# Programming Guide

for metraTec QuasarMR1

Date: February 2015

Version: 1.1

# Table of Contents

## 1. The Protocol

Most RFID readers offered by metraTec RFID Solutions, such as the QuasarMR1 and QuasarSR, are based on SkyeTek's AURA Protocol (Version 2) and are 100% compatible to it. This manual is intended for programming these readers. You can check whether your reader is based on SkyeTek's AURA platform by reading its Operating Instructions. At the same time, you can use this instruction and its programming examples for all other readers based on this protocol, e.g. the SkyeTek RFID modules M1, M1 Mini and M0.

The protocol defines how data is exchanged between host (e.g. a PC) and the RFID Reader. It describes how the host communicates with the reader, how it can configure the reader and how it can make the reader read data from and write data on RFID tags and smart labels.

This manual describes the data exchange format used by reader and host to communicate data and commands. For better understanding it is suggested to read it in conjunction with the Operating Instructions of the RFID reader involved.

The data exchange protocol described here is only discussed on the data layer. Communication on the hardware layer depends on hardware settings and the type of hardware connection between RFID reader and host. Please check your Operating Instructions for more detailed information.

Furthermore, the protocol is available in an ASCII and a binary format. The host initiates every request and reply sequence. If the initiating command is issued in ASCII format the answer will also be in this format. The same is true for binary format.

Table 1 – ASCII Request and Reply

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Host:** | CR | Request | CR | | | | | | |
| **RFID Reader:** | | | | processes Request | | LF | Reply | CR | LF |

• CR corresponds to the ASCII-Code for Carriage Return

( CR = CHR$(13) = 0x0D [ENTER-Key] ).

• LF corresponds to the ASCII-Code for  Line Feed

( LF = CHR$(10) = 0x0A ).

• To allow any terminal program to communicate with any RFID reader each byte of a request or reply is transmitted as two HEX values – one for the lower nibble (half-byte), one for the upper nibble.

- Example: In case the "Flags"-Byte has the hexadecimal value of 0x1A the host transmits two characters: ASCII '1' followed by ASCII 'A'.

Table 2 – Binary Request and Reply

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Host:** | STX | Request | | | | | |
| **RFID Reader:** | | | | processes Request | | STX | Reply |

- STX corresponds to the ASCII-Code for start transmission (STX = CHR$(2) = 0x02).
- In case of binary transmission every byte has to be transmitted within 10 ms after the preceding byte as a delay of 10 ms indicates the end of transmission.

# 2. Explanation of the Request Format

░ mandatory fields

▓ optional fields

Table 3 – ASCII Request format

| Flags | Request | RID | Tag Type | TID | AFI | Starting Block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | 8 Bits | 64 Bits | 8 Bits | 8 Bits | 8 Bits | n*8 Bits | 16 Bits |

Table 4 – Binary Request format

| Request Length | Flags | Request | RID | Tag Type | TID | AFI | Starting Block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | 8 Bits | 8 Bits | 64 Bits | 8 Bits | 8 Bits | 8 Bits | n*8 Bits | 16 Bits |

## 2.1 Request Length Field (mandatory for binary format)

The Request Length field indicates of how many bytes the host request consists not counting the <STX> and the Field Length field.

Example: <STX> 0x05 0x20 0x14 0x00 0xF1CA

## 2.2 Flags Field (mandatory)

The bits of the Flags field set protocol and request options.

Table 5 – Flags field options

| RID_F | TID_F | CRC_F | AFI_F | RF_F | LOCK_F | INV_F | LOOP_F |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

Please remember that the upper four bits and the lower four bits are coded independently in one hexadecimal digit each. In case the TID_F bit, the CRC_F bit and the INV_F bit are set, the Flags field is set to 62 (4+2; 2). The eight Flags bits have to be seen as two sets of four bits each.

### 2.2.1 RID_F

RID_F = 1 The RID field is transmitted as part of a request to be able to address a certain reader in cases where there are several readers attached to one host.

RID_F = 0 The RID field is not included in the request.

### 2.2.2 TID_F

TID_F = 1 In case this flag is set, the serial number of the tag (TID) is part of the request. Since all tags available for the reader have TIDs this mechanism allows addressing only the desired tag. This flag only makes sense and should thus only be used with tag requests (where the Tag-bit is set, see Chapter 2.3).

In case a tag is addressed via SELECT_TAG using the TID the tag is put into selected mode. This mode stays active as long as the tag stays within the reader's RF field and as long as no other tag is addressed (with its respective TID). As long as a tag is in selected mode all commands are directed at that tag. This allows saving transmission time by not sending the TID with each tag request (by setting TID=0). To allow the selected mode to stay active the RF field has to be kept active (by setting RF_F=1) as otherwise the setting is forgotten when the tag is no longer powered (see Chapter 2.2.5).

TID_F = 0 The TID field is not part of the respective request. In case a tag is already in selected mode the selected mode stays active and that tag is automatically addressed.

### 2.2.3 CRC _F

CRC _F = 1 The CRC field is sent with the request allowing to verify whether the according request was transmitted correctly and completely. CRC_F has to be set in case of binary coding of the request and in case of system write commands (0x41 and 0x42).

CRC _F = 0 The CRC field is not transmitted as part of the request.

### 2.2.4 AFI_F

AFI_F = 1 The AFI field is transmitted with the respective Request (only possible with a SELECT_TAG request).

AFI_F = 0 The AFI field is not part of the request.

### 2.2.5 RF_F

RF_F = 1 The RF transmitter stays switched on after the request has been sent. This is important in cases in which tags have to remember current status information (e.g. "Stay Quiet" or selected mode).

RF_F = 0 Switches off the RF transmitter after the request has been processed.

### 2.2.6 LOCK_F

LOCK_F = 1 allows locking of data blocks on the tag. This flag only works with the WRITE_TAG request which is transmitted to the reader without Data field in this case. The "Number of Blocks" of data blocks starting at "Starting Block" are locked against further writing operations.

LOCK_F = 0 For all other requests and when WRITE_TAG is to be used to write data onto a tag.

### 2.2.7 INV_F

INV_F = 1 This bit is reserved for use with the SELECT_TAG and READ_TAG requests (see Chapters 4.4.1 and 4.4.3 – you will get an error message 0x82 if used with other requests, see Chapter 3.2). In cases in which this flag is set, the RFID reader will send a "Stay Quiet" command to the tag after having read its TID. As usual the reader will communicate the TID found to the host. This feature is called "anti collision" as it allows avoiding data collision in cases in which multiple tags are within the RF field. The INVentory mode allows serial reading of data or TIDs of many tags in the field. For this inventory mode to work, the RF field has to be powered between individual tag read operations so that the tags staying quiet remember to stay quiet. For this reason it is important that the RF_F-flag bit is set. In combination with the loop mode (see Chapter 2.2.8) the reader can automatically detect all tags entering the RF field and communicate their TIDs to the host.

INV_F = 0 Turning this flag off will suppress the "Stay Quiet" command from being sent to the tag after having been read. Under these conditions the SELECT_TAG and READ_TAG requests will return the data from the first tag found. All other requests require the flag to be turned off (as otherwise error message 0x82 will be returned, see Chapter 3.2).

### 2.2.8 LOOP_F

LOOP_F = 1 This flag bit is reserved for the use with the SELECT_TAG request (see Chapter 4.4.1 – will produce error message 0x82 when used with other requests, see Chapter 3.2). When the flag is set the SELECT_TAG request is repeated until the loop mode is terminated. A request with this flag set will first produce a reply that indicates the successful activation of the loop mode (0x1C, see Chapter 3.2). Following the successful activation reply the request will report any TIDs the reader finds in the RF field in a rapid succession. In case no tags are detected the reader will stay quiet until a tag is found.

