Reader-Host-Protocol
NEO2-UHF-USB Reader

History of Change:

13.07.2009	Changed output power to attenuation.	v0.01
28.07.2009	Added Lock command	v0.02
03.08.2009	Adapted Return codes and enums	v0.03
10.11.2009	Added state information and interrupt	v0.04
30.04.2010	Added sensitivity setting	v0.05
12.05.2010	Added the status register	v0.06
24.01.2011	Added GPIO commands	v0.07
09.05.2011	Added duplex heartbeat description	v0.08
08.12.2011	Added LBT Params, Added Boot-Up-Finished Interrupt, Added Notifications	v0.09
21.12.2011	Added Antenna Commands	v0.10
06.09.2012	Added return code RESULT_PENDING	v0.11
09.11.2012	Added GPIO-Values-Changed Interrupt	v0.12
30.01.2013	Added new types of heartbeat	v0.13
12.02.2013	Corrected some parts	v0.14
26.06.2014	Corrected some parts	V0.15
16.08.2017	Corrected some parts	v0.16
15.03.2018	Added Error-Occurred-Interrupt	v0.17
20.03.2025	Revision for product clone	v1.01

iDTRONIC GmbH
Ludwig-Reichling-Straße 4
67059 Ludwigshafen
Germany/Deutschland

Phone +49 621 6690094-0
Fax +49 621 6690094-9
E-Mail info@idtronic-rfid.com
Web https://idtronic-rfid.com/

Issue 1.01
– 20 March 2025 –

Contents

1 Introduction

This document describes the communication protocol that is used for the NEO2 UHF-RFID Desktop Reader. It is a very small but essential protocol and has the possibility to be enhanced.
In the following chapters, the structures and the values of the protocol are described.

2 Packet Transmission

This protocol does not depend on the interface that is used for the communication. Hence the data structure is the same with every interface.

3 Packet Structure

All exchanged packets follow the same structure as shown below:

Start Bytes	3 Bytes
Command Start Byte	1 Byte
Command	2 Bytes
Length Start Byte	1 Byte
Length	1 Byte
Payload Start Byte	1 Byte
Payload	Variable
Checksum Start Byte	1 Byte
Checksum	1 Byte

Start Byte: The start bytes are used to signalize the start of a new packet.
They are always **0x52**, **0x46**, **0x45** (ASCII: R, F, E).
Command: The two command bytes describe which command should be executed. These commands are described in chapter 5.
Length & Payload: These two fields contain the payload and the length of the payload as in the count of characters in the payload field. If length is zero, the payload start byte and payload can / must be left out.
Checksum: The checksum is just a simple XOR connection of all data before.

All other start bytes are used to synchronize the protocol and to reduce the probability of the misinterpretation of the byte sequence.

4 Return Codes

The return codes of all reader functions are listed below:

Name	Value	Description
RFE_RET_SUCCESS	0x00	Everything went fine.
RFE_RET_RESULT_PENDING	0x01	The operation is pending, the result will be sent later.
RFE_RET_ERR_OP_NOT_SUPPORTED	0x50	Operation is not supported on this reader.
RFE_RET_ERR_UNKOWN_ERR	0x51	Unknown error.
RFE_RET_ERR_ON_EXEC_OP	0x52	The operation could not be executed.
RFE_RET_ERR_COULD_NOT_WRITE	0x53	The reader could not write the value.
RFE_RET_ERR_WRONG_PARAM_COUNT	0x54	The function was called with the wrong parameter count.
RFE_RET_ERR_WRONG_PARAM	0x55	The function was called with the wrong parameter.

The return codes of the Tag Manipulation Interface (TMI) that can be returned when a tag should be manipulated are listed below:

Name	Value	Description
RFE_RET_TMI_TAG_UNREACHABLE	0xA0	The reader could not reach the tag.
RFE_RET_TMI_MEM_OVERRUN	0xA1	The specified memory space is not valid.
RFE_RET_TMI_MEM_LOCKED	0xA2	The specified memory space is locked.
RFE_RET_TMI_INSUFFICIENT_POWER	0xA3	The tag has too little power.
RFE_RET_TMI_WRONG_PASSWORD	0xA4	The specified password is wrong.

This table should be used as an enum with the name `RFE_RET_VALUE`. This type is used in the following description. It is used as an unsigned char.

5 Variable Dictionary

5.1 Reader Common (01)

Command Byte 1		Command Byte 2		Version
Reader-Common	0x01	Get-Serial Number	0x01	
		Get-Reader Type	0x02	
		Get-Hardware Revision	0x03	
		Get-Software Revision	0x04	
		Get-Bootloader Revision	0x05	
		Get-Current-System	0x06	
		Get-Current-State	0x07	
		Get-Status-Register	0x08	
		Get-Antenna-Count	0x10	1.08

5.2 RF-Settings (02)

Command Byte 1		Command Byte 2		Version
Reader-RF	0x02	Get-Attenuation	0x01	
		Get-Frequency	0x02	
		Get-Sensitivity	0x03	
		Get-LBT-Params	0x04	1.07
		Set-Attenuation	0x81	1.08
		~~Set-Frequency~~	~~0x82~~	Deprecated
		~~Set-Sensitivity~~	~~0x83~~	Deprecated
		~~Set-LBT-Params~~	~~0x84~~	Deprecated

