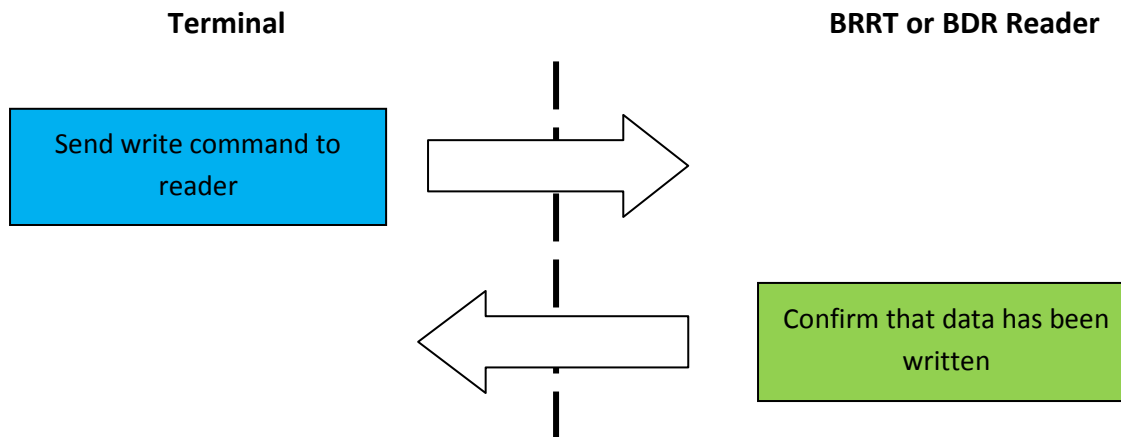
where:

- the first part of the response is optional and will be sent only if a tag of a different protocol was read
 - o **xx xx:** tag protocol
 - o **yy:** tag ID length
 - o **zz zz zz zz:** tag ID (**yy** bytes)
 - o **vv:** card type
- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)

8.3 Writing tag data

a) Communication diagram

The following diagram shows what needs to be done in order to write the contents of an I-CODE SLI-S tag:



b) Data sent from the terminal to the reader

For every block that needs to be modified, the terminal needs to send a write command to the reader. It is recommended to verify what parts of the tag data have been modified and send only the differing blocks.

In order to write a page, the terminal needs to send to the reader a Write data command to the reader. This command is described in the communication protocol guide of the reader; its code ID is 0x82. In the example the user intends to modify block 3 (three), so the format of the command will be as follows (in hexadecimal):

DE 00 0C 82 00 04 00 09 03 00 04 xx xx xx xx

where:

- **DE**: prefix
- **00 0C**: data length (12 bytes will follow)
- **82**: command ID (write tag data)
- **00 04**: protocol mask (ISO 15693 for I-CODE SLI-S)
- **00**: timeout for the response (0 means use default = 5s)
- **09**: card type (I-CODE SLI-S)
- **03**: start block (3)
- **00 04**: data length (4 bytes)
- **xx xx xx xx**: data to be written in block 3

c) Reader's response (successful write)

If the write operation has been successful, the reader will send an end of data packet with the success byte set to 1 (one). On the other hand, if the operation has failed, the reader may send the tag ID and an end of data packet with the success byte set to 0 (zero).

The following is an example of the reader's response:

DE 00 02 81 01,

where

- **DE 00 02 81 01:** end of data, where 00 02 – length, 81 – command ID, 01 – success status (01 means success)

d) Reader's response (write failure)

If the reader cannot write the tag data, the reader's response will be the following:

DE 00 02 81 00,

where:

- **DE 00 02 81 00:** end of data, where 00 02 – length, 81 – command ID, 00 – success status (01 means failure)