LOOP_F = 0 This is the only possible flag setting for all other requests except SELECT_TAG. All other requests generate an error message 0x82, see Chapter 3.2 in case this flag is set differently. In case the bit is not set with the SELECT_TAG request the request will be executed only once and return the appropriate reply.

## 2.3 Request field (mandatory)

The request field specifies the type of request and the memory being addressed by the host request.

Table 6 – Request field

| **Request type** | | | | **Memory addressed by the request** | | | |
|---|---|---|---|---|---|---|---|
| Reserved<br>(set to 0) | Write_Bit | Read_Bit | Sel_Bit | Reserved<br>(set to 0) | Tag_Bit | Sys_Bit | Mem_Bit |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

Each request consists of exactly one request type and of exactly one request destination memory. The request consists consequently of two bit sets – the four bits of the upper nibble (half byte) and the four bits of the lower nibble.

The following table gives an overview over the requests supported by the protocol.

Table 7 – List of supported requests

| Value | Name |
|---|---|
| 0x14 | SELECT_TAG |
| 0x21 | READ_MEM |
| 0x22 | READ_SYS |
| 0x24 | READ_TAG |
| 0x41 | WRITE_MEM |
| 0x42 | WRITE_SYS |
| 0x44 | WRITE_TAG |

## 2.4 Tag Type Field (mandatory for tag requests)

The Tag Type field is required for all requests directed at the tag memory (that is in case Tag_Bit=1 in the request). The Tag Type field sets the type of RFID tag the reader is trying to communicate with. For a list of tag types please refer to Table 34.

## 2.5 RID Field

The RID field is used to define the reader that the host is addressing in situations where multiple readers are connected to one host. The RID field is only present in requests that have RID_F=1 set.

## 2.6 TID Field

The TID field contains the tag ID (serial number) in cases in which the reader is supposed to communicate exclusively with a certain tag. The field is present in the request only if TID_F=1 is set. The TID field only makes sense in case tags are addressed by the request.

## 2.7 AFI Field

AFI is the abbreviation for Application Field Identification. This field can only be used in conjunction with the SELECT_TAG request. ICode1 and ISO15693 tags support AFI. In case a request contains the AFI field only tags with matching AFIs will answer to the request.

## 2.8 The Starting Block Field

The Starting Block field sets the starting address for read and write operations. The Starting Block field is not used in SELECT requests. In case the reader's memory is being addressed a data block corresponds to one byte of data. In cases addressing tag memory the exact size and organization of the memory depend on tag type and can be found in Table 34.

## 2.9 Number of Blocks Field

The Number of Blocks field sets the number of data blocks to be read or written. This field is not used by any SELECT requests. In case of system memory operations it is always set to 0x01.

## 2.10 Data Field

The Data field contains the data to be written by the WRITE request. This field is never used in SELECT or READ requests.

## 2.11 CRC Field

The CRC (cyclic redundancy check) field is optional in cases in which the ASCII format is used. The field is present in case CRC_F=1 is set. In cases in which the request is sent in binary format and in case of system writing commands (0x41 and 0x42) CRC has to be used (and CRC_F=1 has to be set).

The CRC field has two bytes and has to be computed by the host dependent on the request fields present. The fields taken into account include all of the respective request or reply fields except start and stop bits (STX, CR, LF, etc.).

To compute the CRC value the CRC_CCITT polynomial (x16 + x12 + x5 + 1 = 0x8408) is used with a starting value of 0x0000.

Figure 1 – C implementation of the CRC computation

CRC Computation Example

```
// *dataP is a pointer to a byte array over which the CRC is to be
computed
// n is the number of bytes that *dataP points to
//
unsigned int crc16( unsigned char * dataP, unsigned char n )
{
unsigned char i, j; // Byte counter, Bit counter
unsigned int crc_16; // Variable for computed value
crc_16 = 0x0000; // Starting value
for (i = 0; i < n; i++) // Check every byte in the array
{
    crc_16 ^= *dataP++; //
    for (j = 0; j < 8; j++) // Check every bit in the respective byte
    {
          if(crc_16 & 0x0001 ) //
          {
                crc_16 >>= 1;
                crc_16 ^= 0x8408; // Polynome x^16 + x^12 + x^5 + 1
          }
          else
          {
                crc_16 >>= 1;
          }
    }
}
return( crc_16 ); // Returns computed CRC value (16 Bit)
}
```

A simple request (ASCII), to read the tag ID of an ISO15693 tag could be:

<CR><001401><CR>, with Flags=0x00, Request=0x14, Tag Type=0x01.

(without CRC)

In case CRC_F is set the request looks like this:

<CR><201401E043><CR>, with Flags=0x20, Request=0x14, Tag Type=0x01, CRC=0xE043.

A potential reply of the reader could be:

<LF><14E00700000147637A1AA2><CR><LF>, with reply code=0x14,

reply data=0xE00700000147637A and CRC=0x1AA2.

# 3. The Reply Field

Table 8 – ASCII Reply

| Reply Code | RID | Tag Type | Reply Data | CRC |
|---|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | n*8 Bits | 16 Bits |

Table 9 – Binary Reply

| Reply Length | Reply Code | RID | Tag Type | Reply Data | CRC |
|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | 8 Bits | n*8 Bits | 16 Bits |

In the ASCII case the only mandatory field is the Reply Code. The binary version also requires the Reply Length field.

All other fields are optional and their use depends on the value of the Reply Code field.

## 3.1 Reply Length Field (mandatory for binary coding)

The Reply Length field indicates how many bytes the reply consists of excluding the Reply Length byte and the <STX> byte.

This field is necessary in binary coding mode and not provided in ASCII mode. See the examples given in Chapter 5.

## 3.2 Reply Code Field (mandatory)

Table 10 – Reply Code

| Reply Code | Description |
|---|---|
| 0x14 | SELECT _TAG Success |
| 0x1C | SELECT _TAG LOOP Activated |
| 0x94 | SELECT _TAG Failure |
| 0x9C | SELECT _TAG LOOP Terminated |
| 0x21 | READ_MEM Success |
| 0x22 | READ_SYS Success |
| 0x24 | READ_TAG Success |
| 0XA1 | READ_MEM Failure |
| 0xA2 | READ_SYS Failure |
| 0xA4 | READ_TAG Failure |
| 0x32 | EVENT_Report |
| 0xC2 | EVENT_Error |
| 0x41 | WRITE_MEM Success |
| 0x42 | WRITE_SYS Success |
| 0x44 | WRITE_TAG Success |
| 0xC1 | WRITE_MEM Failure |
| 0xC2 | WRITE_SYS Failure |
| 0xC4 | WRITE_TAG Failure |
| 0x80 | Non-ASCII Character in Request |
| 0x81 | CRC not valid |
| 0x82 | Flags don’t fit with Request |
| 0x83 | Flags don’t fit with Tag Type |
| 0x84 | Unknown Request |
| 0x85 | Unknown Tag Type |
| 0x86 | Invalid Starting Block |
| 0x87 | Invalid Number of Blocks |
| 0x88 | Invalid Request Length |

## 3.3 RID Field

The RID field is only present if RID_F = 1 was set in the request’s Flags field. It contains the ID of the reader the reply came from.