5.3 Reader Control (03)

Command Byte 1		Command Byte 2		Version
Reader-Control	0x03	Reboot	0x01	
		Set-Heartbeat	0x02	
		Set-Antenna-Power	0x03	
		~~Set-Attenuation (dBm)~~	~~0x04~~	Deprecated since 1.08
		~~Set-Frequency~~	~~0x05~~	Deprecated since 1.08
		~~Set-Sensitivity~~	~~0x06~~	Deprecated since 1.08

	~~Set-LBT-Params~~	~~0x07~~	Deprecated since 1.08
	Restore-Factory-Settings	0x20	
	Save-Settings-Permanent	0x21	
	~~Set-Param~~	~~0x30~~	Deprecated
	Get-Param	0x31	
	~~Set-Device-Name~~	~~0x32~~	Deprecated
	~~Get-Device-Name~~	~~0x33~~	Deprecated
	~~Set-Device-Location~~	~~0x34~~	Deprecated
	~~Get-Device-Location~~	~~0x35~~	Deprecated

5.4 GPIO (05)

Command Byte 1		Command Byte 2		Version
GPIO-Control	0x05	Get-GPIO-Caps	0x01	
		Get-GPIO-Direction	0x02	
		~~Set-GPIO-Direction~~	~~0x03~~	Deprecated
		Get-GPIO	0x04	
		~~Set-GPIO~~	~~0x05~~	Deprecated
		~~Clear-GPIO~~	~~0x06~~	Deprecated

5.5 Antenna (06)

Command Byte 1		Command Byte 2		Version
Antenna-Control	0x06	~~Set-Antenna-Sequence~~	~~0x01~~	Deprecated
		~~Get-Antenna-Sequence~~	~~0x02~~	Deprecated
		~~Set-Working-Antenna~~	~~0x03~~	Deprecated
		~~Get-Working-Antenna~~	~~0x04~~	Deprecated

5.6 Notifications (10)

Command Byte 1		Command Byte 2		Version
Notifications	0x10	~~Activate-Notifications~~	~~0x01~~	Deprecated
		~~Deactivate-Notifications~~	~~0x02~~	Deprecated
		~~Get-Active-Notifications~~	~~0x03~~	Deprecated

5.7 Tag Control (50)

Command Byte 1		Command Byte 2		Version
Tag-Functions	0x50	Inventory-Single	0x01	
		Inventory-Cyclic	0x02	
		Read-From-Tag	0x03	
		Write-To-Tag	0x04	
		Lock-Tag	0x05	
		Kill-Tag	0x06	
		~~Custom-Tag-Command~~	~~0x10~~	Deprecated
		~~Read-Multiple-From-Tag~~	~~0x20~~	Deprecated

5.8 Reader-Interrupts (90)

Command Byte 1		Command Byte 2		Version
Interrupt	0x90	Heart-Beat-Interrupt	0x01	
		Inventory-Cyclic-Interrupt	0x02	
		State-Changed-Interrupt	0x03	
		Status-Reg-Changed-Interrupt	0x04	
		Boot-Up-Finished	0x05	1.02
		~~Notification-Interrupt~~	~~0x06~~	Deprecated
		Operation-Result-Interrupt	0x08	
		GPIO-Values-Changed-Interrupt	0x09	1.16

6 Parameter Dictionary

The Parameters that can be read with the command Get-Param are shown in the following table:

Name	Address	Size	Description
Inventory Mode	0x0000	1 Byte	The used inventory mode (0x02 -> Standard Multi Tag)
Power Safe Settings	0x0001	3 Bytes	The settings for the power safe (0x010032 -> On, 50 ms sleep time)
RSSI	0x0002	1 Byte	Defines if the RSSI value is sent to the host (0x00 -> Off)
Tag-ID-Behaviour Mode	0x0003	1 Byte	Defines how the reader should behave if a tag is detected (0x00 -> Send-Tag-ID-Immediately)
Send-Frequency	0x0005	1 Byte	Defines if the reader should send the frequency where a tag was detected (0x00 -> Off)
PostDetect Read	0x0006	9 Bytes	Settings for a read after a tag was detected (0 -> Off)
PowerSafeCount	0x0007	2 Bytes	
Gen2-Link-Frequency	0x0020	1 Byte	The used link frequency (0x02 -> 160 kHz)
Gen2-Bit-Encoding	0x0021	1 Byte	The used bit encoding (0x01 -> Miller 2)
Gen2-Modulation-Depth	0x0022	1 Byte	The used modulation depth (0x64 -> 100%)
Gen2-EPC-Size	0x0023	1 Byte	The expected EPC size (0x0C -> 12 Bytes)
Gen2-Send-Handle	0x0024	1 Byte	Send handle with EPC (0x00 -> Off)
Gen2-Send-PC	0x0025	1 Byte	Send PC with EPC (0x00 -> Off)
Gen2-Q-Setting	0x0026	3 Bytes	Q settings for the inventory (0x04020F -> initial slot count 16, min 4, max 32768)
Gen2-Q-Method	0x0027	1 Byte	Q adjusted method (adjust Q value between inventories) (0x01 -> RFE Dynamic Adjust Method)
Gen2-Session	0x0028	1 Byte	Gen2 session (0x01 -> Session 1)
Gen2-InventoryCount	0x0029	1 Byte	Inventory count (0x01 -> 1 Inventory per select cmd)
Gen2-Select-Mask #1	0x002A	6 Bytes + Mask	Selection Mask #1 used for Gen2-Select command (0x000000000000 -> no filter specified)
Gen2-Select-Mask #2	0x002B	6 Bytes + Mask	Selection Mask #2 used for Gen2-Select command (0x000000000000 -> no filter specified)