## 3.4 Tag Type Field

The Tag Type field is only present in a reply in case a SELECT_TAG request led to the reply which had its Tag Type field set to Auto (0x00). Table 34 in Chapter 6.6 shows a list of tag type codes for different tag types.

## 3.5 Reply Data Field

The Reply Data field is present in replies to host requests asking for data such as tag IDs, tag block data or system parameter data.

## 3.6 CRC Field

The CRC (cyclic redundancy check) field is present in any reply whose request had the CRC_F flag set.

To determine the CRC for the reply the same algorithm is used as in the request (see Chapter 2.11). In that example the reader had replied with <LF><14E00700000147637A1AA2><CR><LF>. The host has two ways of validating the completeness and correctness of the reply:

### 3.6.1 Computing the CRC

The host simply computes the CRC from the remaining data transmitted (0x14E00700000147637A). In case it computes the same value as was transmitted (0x1AA2) the transmission was without errors.

### 3.6.2 Computing of 0x0000 from the Reply-CRC with LSByte first

The host verifies the reply by computing the CRC from 0x14E00700000147637AA21A and receiving 0x0000 as the answer. Please note that the order of the CRC bytes has been reversed as the reply was transmitted MSByte first whereas they need to be used LSByte first in the CRC computation.

# 4. Description of the Requests

## 4.1 Reader Memory Organization

The RFID reader has 256 bates of EEPROM memory. Part of this memory is used for system parameters. System requests allow reading and setting these parameters in effect changing the way the reader behaves. In general there are two ways of changing the parameters. The first one is by changing them in the permanent system memory (request 0x41). Changes made here only take effect after power cycling the reader as at boot time the contents of permanent memory are copied to volatile memory. The second way for changing system parameters is to change them directly in volatile system memory (request 0x42) where the changes take immediate effect but are forgotten when the power is turned off. You can find examples of both types of changes in Chapters 5.15 to 5.22.

The following table shows how system memory is organized.

Table 11 – Organization of system memory

| Name | Address | Value Range | Factory Setting | Defines | Readable | Writable |
|---|---|---|---|---|---|---|
| Serial Number | 0x00 | 0x00000000-0xFFFFFFFF | Individual | Serial Number | individual | No |
| Firmware Version | 0x01 | 0x0000-0xFFFF | Release dependent | Firmware-Version | Yes | No |
| Reader ID (RID) | 0x02 | 0x00-0xFF | 0xFF (“No RID”) | Reader Network ID | Yes | Yes |
| Baud Rate | 0x03 | 0xFF<br>0x00<br>0x01<br>0x02<br>0x03<br>0x04-0xFE | <br>0x00 | 4800<br>9600<br>19200<br>38400<br>57600<br>Reserved | No | Yes |
| Operating Status | 0x04 | 0x00<br>0x01-0xFF | Not applicable | Sleep Mode<br>Activated | No | Yes |
| Reserved | 0x05-0x06 | | | | No | No |
| I/O-Pin Direction | 0x07 | 0x00-0xFF | 0x00 | Defines, whether I- or O-Pin | Yes | Yes |
| I/O Values | 0x08 | | | Writes O-values, Reads I-Values | Yes | Yes |
| Reserved | 0x09-0x11 | | | | No | No |
| Startup Request | 0x12 | special | 0x00 | Special | No | Yes |

## 4.2 Requests that Access Runtime System Parameters

The reader uses volatile runtime system memory to store current system parameter values. These can be read using request 0x22 and can be written using request 0x42. The addresses that can be written to / read from are shown in Table 11. Any changes made using request WRITE_SYS (0x42) take immediate effect but are lost when rebooting the system.

### 4.2.1 READ_SYS

The host reads data from runtime system memory of the RFID reader.

Table 12 – Request options for READ_SYS

| Request Length | Flags | Request | RID | Starting Block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 0x22 | 8 Bits | 8 Bits | 0x01 | n*8 Bits | 16 Bits |

The Starting Block field defines at which address the read request starts reading data. What can be found at which address can be seen in Table 11.

The Number of Blocks field defines how many system parameters are read by the request.

The Data field is only used in special system requests.

Table 13 – Flags of the READ_SYS request

| RID_F | | CRC_F | | | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

In case RID_F=1, the optional RID field is present in the request. This is useful in environments with multiple readers attached to one host.

If CRC_F=1, the optional CRC field is part of the request.

Table 14 – Reply to READ_SYS

| Reply Length | Reply Code | RID | Reply Data | CRC |
|---|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | N*8 Bits | 16 bits |

In case the Reply Code field indicates a successful READ_SYS request (Reply Code = 0x22) the Reply Data field contains Number of Blocks bytes of system parameter data that were read from volatile runtime memory. The interpretation of what the individual data bytes mean can be found in Table 11.

In case the Reply Code field indicates an error the Reply Data field is not present in the reply.

In case that the CRC_F flag was set in the request the CRC is also present in the reply.

If the READ_SYS request had the RID_F flag bit set the RID field is part of the reply.

### 4.2.2 WRITE_SYS

The Host writes data into runtime volatile memory of the RFID reader.

Table 15 – Request options for WRITE_SYS

| Request Length | Flags | Request | RID | Starting Block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 0x42 | 8 Bits | 8 Bits | 0x01 | n*8 Bits | 16 Bits |

The Starting Block field defines where the writing of system parameters into volatile runtime memory should start. The address space for this is shown in Table 11. Please keep in mind that writing into volatile runtime memory of the reader will produce immediate results but will not outlast a reboot of the system.

The Number of Blocks field defines the number of system parameters to write.

The Data field contains the values to be written into volatile runtime system memory. It has to contain Number of Blocks bytes.

The CRC field is always required in WRITE_SYS requests.

Table 16 – Flags of the WRITE_SYS request

| RID_F | | CRC_F | | | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

If RID_F=1 is set the optional RID field is part of the request to address a certain reader with the write request in networks containing multiple readers.

It is mandatory to set CRC_F to 1. For this reason the CRC field will always be included in a WRITE_SYS request.

Table 17 – Reply to WRITE_SYS

| Reply Length | Reply Code | RID | CRC |
|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | 16 Bits |

A Reply Data field is not part of a reply to the WRITE_SYS request.

Since CRC is compulsory in the request it will also always be included in the reply.

If RID_F was set in the request, the RID field will be part of the reply.

## 4.3 Access System Parameters in Permanent Memory

Besides volatile runtime memory the reader also has permanent nonvolatile memory for storing system parameters. This memory is read using request 0x21 and it is written to using request 0x41. The addressing of the memory is the same as in the volatile memory case and can be seen in Table 11. Changes made to parameters in non-volatile memory will take effect only after rebooting the reader.

### 4.3.1 READ_MEM

The host reads data from reader memory. For the addressing of the data and the interpretation of data read please refer to Table 11. In the same table you will find data on whether a certain address in reader memory can be read at all.

Table 18 – Request options for READ_MEM

| Request Length | Flags | Request | RID | Starting Block | Number of Blocks | CRC |
|---|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 0x21 | 8 Bits | 8 Bits | 0x01 | 16 Bits |

The Starting Block field defines the memory address from where reading starts.

The Number of Blocks field sets the number of memory blocks to be read.

Table 19 – Flags of the READ_MEM request

| RID_F | | CRC_F | | | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

In case RID_F=1, the optional RID field is present in the request. This is useful in environments with multiple readers attached to one host.

If CRC_F=1, the optional CRC field is part of the request.

Table 20 – Reply to a READ_MEM request

| Reply Length | Reply Code | RID | Reply Data | CRC |
|---|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | n*8 Bits | 16 Bits |

In case the Reply Code field indicates a successful READ_MEM request (Reply Code = 0x21) the Reply Data field contains Number of Blocks bytes of system parameter data that were read from permanent memory. The interpretation of what the individual data bytes mean can be found in Table 11.

In case the Reply Code field indicates an error the Reply Data field is not present in the reply.

In case that the CRC_F flag was set in the request the CRC is also present in the reply.

If the READ_MEM request had the RID_F flag bit set the RID field is part of the reply.

### 4.3.2 WRITE_MEM

The host writes data into permanent system parameter memory of the reader. For questions regarding addressing please refer to Table 11. In that table you will also find information on which memory addresses are writable.

Table 21 – Request options for WRITE_MEM

| Request Length | Flags | Request | RID | Starting Block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 0x41 | 8 Bits | 8 Bits | 8 Bits | n*8 Bits | 16 Bits |

The Starting Block field defines where the writing of system parameters into permanent system memory should start. Please keep in mind that changes written into permanent system memory of the reader will only become active after rebooting the reader.

The Number of Blocks field defines the number of system parameters to write.

The Data field contains the values to be written into volatile runtime system memory. It has to contain Number of Blocks bytes.

The CRC field is always required in WRITE_MEM requests.

Table 22 – Flags for WRITE_MEM request

| RID_F | | CRC_F | | | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

If RID_F=1 is set the optional RID field is part of the request to address a certain reader with the write request in networks containing multiple readers.

It is mandatory to set CRC_F to 1. For this reason the CRC field will always be included in a WRITE_MEM request.

Table 23 – Reply to WRITE_MEM

| Reply Length | Reply Code | RID | CRC |
|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | 16 Bits |

A Reply Data field is not part of a reply to the WRITE_MEM request.

Since CRC is compulsory in the request it will also always be included in the reply.

If RID_F was set in the request, the RID field will be part of the reply.

## 4.4 Description of Requests Directed at Tags

The tag requests interact with one or multiple tags of the defined tag type in the RF field of the RFID reader.

### 4.4.1 SELECT_TAG

The SELECT_TAG request is very versatile in its effect depending on the bits set in the Flags field.

Table 24 – Options for SELECT_TAG

| Request Length | Flags | Request | RID | Tag Type | TID | AFI | CRC |
|---|---|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 0x14 | 8 Bits | 8 Bits | 64 Bits | 8 Bits | 16 Bits |

Table 25 – Flags for SELECT_TAG

| RID_F | TID_F | CRC_F | AFI_F | RF_F |  | INV_F | LOOP_F |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

If CRC_F=1, the optional CRC field is part of the request.

If RID_F=1 is set the optional RID field is part of the request to address a certain reader with the write request in networks containing multiple readers.

Table 26 – Reply to SELECT_TAG request

| Reply Length | Reply Code | RID | Tag Type | Reply Data | CRC |
|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | 8 Bits | n*8 Bits | 16 Bits |

In case the reply indicates a successful SELECT_TAG request (by a return code of 0x14) the Reply Data field contains the unique TID. The size of the TID depends on tag type and version.

If the request had set Tag Type to Auto (0x00) the Tag Type field is present in the reply and indicates the tag type of the tag found.

If RID_F was set in the request, the RID field will be part of the reply.

## 4.4.2 WRITE_TAG

The WRITE_TAG request writes data onto tags.

Table 27 – Request options for WRITE_TAG

| Request Length | Flags | Request | RID | Tag Type | TID | Starting Block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 0x44 | 8 Bits | 8 Bits | 64 Bits | 8 Bits | 8 Bits | n*8 Bits | 16 Bits |

Table 28 – Flags for the WRITE_TAG request

| RID_F | TID_F | CRC_F | | RF_F | LOCK_F | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

In case RID_F=1, the optional RID field is present in the request. This is useful in environments with multiple readers attached to one host.

In case TID_F=1, the optional TID field is present in the request to define, which tag to write to.

If LOCK_F=1, the Data field is not used.

The Starting Block field defines the address of the block where writing starts.

The Number of Blocks field sets the number of tag blocks to be written.

In the Data field the data to be written onto the according tag blocks is stored.

In case the Number of Blocks field specifies that more than one data block is to be written to the tag, the data in the Data field is organized such that the first block of data to be written is at the beginning of the data field.

Table 29 – Reply to WRITE_TAG

| Reply Length | Reply Code | RID | CRC |
|---|---|---|---|
| 8 Bits | 8 Bits | 8 Bits | 16 Bits |

A Reply Data field is not part of the reply to a WRITE_TAG request.

If RID_F was set in the request, the RID field will be part of the reply.

## 4.4.3 READ_TAG

The Host reads data from tags present in the RF field of the RFID Reader.

Table 30 – Request option for READ_TAG

| Request Length | Flags | Request | RID | Tag Type | TID | Starting Block | Number of Blocks | CRC |
|---|---|---|---|---|---|---|---|---|
| 8 Bits | 8 Bits | 0x24 | 8 Bits | 8 Bits | 64 Bits | 8 Bits | 8 Bits | 16 Bits |

Table 31 – Flags for the READ_TAG request

| RID_F | TID_F | CRC_F | | RF_F | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

In case RID_F=1, the optional RID field is present in the request. This is useful in environments with multiple readers attached to one host.

In case TID_F=1, the optional TID field is present in the request to define, which tag to read from

If CRC_F=1, the optional CRC field is part of the request.

If RF_F=1, the RF reader does not switch off the RF field after completing the READ_Tag request.

Table 32 – Reply to READ_TAG

| Reply Length | Reply Code | RID | Reply Data | CRC |
|---|---|---|---|---|
| 8 Bits | 8 Bit | 8 Bit | n*8 Bit | 16 Bits |

In cases in which the Reply Code field indicates a successful READ_TAG request (Reply Code=0x24) the Reply Data field contains Number of Blocks of data which was successfully read from the tag. The data block size depends on tag type and tag version and can be found in Table 34.

If the Reply Code field shows an error code, the Reply Data field is not part of the reply.

If RID_F was set in the request, the RID field will be part of the reply.

# 5. Examples

After describing the general composition and syntax of the requests in Chapters 1 to 4 this chapter is intended to give some application examples that show how the different requests can be used in practice when programming your own interface.

## 5.1 Read the Tag Serial Number (TID) of One Tag

To read TIDs in general the SELECT_TAG request (0x14) is used. To read just one tag, the INV_F flag is not set. Under these conditions only one TID is read – in case several tags are within the RF field of the reader only the tag answering first is read.

### 5.1.1 Read TID Independently of Tag Type (Auto-Detect)

If the tag type to be read is not set by the host (Auto-Detect mode) the read request returns the reply code, the tag type and the TID.

#### 5.1.1.1ASCII Version

Request:

| | Flags | Request | Tag Type | |
|---|---|---|---|---|
| <CR> | 00 | 14 | 00 | <CR> |

Reply:

| | Reply Code | Tag Type | Data (TID) | |
|---|---|---|---|---|
| <LF> | 14 | 01 | E0 07 00 00 01 64 5E 37 | <CR><LF> |

In this case a tag of type ISO 15693 was found.

#### 5.1.1.2Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | CRC |
|---|---|---|---|---|---|
| <STX> | 0x05 | 0x20 | 0x14 | 0x00 | 0x9F9D |

Reply:

| | Reply Length | Reply Code | Tag Type | Data (TID) | CRC |
|---|---|---|---|---|---|
| <STX> | 0x0C | 0x14 | 0x02 | 0x01000000094B3E5<br>1 | 0x2379 |

In this case an ICode-SL1 tag was found.