7 Detailed Description

7.1 Get Serial Number (01-01)

This command returns the serial number of the reader.
Return Values: unsigned long serialNumber
Example:

PC -> Reader: `52 46 45 01 0101 02 00 04 cs`
Reader -> PC: `52 46 45 01 0101 02 04 03 03D12C41 04 cs`
dataLength = 0x04 -> 4 Bytes
serialNumber = 0x03 0xD1 0x2C 0x41

7.2 Get Reader Type (01-02)

This command returns the type of the reader.
Return Values: unsigned long readerType
Example:

PC -> Reader: `52 46 45 01 0102 02 00 04 cs`
Reader -> PC: `52 46 45 01 0102 02 04 03 81220501 04 cs`
dataLength = 0x04 -> 4 Bytes
readerType = 0x81 0x22 0x05 0x01

7.3 Get Hardware Revision (01-03)

This command returns the hardware revision of the reader. The version number is split into blocks of 4 bit. One of these blocks represents a decimal character. There are always two characters in front of the point and two after the point.
Return Values: unsigned long hardwareRevision
Example:

PC -> Reader: `52 46 45 01 0103 02 00 04 cs`
Reader -> PC: `52 46 45 01 0103 02 04 03 00100100 04 cs`
dataLength = 0x04 -> 4 Bytes
hardwareRevision = 0x00 0x10 0x01 0x00

7.4 Get Software Revision (01-04)

This command returns the software revision of the reader. The version number is split into blocks of 4 bit. One of these blocks represents a decimal character. There are always two characters in front of the point and two after the point. The first two bytes define the application version and the last two bytes the version of the used kernel.
Return Values: unsigned long softwareRevision
Example:

PC -> Reader: `52 46 45 01 0104 02 00 04 cs`
Reader -> PC: `52 46 45 01 0104 02 04 03 00000226 04 cs`
dataLength = 0x04 -> 4 Bytes
softwareRevision = 0x00 0x00 0x02 0x26 -> App: 00.00, Kernel: 02.26

7.5 Get Bootloader Revision (01-05)

This command returns the bootloader revision of the reader. The version number is split into blocks of 4 bit. One of these blocks represents a decimal character. There are always two characters in front of the point and two after the point.
Return Values: unsigned long bootloaderRevision
Example:

PC -> Reader: `52 46 45 01 0105 02 00 04 cs`
Reader -> PC: `52 46 45 01 0105 02 04 03 00000117 04 cs`
dataLength = 0x04 -> 4 Bytes
softwareRevision = 0x00 0x00 0x01 0x17 -> 01.17

7.6 Get Current State (01-07)

This command returns the current state of the reader.
Return Values: RFE_RET_STATE state
Example:

PC -> Reader: `52 46 45 01 0107 02 00 04 cs`
Reader -> PC: `52 46 45 01 0107 02 01 03 00 04 cs`
dataLength = 0x04 -> 4 Bytes
state = 0x00 -> RFE_STATE_IDLE

7.7 Get Status Register (01-08)

This command returns the status register of the reader.
Return Values: unsigned long statusRegister
Example:

PC -> Reader: `52 46 45 01 0108 02 00 04 cs`
Reader -> PC: `52 46 45 01 0108 02 08 03 0000 0000 0000 0000 04 cs`
dataLength = 0x08 -> 8 Bytes
softwareRevision = 0x0000000000000000

7.8 Get Antenna Count (01-10)

This command returns the antenna count of the reader.
Return Values: unsigned char count
Example:

PC -> Reader: `52 46 45 01 0110 02 00 04 cs`
Reader -> PC: `52 46 45 01 0110 02 01 03 01 04 cs`
dataLength = 0x01 -> 1 Byte
softwareRevision = 0x01 -> 1 Antenna

7.9 Get Attenuation (02-01)

This command returns the maximal potential and the current attenuation in dBm.
Return Values: RFE_RET_VALUE status, unsigned short maxAttenuation, unsigned short currentAttenuation
Example:

PC -> Reader: `52 46 45 01 0201 02 00 04 cs`
Reader -> PC: `52 46 45 01 0201 02 05 03 00 0013 0000 04 cs`

dataLength = 0x05 -> 5 Bytes
status = 0x00 -> Success
maxAttenuation = 0x0013 -> 19 dBm
currentAttenuation = 0x0000 -> 0 dBm

7.10 Get Frequency (02-02)

This command returns the current frequency and the maximum count of frequencies that can be set.

enum HOPPING_MODE (unsigned char)	
STATIC_UP	0x00
RANDOM	0x01

Return Values: RFE_RET_VALUE status, HOPPING_MODE mode, unsigned char maxFrequencyCount, unsigned char frequencyCount, unsigned char frequencys [frequencyCount] [3]

Example:

PC -> Reader: `52 46 45 01 0202 02 00 04 cs`

Reader -> PC: `52 46 45 01 0202 02 10 03 00 01 32 04 0D35A4 0D37FC 0D3A54 0D3CAC 04 cs`

dataLength = 0x10 -> 16 Bytes
status = 0x00 -> Success
mode = 0x01 -> Random
maxFrequencyCount = 0x32 -> 50 Frequencies
frequencyCount = 0x04 -> 4 Frequencies
frequency1 = 0x0D35A4 -> 865700 kHz
frequency2 = 0x0D37FC -> 866300 kHz
frequency3 = 0x0D3A54 -> 866900 kHz
frequency4 = 0x0D3CAC -> 867500 kHz

7.11 Get Sensitivity (02-03)

This command returns the minimal potential and the current sensitivity in dBm.

Return Values: RFE_RET_VALUE status, signed short maxSensitivity, signed short minSensitivity, signed short currentSensitivity

Example:

PC -> Reader: `52 46 45 01 0203 02 00 04 cs`

Reader -> PC: `52 46 45 01 0203 02 07 03 00 FF9C FFE2 FFB5 04 cs`

dataLength = 0x07 -> 7 Bytes
status = 0x00 -> Success
maxSensitivity = 0xFF9C -> -100 dBm
minSensitivity = 0xFFE2 -> -30 dBm
currentSensitivity = 0xFFB5 -> -75 dBm

7.12 Get LBT Params (02-04)

Since v1.07. This command returns the current set LBT parameters.

Return Values: RFE_RET_VALUE status, unsigned short listenTime, unsigned short idleTime, unsigned short maxAllocationTime, signed short rssiThreshold

Example:

PC -> Reader: `52 46 45 01 0204 02 00 04 cs`

Reader -> PC: `52 46 45 01 0204 02 09 03 00 0001 0000 0190 FFD8 04 cs`

dataLength = 0x09 -> 9 Bytes
status = 0x00 -> Success
listenTime = 0x0001 -> 1 ms
idleTime = 0x0000 -> 0 ms
maxAllocationTime = 0x0190 -> 400 ms
rssiThreshold = 0xFFD8 -> -40 dBm

7.13 Set Attenuation (02-81)

This command can be used to set the attenuation of the reader in dBm. The maximal attenuation value can be found in the documentation of the reader or can be read from the reader using the Get-Attenuation command.
Parameters: unsigned short value
Return Values: RFE_RET_VALUE status
Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM
Example:

PC -> Reader: `52 46 45 01 0281 02 02 03 000A 04 cs`
dataLength = 0x02 -> 2 Bytes
value = 0x000A -> 10 dBm
Reader -> PC: `52 46 45 01 0281 02 01 03 00 04 cs`
dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

7.14 Reboot (03-01)

This function can be used to reboot the reader.
Return Values: None, the reader is rebooted immediately.
Example: PC -> Reader `52 46 45 01 0301 02 00 04 cs`

7.15 Set Heartbeat (03-02)

This command can be used to enable the reader to send a periodic heartbeat. The heartbeat can be turned ON/OFF and an interval for these heartbeat messages can be specified. If no interval is specified, the reader takes the interval from the factory settings. This value can be found in the data sheet of the reader. If the DUPLEX heartbeat is selected, the reader expects the same heartbeat package from the host. If the reader does not get the package from the host in the specified interval, it stops any active scan immediately. The best method to send the heartbeat is to respond to the heartbeat of the reader with the host heartbeat.
If the STATE heartbeat is selected, the reader attaches the current state to each heartbeat package.

enum HEARTBEAT_SIGNAL (unsigned char)	
HEARTBEAT_OFF	0x00
HEARTBEAT_ON	0x01
HEARTBEAT_DUPLEX_ON	0x02
HEARTBEAT_STATE_ON	0x03
HEARTBEAT_DUPLEX_STATE_ON	0x04

Parameters: HEARTBEAT_SIGNAL mode, (unsigned short interval_in_ms)
Return Values: RFE_RET_VALUE status
Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example:

PC -> Reader: `52 46 45 01 0302 02 03 03 01 01F4 04 cs`
dataLength = 0x03 -> 3 Bytes
mode = 0x01 -> HEARTBEAT_ON
interval = 0x01F4 -> 500 ms

Reader -> PC: `52 46 45 01 0302 02 01 03 00 04 cs`
dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

7.16 Set Antenna Power (03-03)

This function can be used to set the antenna power on and off. The values are listed below:

enum ANTENNA_POWER (unsigned char)	
ANTENNA_OFF	0x00
ANTENNA_ON	0x01

Parameters: ANTENNA_POWER mode
Return Values: RFE_RET_VALUE status
Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM
Example:

PC -> Reader: `52 46 45 01 0303 02 01 03 00 04 cs`
dataLength = 0x01 -> 1 Byte
mode = 0x00 -> ANTENNA_OFF

Reader -> PC: `52 46 45 01 0303 02 01 03 00 04 cs`
dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

7.17 Restore Factory Settings (03-20)

This function restores the factory settings. All settings are overwritten.
Return Values: RFE_RET_VALUE status
Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_COULD_NOT_WRITE
Example:

PC -> Reader: `52 46 45 01 0320 02 00 04 cs`
Reader -> PC: `52 46 45 01 0320 02 01 03 00 04 cs`
dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

7.18 Save Settings Permanent (03-21)

This function saves all settings permanently to the chip. Thus, the settings are the same after power off.
Return Values: RFE_RET_VALUE status
Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_COULD_NOT_WRITE
Example:

PC -> Reader: `52 46 45 01 0321 02 00 04 cs`
Reader -> PC: `52 46 45 01 0321 02 01 03 00 04 cs`
dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

7.19 Get Param (03-31)

Some readers have more settings than is possible to set with this protocol. With this function it is possible to read reader specific parameters. The address and the meaning of the fields can be found in the data sheet of the reader.
Parameters: unsigned short address
Return Values: RFE_RET_VALUE status, unsigned short size, unsigned char data[size]
Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM
Example:

PC -> Reader: `52 46 45 01 0331 02 02 03 0005 04 cs`
dataLength = 0x02 -> 2 Bytes
address = 0x0005
Reader -> PC: `52 46 45 01 0331 02 04 03 00 02 1785 04 cs`
dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS
size = 0x02 -> 2 Bytes
data = 0x1785

This is only an example! A valid address and data structure can be found in the data sheet of the reader!

7.20 Get GPIO Caps (05-01)

This command returns the capabilities of the reader's GPIO pins. It returns a bitmask each for the available pins, output pins and input pins.
Return Values: unsigned long mask, unsigned long output, unsigned long input
Example:

PC -> Reader: `52 46 45 01 0501 02 00 04 cs`
Reader -> PC: `52 46 45 01 0501 02 0C 03 0000FFFF 0000FFFF 0000FF00 04 cs`
dataLength = 0x0C -> 12 Bytes
mask = 0x0000FFFF -> 16 GPIO-pins (0 to 15)
output = 0x0000FFFF -> 16 Output-pins (0 to 15)
input = 0x0000FF00 -> 8 GPIO-pins (8 to 15)

7.21 Get GPIO Direction (05-02)

This command returns the current direction of the pin. If the direction bit for the pin is 1, the direction is output, if the bit is 0, the pin is input.
Return Values: unsigned long direction
Example:

PC -> Reader: `52 46 45 01 0502 02 00 04 cs`
Reader -> PC: `52 46 45 01 0502 02 04 03 0000 0FFF 04 cs`
dataLength = 0x04 -> 4 Bytes
direction = 0x00000FFF -> pins 0 to 11 are configured as output, pins 12 to 15 as input

7.22 Get GPIO (05-04)

This command returns the current level of the GPIO pins.
Return Values: unsigned long levelMask
Example:

PC -> Reader: `52 46 45 01 0504 02 00 04 cs`
Reader -> PC: `52 46 45 01 0504 02 04 03 0000 980F 04 cs`
dataLength = 0x04 -> 4 Bytes
levelMask = 0x0000980F -> input pins 15, 12 and 11 are currently high, output pins 0 to 3 are set to high, all else are currently low

7.23 Inventory Single (50-01)

This function can be used to make a single inventory round. The reader then returns a list of tags that are in its range. With some interfaces, the data length for transmission is limited, hence the return packet is separated into multiple packets if it would be too large. Therefore, two counters are transmitted, one that indicates how many tags were found and one that indicates how many tag ids are transmitted in this packet.
The reader can send more additional information instead of just the tag id. Therefore, the structure TagInfo is used. This structure has a variable length that is dependent of the information sent from the reader. The type of additional information is dependent of the used reader. These informations and the specific identifier can be found in the documentation of the reader.
The first byte of the structure indicates the length of the whole structure. After this, the tag id is sent. The Tag ID is split into a start byte, a length indicator and the id itself. After that, additional information can be added by the reader. In the example below, the RSSI value of the tag is added.