### 5.1.2 Read TID for a Given Tag Type

In some cases only tags of one tag type are to be read. In such a case the hosts sets the tag type directly instead of setting it to Auto-Detect (0x00) mode (see Table 34, Chapter 6.6). Under these conditions the reply does not contain the Tag Type field as it is already known to the host.

#### 5.1.2.1ASCII Version

Request:

| | Flags | Request | Tag Type | |
|---|---|---|---|---|
| <CR> | 00 | 14 | 06 | <CR> |

As the Tag Type is set to 06 the reader will only search for PicoTags.

Reply:

| | Reply Code | Data (TID) | |
|---|---|---|---|
| <LF> | 14 | 00 0C 00 00 00 2B 5B A4 | <CR><LF> |

#### 5.1.2.2Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | CRC |
|---|---|---|---|---|---|
| <STX> | 0x05 | 0x20 | 0x14 | 0x04 | 0xD9B9 |

As the Tag Type field is set to 04 the reader will only search for ISO 14443-A tags.

Reply:

| | Reply Length | Reply Code | Data (TID) | CRC |
|---|---|---|---|---|
| <STX> | 0x07 | 0x14 | 0x710C8765 | 0x93B3 |

## 5.2 Read the Tag IDs of all Tags in the RF Field

To read the TIDs of all tags in the reader’s RF field the INV_F bit has to be set to activate inventory mode. This deactivates tags temporarily after having read their TID to allow for reading other tags without being interrupted („anti collision mode“).

Since the protocol supports tags of different standards the Auto Detect setting allows reading tags of different tag types when they are present in the field.

### 5.2.1 Read the TID of all tags (independent of Tag Type)

In case the Tag Type field is set to Auto Detect (0x00) the replies to the request will contain the Tag Type field as part of the reply as well as the TID. When all tags that are present in the RF field have been found and their data reported the final reply to the request has a Reply Code of 0x94 to show the end of the inventory read. This implies that the LOOP_F bit is not set.

### 5.2.1.1ASCII Version

Request:

| | Flags | Request | Tag Type | |
|---|---|---|---|---|
| <CR> | 02 | 14 | 00 | <CR> |

Please note that the INV_F bit is set.

Reply:

| | Reply Code | Tag Type | Data (TID) | |
|---|---|---|---|---|
| <LF> | 14 | 01 | E0 07 00 00 01 64 5E 37 | <CR><LF> |
| <LF> | 14 | 01 | E0 07 00 00 01 54 65 31 | <CR><LF> |
| <LF> | 14 | 01 | E0 07 00 00 01 54 41 32 | <CR><LF> |
| <LF> | 14 | 02 | 01 00 00 00 33 B1 DF 8E | <CR><LF> |
| <LF> | 14 | 02 | 01 00 00 00 02 5D CA D2 | <CR><LF> |

In this example three ISO15693 and two I-Code SL1 tags were found.

| | | |
|---|---|---|
| <LF> | 94 | <CR><LF> |

This last line of the reply indicates that no further tags were found.

## 5.2.2 Read the TID of all Tags of One Type

If the Tag Type field is not set to Auto Detect the replies will not contain the Tag Type field and will only report the TIDs of tags of the tag type set. As soon as all tags of that type have been read the reader terminates the inventory read by reporting Reply Code 0x94. Again, this is for applications where the LOOP_F bit is not set.

### 5.2.2.1ASCII Version

Request:

| | Flags | Request | Tag Type | |
|---|---|---|---|---|
| <CR> | 02 | 14 | 01 | <CR> |

Please note that the INV_F bit is set. The Tag Type is set such that only ISO 15693 tags are found.

Reply:

| | Reply Code | Data (TID) | |
|---|---|---|---|
| <LF> | 14 | E0 07 00 00 01 64 5E 37 | <CR><LF> |
| <LF> | 14 | E0 07 00 00 01 54 65 31 | <CR><LF> |
| <LF> | 14 | E0 07 00 00 01 54 41 32 | <CR><LF> |

| | | |
|---|---|---|
| <LF> | 94 | <CR><LF> |

The last line states that no further tags of the defined type have been found.

## 5.3 Read the TIDs of all Tags Continuously

If the LOOP_F bit is set the reader will reply with a Reply Code of 0x1C to indicate the activation of the loop mode. After this initial reply it will give additional replies whenever it detects a new TID.

### 5.3.1 Read the TIDs of all Tags Independently of Tag Type

If Tag Type is set to Auto Detect the replies to the request will contain the Tag Type field to report the tag type of all tags found. The replies will also contain the TID field to report the tag serial numbers. The inventory mode is deactivated (INV_F bit is not set) and the loop mode is activated (LOOP_F bit is set).

### 5.3.1.1ASCII Version

Request:

| | Flags | Request | Tag Type | |
|---|---|---|---|---|
| <CR> | 01 | 14 | 00 | <CR> |

Please note that the LOOP_F bit is set. The tag type is set to Auto Detect.

Reply:

| | Reply Code | |
|---|---|---|
| <LF> | 1C | <CR><LF> |

This first part of the reply indicates that loop mode has been successfully activated.

| | Reply Code | Tag Type | Data (TID) | |
|---|---|---|---|---|
| <LF> | 14 | 01 | E0 07 00 00 01 64 5E 37 | <CR><LF> |
| <LF> | 14 | 01 | E0 07 00 00 01 64 5E 37 | <CR><LF> |
| <LF> | 14 | 01 | E0 07 00 00 01 64 5E 37 | <CR><LF> |

As soon as a tag enters into the RF field the reader will continuously send its tag type (in this case ISO15693) and TID. When a second tag enters into the RF field in such a way that it is preferentially read the new tag's tag type and TID are repeatedly reported:

| | Reply Code | Tag Type | Data (TID) | |
|---|---|---|---|---|
| <LF> | 14 | 02 | 01 00 00 00 02 5D CA D2 | <CR><LF> |
| <LF> | 14 | 02 | 01 00 00 00 02 5D CA D2 | <CR><LF> |

In this case the reply indicates an I-Code SL1 tag. If at this point another tag is entered into the reader's RF field and the previous two are removed the further replies could be:

| | Reply Code | Tag Type | Data (TID) | |
|---|---|---|---|---|
| <LF> | 14 | 0A | 04 A6 8D 11 12 7A 00 | <CR><LF> |
| <LF> | 14 | 0A | 04 A6 8D 11 12 7A 00 | <CR><LF> |

In this case a tag using MIFARE® Ultralight® protocol is detected.

To terminate loop mode the host just has to send one or multiple bytes to the reader. This could be:

Request:

|  | Flags | Request | Tag Type |  |
|---|---|---|---|---|
| <CR> | 00 | 14 | 00 | <CR> |

Reply:

|  |  |  |
|---|---|---|
| <LF> | 9C | <CR><LF> |

The request sent to the reader terminated loop mode which is indicated in the Reply Code field. The actual request sent is ignored.

## 5.3.2 Continuously Read TIDs of Tags of a Given Type

If Tag Type is not set to Auto Detect the replies do not contain the Tag Type field as the type is already known. The replies only report TIDs of tags of the correct tag type found. The inventory mode is deactivated (INV_F bit is not set), the loop mode is activated (LOOP_F bit is set).