Length of TagInfo	1 Byte	
StartByte-Tag ID	1 Byte	0x01
ID-Length	1 Byte	
ID	ID-Length Bytes	
StartByte-RSSI	1 Byte	0x02
RSSI	2 Bytes	

Return Values: RFE_RET_VALUE status, unsigned char idCount, unsigned char packetIdCount, TagInfo tags[packetIdCount]
Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM
Example:
PC -> Reader: `52 46 45 01 5001 02 00 04 cs`
Reader -> PC: `52 46 45 01 5001 02 12 03 00 01 01 0E01 0C 0102030405060708090A0B0C 04 cs`
dataLength = 0x12 -> 18 Bytes
status = 0x00 -> RFE_RET_SUCCESS
idCount = 0x01 -> 1 ID in total
packetIdCount = 0x01 -> 1 ID in this packet
Length of TagInfo = 0x0E -> 14 Bytes
StartByte-TagID = 0x01
idLength = 0x0C -> 12 Bytes
TagID = 0x0102030405060708090A0B0C

7.24 Inventory Cyclic (50-02)

This function can be used to start and stop a cyclic inventory. With a cyclic inventory, the reader does inventories autonomously in a specified cycle. The timeout between such cycles can be found in the manual of the reader.
This function can be activated and deactivated.

enum INVENTORY_MODE (unsigned char)

INVENTORY_OFF	0x00
INVENTORY_ON	0x01

After the inventory is activated, the reader sends an answer with the status of the operation and interrupt messages are sent to the PC in a defined cycle.

Parameters: InventoryMode mode, (ulong time for turning on)

Return Values: RFE_RET_VALUE status

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM

Example:

PC -> Reader: `52 46 45 01 5002 02 05 03 01 000003E8 04 cs`

dataLength = 0x05 -> 5 Bytes

mode = 0x01 -> INVENTORY_ON

time = 0x000003E8 -> 1000 ms

Reader -> PC: `52 46 45 01 5002 02 01 03 00 04 cs`

dataLength = 0x01 -> 1 Byte

status = 0x00 -> RFE_RET_SUCCESS

7.25 Read From Tag (50-03)

With this function, data can be read from the memory of a tag. The meaning of the data, that can be read, can be found in the manual of the tag. On some tags, a memory bank must be specified. The memory banks are listed in the manual of the reader.

Parameters: unsigned char tagIdCount, unsigned char tagId[tagIdCount], unsigned char memoryBank, unsigned short startAddress, unsigned long accessPassword, unsigned char byteCount

Return Values: RFE_RET_VALUE status, unsigned char byteCount, unsigned char data[byteCount]

Status Values: RFE_RET_SUCCESS, RFE_RET_RESULT_PENDING, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM, Every TMI Return Code

Example:

PC -> Reader: `52 46 45 01 5003 02 15 03 0C 0102030405060708090A0B0C 01 0000 00000000 06 04 cs`

dataLength = 0x15 -> 21 Bytes

tagIdCount = 0x0C -> 12 Bytes

tagId = 0x0102030405060708090A0B0C

memoryBank = 0x01 -> second bank

startAddress = 0x0000

accessPassword = 0x00000000

bytesCount = 0x06 -> 6 Bytes

Reader -> PC: `52 46 45 01 5003 02 08 03 00 06 020002000200 04 cs`

dataLength = 0x08 -> 8 Bytes

status = 0x00 -> RFE_RET_SUCCESS

bytesCount = 0x06 -> 6 Bytes

data = 0x020002000200

7.26 Write To Tag (50-04)

With this function, data can be written to the memory of a tag. The meaning of the data that can be written, can be found in the manual of the tag. On some tags, a memory bank must be specified. The memory banks are listed in the manual of the reader.

Parameters: unsigned char tagIdCount, unsigned char tagId[tagIdCount], unsigned char memoryBank, unsigned short startAddress, unsigned long accessPassword, unsigned char byteCount, unsigned char data[byteCount]
Return Values: RFE_RET_VALUE status
Status Values: RFE_RET_SUCCESS, RFE_RET_RESULT_PENDING, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM, Every TMI Return Code
Example:

PC -> Reader: `52 46 45 01 5004 02 15 03 0C 0102030405060708090A0B0C 01 0000 00000000 06 020002000200 04 cs`
dataLength = 0x15 -> 21 Bytes
tagIdCount = 0x0C -> 12 Bytes
tagId = 0x0102030405060708090A0B0C
memoryBank = 0x01 -> second bank
startAddress = 0x0000
accessPassword = 0x00000000
bytesCount = 0x06 -> 6 Bytes
data = 0x020002000200

Reader -> PC: `52 46 45 01 5004 02 01 03 00 04 cs`
dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

7.27 Lock Tag (50-05)

With this function, the tag can be locked. The meaning of the mode and the memory space can be found in the documentation of the reader.
Parameters: unsigned char tagIdCount, unsigned char tagId[tagIdCount], unsigned char mode, unsigned char mem_space, unsigned long accessPassword
Return Values: RFE_RET_VALUE status
Status Values: RFE_RET_SUCCESS, RFE_RET_RESULT_PENDING, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM, Every TMI Return Code
Example:

PC -> Reader: `52 46 45 01 5005 02 14 03 0C 0102030405060708090A0B0C 0101 00000000 04 cs`
dataLength = 0x13 -> 19 Bytes
tagIdCount = 0x0C -> 12 Bytes
tagId = 0x0102030405060708090A0B0C
mode = 0x01
mem_space = 0x01
accessPassword = 0x00000000

Reader -> PC: `52 46 45 01 5005 02 01 03 00 04 cs`
dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

7.28 Kill Tag (50-06)

With this function, the tag can be killed.
Parameters: unsigned char tagIdCount, unsigned char tagId[tagIdCount], unsigned char rfuRecom, unsigned long killPassword
Return Values: RFE_RET_VALUE status

Status Values: RFE_RET_SUCCESS, RFE_RET_RESULT_PENDING, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM, Every TMI Return Code

Example:

PC -> Reader: `52 46 45 01 5006 02 12 03 0C 0102030405060708090A0B0C 00 00000000 04 cs`

dataLength = 0x12 -> 18 Bytes
tagIdCount = 0x0C -> 12 Bytes
tagId = 0x0102030405060708090A0B0C
rfuRecom = 0x00
killPassword = 0x00000000

Reader -> PC: `52 46 45 01 5006 02 01 03 00 04 cs`

dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

7.29 Custom Tag Command (50-10)

With this function, a custom tag command can be performed. The possible tag commands and the data structure that must be sent to the reader can be found in the documentation of the reader.

Parameters: unsigned char command, unsigned char tagIdCount, unsigned char tagId[tagIdCount], unsigned long accessPassword

Return Values: RFE_RET_VALUE status

Status Values: RFE_RET_SUCCESS, RFE_RET_ERR_ON_EXEC_OP, RFE_RET_ERR_WRONG_PARAM_COUNT, RFE_RET_ERR_WRONG_PARAM, Every TMI Return Code

Example:

PC -> Reader: `52 46 45 01 5010 02 12 03 01 0C 0102030405060708090A0B0C 00000000 04 cs`

dataLength = 0x12 -> 18 Bytes
command = 0x01
tagIdCount = 0x0C -> 12 Bytes
tagId = 0x0102030405060708090A0B0C
accessPassword = 0x00000000

Reader -> PC: `52 46 45 01 5010 02 01 03 00 04 cs`

dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

7.30 Heartbeat Interrupts (90-01)

The interrupt of a heartbeat is sent in a specified interval if the heartbeat is turned on.

Reader -> PC: `52 46 45 01 9001 02 01 03 00 04 cs`

dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_RET_SUCCESS

If a state heartbeat is selected, the heartbeat looks as follows.

Reader -> PC: `52 46 45 01 9001 02 02 03 00 10 04 cs`

dataLength = 0x02 -> 2 Bytes
status = 0x00 -> RFE_RET_SUCCESS
state = 0x10 -> RFE_STATE_SCANNING

7.31 Inventory Cyclic Interrupt (90-02)

The interrupt to the cyclic inventory is sent from the reader to the host with exactly one TagInfo. The TagInfo is built the same way as it is described in the response of a single inventory. The length of the TagInfo is not included because there will only be one TagInfo.

Reader -> PC: `52 46 45 01 9002 02 0E 03 01 0C 0102030405060708090A0B0C 04 cs`

dataLength = 0x0E -> 14 Bytes
StartByte-TagID = 0x01
idLength = 0x0C -> 12 Bytes
TagID = 0x0102030405060708090A0B0C

7.32 State Changed Interrupt (90-03)

The interrupt is sent every time the state changes. See the enum CURRENT_READER_STATE.

Reader -> PC: `52 46 45 01 9003 02 01 03 00 04 cs`

dataLength = 0x01 -> 1 Byte
status = 0x00 -> RFE_STATE_IDLE

7.33 Status Reg Changed Interrupt (90-04)

The interrupt is sent every time the reader detects an error.

Reader -> PC: `52 46 45 01 9004 02 08 03 0000 0000 0000 0000 04 cs`

dataLength = 0x08 -> 8 Bytes
statusRegister = 0x0000000000000000

7.34 Boot-Up Finished Interrupt (90-05)

Since v1.17, the reader sends "0x4F4B" (=OK) when boot-up finished.

7.35 Operation Result Interrupt (90-08)

The Interrupt is sent every time a pending operation finished and contains the data of the operation result.

Reader -> PC: `52 46 45 01 9008 02 06 03 23 00 03 02699A 04 cs`

dataLength = 0x06 -> 6 Bytes
pending-id = 0x23
status = 0x00 -> RFE_RET_SUCCESS
byteCount = 0x03 -> 3 Bytes
result data = 0x02699A

7.36 GPIO Values Changed Interrupt (90-09)

The interrupt is sent every time a GPIO input pin's value is changed.

Reader -> PC: `52 46 45 01 9009 02 04 03 00 00 98 0F 04 cs`

dataLength = 0x04 -> 4 Bytes
gpioValues = 0x0000980F -> Pins 15, 12 and 11 are high, pins 0 to 3 are high as well