### 5.3.2.1ASCII Version

Request:

|  | Flags | Request | Tag Type |  |
|---|---|---|---|---|
| <CR> | 01 | 14 | 01 | <CR> |

Please note that the LOOP_F bit is set. The Tag Type is set to ISO 15693 tags.

Reply:

|  | Reply Code |  |
|---|---|---|
| <LF> | 1C | <CR><LF> |

This first part of the reply confirms that loop mode was activated successfully.

| | Reply Code | Data (TID) | |
|---|---|---|---|
| <LF> | 14 | E0 07 00 00 01 64 5E 37 | <CR><LF> |
| <LF> | 14 | E0 07 00 00 01 64 5E 37 | <CR><LF> |
| <LF> | 14 | E0 07 00 00 01 64 5E 37 | <CR><LF> |

As soon as a tag enters the RF field of the reader the reader repeatedly sends its TID to the host. If no tag is within reach of the reader it stops sending data. If a second tag enters the RF field its TID is then read and sent to the host repetitively.

| | Reply Code | Data (TID) | |
|---|---|---|---|
| <LF> | 14 | E0 07 00 00 01 54 65 31 | <CR><LF> |
| <LF> | 14 | E0 07 00 00 01 54 65 31 | <CR><LF> |

To terminate the loop mode the host just has to send one or more bytes to the reader. This could e.g. be the following:

Request:

| | Flags | Request | Tag Type | |
|---|---|---|---|---|
| <CR> | 00 | 14 | 01 | <CR> |

Reply:

| | | |
|---|---|---|
| <LF> | 9C | <CR><LF> |

The actual request is ignored and the reader sends the Reply Code indicating termination of loop mode.

## 5.4 Read the TID of all Tags in the RF Field Continuously

To read the TID of all tags in the RF field of the reader both the LOOP_F and the INV_F bits are set. This way the reader reads each tag once and then sends a „stay quiet" command to the tag so that it doesn't interfere with the reader reading the next tag's ID. As long as the tags stay within the RF field which powers them they will only be read once. If, however, a tag is removed from the field and it looses power, it looses the "stay quiet" mode and when re-entered into the field will be read again. To detect all tag types the Tag Type field is set to Auto Detect (0x00).

## 5.4.1 ASCII Version

Request:

| | Flags | Request | Tag Type | |
|---|---|---|---|---|
| <CR> | 03 | 14 | 00 | <CR> |

Please not that the LOOP_F and INV_F bits are set. The tag type is set to Auto Detect (0x00).

Reply:

| | Reply Code | |
|---|---|---|
| <LF> | 1C | <CR><LF> |

| | Reply Code | Tag Type | Data (TID) | |
|---|---|---|---|---|
| <LF> | 14 | 01 | E0 07 00 00 01 64 5E 37 | <CR><LF> |
| <LF> | 14 | 01 | E0 07 00 00 01 64 3D 21 | <CR><LF> |

This first part of the reply confirms that loop mode was activated successfully.

The TIDs of the two ISO 15693 tags present in the RF field of the reader are reported back once each. If additional tags are brought into the field their TIDs are reported once:

| | Reply Code | Tag Type | Data (TID) | |
|---|---|---|---|---|
| <LF> | 14 | 02 | 01 00 00 00 05 CA 5D E2 | <CR><LF> |
| <LF> | 14 | 03 | 01 32 1F A7 | <CR><LF> |

If one of the tags present in the field is removed from it and then re-enters the field, its TID is reported again:

| | Reply Code | Tag Type | Data (TID) | |
|---|---|---|---|---|
| <LF> | 14 | 01 | E0 07 00 00 01 64 5E 37 | <CR><LF> |

To terminate the loop mode the host just has to send one or more bytes to the reader. This could e.g. be the following:

Request:

| | Flags | Request | Tag Type | |
|---|---|---|---|---|
| <CR> | 00 | 14 | 00 | <CR> |

Reply:

| | | |
|---|---|---|
| <LF> | 9C | <CR><LF> |

The actual request is ignored and the reader sends the Reply Code indicating termination of loop mode.

## 5.5 Putting a Tag into Selected Mode

In this example a tag is put into selected mode to avoid having to transmit the TID with every write and read command. To activate this mode the SELECT_TAG request is sent to the tag with the TID_F bit activated and the tag's TID as part of the request. If a tag with matching TID is present in the RF field of the reader the reader will reply with a Reply Code of 0x14 (select tag success), otherwise with a Reply Code of 0x94 (select tag failure).

To keep the selected mode activated the RF field of the reader has to stay powered. This is achieved by setting the RF_F bit.

### 5.5.1 ASCII Version

Request:

| | Flags | Request | Tag Type | TID | |
|---|---|---|---|---|---|
| <CR> | 48 | 14 | 02 | 01 00 00 00 05 CA 5D E2 | <CR> |

In this case the reader will reply with a Reply Code of 0x14 in case an I-Code SL1 tag with the given TID is present in the RF field of the reader.

Reply:

| | Reply Code | |
|---|---|---|
| <LF> | 14 | <CR><LF> |

An appropriate tag was found and put into selected mode.

### 5.5.2 Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | TID | CRC |
|---|---|---|---|---|---|---|
| <STX> | 0x0D | 0x68 | 0x14 | 0x01 | 0xE00401000EE68E7B | 0x7356 |

In this case the reader will reply with a Reply Code of 0x14 in case an ISO15693 tag with the given TID is present in the RF field of the reader.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x14 | 0x7CCD |

An appropriate tag was found and put into selected mode.

## 5.6 Reading Data from a Tag Using its TID

In this example a block of data is read from a tag whose TID was previously determined e.g. by using SELECT_TAG. The host has to supply a valid Starting Block number and a valid Number of Blocks. The Number of Blocks is set to one for this example.

## 5.6.1 ASCII Version

Request:

| | Flags | Request | Tag Type | TID | Starting Block | Number of Blocks | |
|---|---|---|---|---|---|---|---|
| <CR> | 40 | 24 | 01 | E0 07 00 00 01 64 5E 37 | 00 | 01 | <CR> |

In this case a block of data is read from an ISO 15693 tag starting from block 0x00.

Reply:

| | Reply Code | Data | |
|---|---|---|---|
| <LF> | 24 | 11 22 33 44 | <CR><LF> |

The request was successful and the data is transmitted to the host.

## 5.6.2 Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | TID | Starting Block | Number of Blocks | CRC |
|---|---|---|---|---|---|---|---|---|
| <STX> | 0x0F | 0x68 | 0x24 | 0x02 | 0x010000000095B3E51 | 0x05 | 0x01 | 0x8EFD |

In this case a block of data is read from an I-CODE SL1 tag starting at block 0x05.

Reply:

| | Reply Length | Reply Code | Data | CRC |
|---|---|---|---|---|
| <STX> | 0x07 | 0x24 | 0x69696969 | 0xE40E |

The request was successful and the data is transmitted to the host.

## 5.7 Reading Data from a Tag in Selected Mode

In this example a block of data is read from a tag that was previously put into selected mode via SELECT_TAG. Because the tag is in selected mode the TID does not have to be sent to the reader. To keep the selected mode activated the RF_F bit has to be set in all requests. The host has to supply a valid Starting Block number and a valid Number of Blocks. The Number of Blocks is set to one for this example.

### 5.7.1 ASCII Version

Request:

| | Flags | Request | Tag Type | Starting Block | Number of Blocks | |
|---|---|---|---|---|---|---|
| <CR> | 08 | 24 | 03 | 05 | 01 | <CR> |

In this case a block of data is read from a Tag-It HF tag starting at block 0x05.