7.37 Error Occurred Interrupt (90-FF)

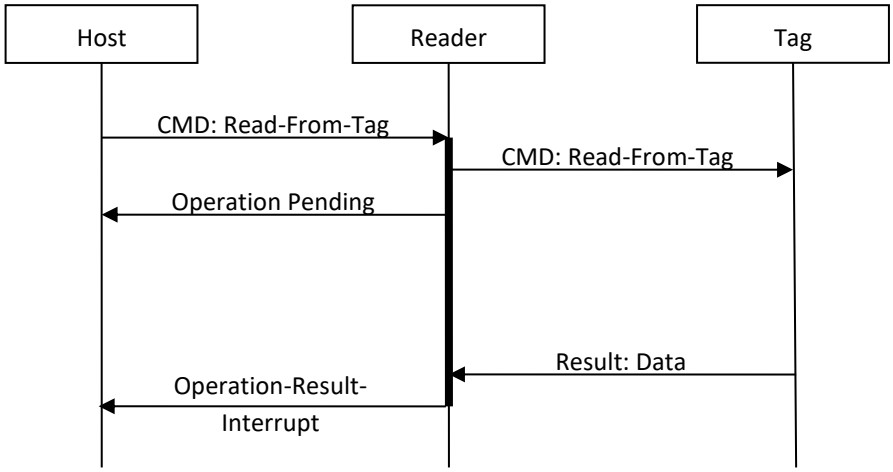
The interrupt is sent every time an error occurred. The meaning of the error codes and the attached data can be found in the documentation of the reader.

Reader -> PC: 52 46 45 01 90FF 02 02 03 01 02 04 cs
dataLength = 0x02 -> 2 Bytes
errorCode = 0x01 -> PUR: Too High RFP
antennaId = 0x02 -> Antenna #3

8 Pending Results

Some operations, tag operations especially, can last for quite a long time. To not get in doubt if the reader did not get the command or is not responding anymore, the return code RFE_RET_RESULT_PENDING was introduced. This return code should inform the application that the reader is working on the operation.

Example:
Read-From-Tag command
Message-Flow:



PC -> Reader: 52 46 45 01 5003 02 15 03 0C 0102030405060708090A0B0C 010000 00000000 06 04 cs
Reader -> PC: 52 46 45 01 5003 02 02 03 01 32 04 cs
dataLength = 0x02 -> 2 Bytes
status = 0x01 -> RFE_RET_RESULT_PENDING
pending-id = 0x32
Reader -> PC: 52 46 45 01 9008 02 09 03 32 00 06 020002000200 04 cs
dataLength = 0x09 -> 9 Bytes
pending-id = 0x32
status = 0x00 -> RFE_RET_SUCCESS
bytesCount = 0x06 -> 6 Bytes
data = 0x020002000200

9 Data Structures

Used data structures are collected in this chapter and shown in C syntax.

9.1 Enums

```
enum ANTENNA_POWER {
    ANTENNA_OFF = 0x00,
    ANTENNA_ON = 0x01
};
```

```
enum CURRENT_READER_STATE {
    RFE_STATE_IDLE = 0x00,
```

```
    RFE_STATE_REBOOTING = 0x01,
    RFE_STATE_SCANNING = 0x10,
    RFE_STATE_WRITING = 0x11,
    RFE_STATE_READING = 0x12
};
```

```
enum HEARTBEAT_SIGNAL {
    HEARTBEAT_OFF = 0x00,
    HEARTBEAT_ON = 0x01,
    HEARTBEAT_DUPLEX_ON = 0x02,
    HEARTBEAT_STATE_ON = 0x03,
    HEARTBEAT_DUPLEX_STATE_ON = 0x04
};
```

```
enum HOPPING_MODE {
    STATIC_UP = 0x00,
    RANDOM = 0x01
};
```

```
enum INVENTORY_MODE {
    INVENTORY_OFF = 0x00,
    INVENTORY_ON = 0x01
};
```

```
enum RFE_RET_VALUE {
    RFE_RET_SUCCESS = 0x00,
    RFE_RET_RESULT_PENDING = 0x01,
    RFE_RET_ERR_OP_NOT_SUPPORTED = 0x50,
    RFE_RET_ERR_UNKOWN_ERR = 0x51,
    RFE_RET_ERR_ON_EXEC_OP = 0x52,
    RFE_RET_ERR_COULD_NOT_WRITE = 0x53,
    RFE_RET_ERR_WRONG_PARAM_COUNT = 0x54,
    RFE_RET_ERR_WRONG_PARAM = 0x55,
    RFE_RET_TMI_TAG_UNREACHABLE = 0xA0,
    RFE_RET_TMI_MEM_OVERRUN = 0xA1,
    RFE_RET_TMI_MEM_LOCKED = 0xA2,
    RFE_RET_TMI_INSUFFICIENT_POWER = 0xA3,
    RFE_RET_TMI_WRONG_PASSWORD = 0xA4
};
```

```
enum eRFE_LOCK_MODE {
    UNLOCK = 0x00,
    LOCK = 0x01,
    PERMALOCK = 0x02,
    LOCK_AND_PERMALOCK = 0x03
```

```
};
```

```
enum eRFE_LOCK_MEMORY_SPACE {
    KILL_PASSWORD = 0x00,
    ACCESS_PASSWORD = 0x01,
    EPC = 0x02,
    TID = 0x03,
    USER = 0x04
};
```

9.2 Structs

```
struct MemBankData {
    unsigned char byteCount,
    unsigned char data[byteCount]
};
```

```
struct MemBankInfo {
    unsigned char memoryBank,
    unsigned short startAddress,
    unsigned long accessPassword,
    unsigned char byteCount
};
```

```
struct SequenceStruct {
    unsigned char antennaIndex,
    unsigned long activeTime
};
```