Reply:

| | Reply Code | Data | |
|---|---|---|---|
| <LF> | 24 | BA DF AC E0 | <CR><LF> |

The request was successful and the data is transmitted to the host.

### 5.7.2 Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | Starting Block | Number of Blocks | CRC |
|---|---|---|---|---|---|---|---|
| <STX> | 0x07 | 0x28 | 0x24 | 0x0A | 0x07 | 0x01 | 0xF424 |

In this case a block of data is read from a tag using MIFARE Ultralight protocol starting at block 0x07.

Reply:

| | Reply Length | Reply Code | Data | CRC |
|---|---|---|---|---|
| <STX> | 0x07 | 0x24 | 0xDEADDEAD | 0x4C06 |

The request was successful and the data is transmitted to the host.

## 5.8 Reading Multiple Blocks from a Tag in Selected Mode

In this example several blocks of data are read from a tag that was previously put into selected mode via SELECT_TAG. Because the tag is in selected mode the TID does not have to be sent to the reader. To keep the selected mode activated the RF_F bit has to be set in all requests. The host has to supply a valid Starting Block number and a valid Number of Blocks.

### 5.8.1 ASCII Version

Request:

| | Flags | Request | Tag Type | Starting Block | Number of Blocks | |
|---|---|---|---|---|---|---|
| <CR> | 08 | 24 | 01 | 03 | 03 | <CR> |

In this case three blocks are read from an ISO 15693 tag starting at block 0x03.

Reply:

| | Reply Code | Data | |
|---|---|---|---|
| <LF> | 24 | 99 99 99 99 AA AA AA AA BB BB BB BB | <CR><LF> |

The request was successful and the data is transmitted to the host.

### 5.8.2 Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | Starting Block | Number of Blocks | CRC |
|---|---|---|---|---|---|---|---|
| <STX> | 0x07 | 0x28 | 0x24 | 0x06 | 0x05 | 0x02 | 0x50AC |

In this case two blocks of data are read from a PicoTag starting at block 0x05.

Reply:

| | Reply Length | Reply Code | Data | CRC |
|---|---|---|---|---|
| <STX> | 0x13 | 0x24 | 0x11111111222222223333333344444444 | 0x5362 |

The request was successful and the data is transmitted to the host.

## 5.9 Writing Data onto a Tag Using the TID

In this example a block of data is written onto a tag whose TID was previously determined using e.g. SELECT_TAG. The host has to supply a valid Starting Block and a valid Number of Blocks as well as the appropriate amount of data to write. The Number of Blocks is set to one in this example. Valid values fort he Starting Block and Number of Blocks parameters depend on tag type (see Table 34).

### 5.9.1 ASCII Version

Request:

| | Flags | Request | Tag Type | TID | Starting Block | Number of Blocks | Data | |
|---|---|---|---|---|---|---|---|---|
| <CR> | 40 | 44 | 02 | 01 00 00 00 05 CA 5D E2 | 05 | 01 | 00 11 22 33 | <CR> |

In this example one block of data is written onto an I-Code SL1 tag starting at block 0x05.

Reply:

| | Reply Code | |
|---|---|---|
| <LF> | 44 | <CR><LF> |

The request was successful and the data is written onto the tag.

### 5.9.2 Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | TID | Start ing block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|---|---|
| <STX> | 0x13 | 0x60 | 0x44 | 0x01 | 0xE0070000 06E5D3A7 | 0x00 | 0x01 | 0x12345678 | 0x3538 |

In this example a block of data was written to an ISO 15693 tag starting at block 0x00.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x44 | 0x2E48 |

The request was successful and the data is written onto the tag.

## 5.10 Writing Data onto a Tag in Selected Mode

In this example a block of data is written onto a tag that was put into selected mode using a SELECT_TAG request. Because the tag is in selected mode the TID does not have to be sent to the reader. To keep the selected mode activated the RF_F bit has to be set in all requests. The host has to supply a valid Starting Block and a valid Number of Blocks as well as the appropriate amount of data to write. The Number of Blocks is set to one in this example. Valid values fort he Starting Block and Number of Blocks parameters depend on tag type (see Table 34).

### 5.10.1 ASCII Version

Request:

| | Flags | Request | Tag Type | Starting Block | Number of Blocks | Data | |
|---|---|---|---|---|---|---|---|
| <CR> | 08 | 44 | 0A | 07 | 01 | 51 52 53 54 | <CR> |

Reply:

| | Reply Code | |
|---|---|---|
| <LF> | 44 | <CR><LF> |

The request was successful and the data is written onto the tag.

### 5.10.2 Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | Starting Block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|---|
| <STX> | 0x0F | 0x28 | 0x44 | 0x06 | 0x06 | 0x01 | 0x1234567890ABCDEF | 0x5694 |

In this example a block of data is written onto a PicoTag starting at block 0x06.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x44 | 0x2E48 |

The request was successful and the data was written onto the tag.

## 5.11 Writing of Several Blocks onto a Tag in Selected Mode

In this example a block of data is written onto a tag that was put into selected mode using a SELECT_TAG request. Because the tag is in selected mode the TID does not have to be sent to the reader. To keep the selected mode activated the RF_F bit has to be set in all requests. The host has to supply a valid Starting Block and a valid Number of Blocks as well as the appropriate amount of data to write. Valid values fort he Starting Block and Number of Blocks parameters depend on tag type (see Table 34).

### 5.11.1 ASCII Version

Request:

| | Flags | Request | Tag Type | Starting Block | Number of Blocks | Data | |
|---|---|---|---|---|---|---|---|
| <CR> | 08 | 44 | 01 | 08 | 02 | 01 02 03 04 05 06 07 08 | <CR> |

In this example two blocks of data are written onto an ISO 15693 tag starting at block 0x03.

Reply:

| | Reply Code | |
|---|---|---|
| <LF> | 44 | <CR><LF> |

The request was successful and the data was written onto the tag.

### 5.11.2 Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | Starting Block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|---|
| <STX> | 0x0F | 0x28 | 0x44 | 0x03 | 0x06 | 0x02 | 0xBADFACE0DEADDEAD | 0xB2B8 |

In this example two blocks of data are written onto a Tag-It HF tag starting at block 0x06.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x44 | 0x2E48 |

The request was successful and the data was written onto the tag.

## 5.12 Locking a Data Block on a Tag Using its TID

In this example a block of data is locked on a tag whose TID was previously determined using SELECT_TAG. To do this the WRITE_TAG request is used with the LOCK_F bit set. When used this way the request has to be sent without Data field. The host has to supply a valid Starting Block and a valid Number of Blocks. The Number of Blocks is set to one in this example. Valid values for the Starting Block and Number of Blocks depend on tag type and can be found in chapter 6. Certain tag types do not allow locking particular memory blocks at all.

### 5.12.1 ASCII Version

Request:

| | Flags | Request | Tag Type | TID | Starting Block | Number of Blocks | |
|---|---|---|---|---|---|---|---|
| <CR> | 44 | 44 | 0A | 04 A9 1D 11 12 7A 10 | 04 | 01 | <CR> |

In this case block 0x04 of a tag using MIFARE Ultralight protocol is locked.

Reply:

| | Reply Code | |
|---|---|---|
| <LF> | 44 | <CR><LF> |

The request was successful.

### 5.12.2 Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | TID | Start-block | Number of Blocks | CRC |
|---|---|---|---|---|---|---|---|---|
| <STX> | 0x0F | 0x64 | 0x44 | 0x01 | 0xE007000006E5D3A7 | 0x00 | 0x01 | 0xB45A |

In this example block 0x00 of an ISO 15693 tag is locked.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x44 | 0x2E48 |

The request was successful.

## 5.13 Locking Several Blocks on a Tag Using its TID

In this example several blocks of data are locked on a tag whose TID was previously determined using SELECT_TAG. To do this the WRITE_TAG request is used with the LOCK_F bit set. When used this way the request has to be sent without Data field. The host has to supply a valid Starting Block and a valid Number of Blocks. Valid values for the Starting Block and Number of Blocks depend on tag type and can be found in chapter 6. Certain tag types do not allow locking particular memory blocks at all.

### 5.13.1 ASCII Version

Request:

| | Flags | Request | Tag Type | TID | Starting Block | Number of Blocks | |
|---|---|---|---|---|---|---|---|
| <CR> | 44 | 44 | 03 | 01 32 1F A7 | 04 | 03 | <CR> |

In this example blocks 4, 5 and 6 of a Tag-It HF tag are locked.

Reply:

| | Reply Code | |
|---|---|---|
| <LF> | 44 | <CR><LF> |

The request was successful.

## 5.14 Locking a Block of Data on a Tag in Selected Mode

In this example a block of data is locked that was previously put into selected mode using SELECT_TAG. Because the tag is in selected mode the TID does not have to be included in the request. To keep the selected mode activated the RF_F bit has to be set in all requests. To lock the data block the WRITE_TAG request is used with the LOCK_F bit set. When used this way the request has to be sent without Data field. The host has to supply a valid Starting Block and a valid Number of Blocks. Valid values for the Starting Block and Number of Blocks depend on tag type and can be found in chapter 6. Certain tag types do not allow locking particular memory blocks at all.

### 5.14.1 Binary Version

Request:

| | Request Length | Flags | Request | Tag Type | Starting block | Number of Blocks | CRC |
|---|---|---|---|---|---|---|---|
| <STX> | 0x07 | 0x2C | 0x44 | 0x02 | 0x05 | 0x01 | 0xB5A2 |

In this example block 0x05 on an I-Code SL1 tag is locked.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x44 | 0x2E48 |

The request was successful.

## 5.15 Read the Reader’s Firmware Version

The firmware version is a two byte long number that can usually only be read and is written only during a firmware update. Reading the firmware version uses the READ_SYS request.

### 5.15.1 ASCII Version

Request:

| | Flags | Request | Starting Block | Number of Blocks | |
|---|---|---|---|---|---|
| <CR> | 00 | 22 | 01 | 01 | <CR> |

In this case block 0x01 of the reader’s memory is read where the firmware version is stored according to table 11.

Reply:

| | Reply Code | Data | |
|---|---|---|---|
| <LF> | 22 | 1002 | <CR><LF> |

The request was successful. As can be seen in the reply the data block returned is two bytes long and contains the current firmware version which is 0x1002 in this example.

### 5.15.2 Binary Version

Request:

| | Request Length | Flags | Request | Start-block | Number of Blocks | CRC |
|---|---|---|---|---|---|---|
| <STX> | 0x06 | 0x20 | 0x22 | 0x01 | 0x01 | 0x0A19 |

In this case block 0x01 of the reader's memory is read where the firmware version is stored according to table 11. Please note the additional CRC and Request Length fields which are required. The different Flags value is due to the presence of the CRC field.

Reply:

| | Reply Length | Reply Code | Data | CRC |
|---|---|---|---|---|
| <STX> | 0x05 | 0x22 | 0xF002 | 0x87CE |

The request was successful. As can be seen in the reply the data block returned is two bytes long and contains the current firmware version which is 0xF002 in this example. Here as well there are additional fields when compared to the ASCII version.

## 5.16 Reading the Reader ID (RID)

The Reader ID (RID) is stored both in volatile runtime memory and in permanent system parameter memory. This allows reading it from both locations. Since it is possible to change the RID in both locations independently, the results of reading it from both locations do not necessarily have to match. When switching on the device the RID stored in permanent system parameter memory is copied into runtime volatile memory so that right after startup the two will agree.

### 5.16.1 ASCII Version

Request:

| | Flags | Request | Starting Block | Number of Blocks | |
|---|---|---|---|---|---|
| <CR> | 00 | 21 | 02 | 01 | <CR> |

In this case block 0x02 of the permanent system parameter memory is read where the RID is stored according to Table 11.

Reply:

| | Reply Code | Data | |
|---|---|---|---|
| <LF> | 21 | FF | <CR><LF> |

The request was successful. As can be seen from the reply the data returned is one byte in size and contains the current RID which is 0xFF in this example. This is the RID which enables the reader to answer requests that are not directed at a defined RID and this is also the factory setting.

### 5.16.2 Binary Version

Request:

| | Request Length | Flags | Request | Starting block | Number of Blocks | CRC |
|---|---|---|---|---|---|---|
| <STX> | 0x06 | 0x20 | 0x22 | 0x02 | 0x01 | 0x2071 |

In this case block 0x02 of the permanent system parameter memory is read where the RID is stored according to Table 11. Please note the additional CRC and Request Length fields which are required. The different Flags value is due to the presence of the CRC field.

Reply:

| | Reply Length | Reply Code | Data | CRC |
|---|---|---|---|---|
| <STX> | 0x04 | 0x22 | 0xFF | 0x7C9A |

The request was successful. As can be seen from the reply the data returned is one byte in size and contains the current RID which is 0xFF in this example. This is the RID which enables the reader to answer requests that are not directed at a defined RID and this is also the factory setting. Here as well there are additional fields when compared to the ASCII version.

## 5.17 Setting the Reader ID (RID)

The reader ID (RID) is stored both in volatile runtime memory and in permanent system parameter memory of the reader. This allows setting it in these two locations. If the RID is changed in permanent memory the changes only take effect after rebooting the system. If it is changed in volatile runtime memory the changes take effect immediately but are lost when rebooting the system.

### 5.17.1 ASCII Version

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 41 | 02 | 01 | FF | C985 | <CR> |

In this example the value 0xFF is written into block 0x02 of permanent system parameter memory which contains the RID according to Table 11. Please note that the CRC Field is necessary.

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 41 | 538D | <CR><LF> |

The request was successful.

### 5.17.2 Binary Version

Request:

| | Request Length | Flags | Request | Starting block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|
| <STX> | 0x07 | 0x20 | 0x42 | 0x02 | 0x01 | 0xFF | 0xF099 |

In this example the value 0xFF is written into block 0x02 of permanent system parameter memory which contains the RID according to Table 11. Please note the Request Length field which is required.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x42 | 0x4B7E |

The request was successful.

## 5.18 Setting Baud Rate

The baud rate parameter defines the data rate used when communicating over the serial interface via RS232, USB232 or Ethernet232. To allow for communication between host and reader the baud rate has to be set to the same value on both systems. Possible settings fort he baud rate can be found in Table 11.

### 5.18.1 ASCII Version

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 41 | 03 | 01 | 00 | 9C21 | <CR> |

In this example the value 0x00 is written into block 0x03 of permanent system parameter memory where according to Table 11 the baud rate is stored. This sets the baud rate to 9600.

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 41 | 538D | <CR><LF> |

The request was successful.

### 5.18.2 Binary Version

Request:

| | Request Length | Flags | Request | Starting block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|
| <STX> | 0x07 | 0x20 | 0x42 | 0x03 | 0x01 | 0x00 | 0xA53D |

In this example the value 0x00 is written into block 0x03 of volatile runtime memory where according to Table 11 the baud rate is stored. This sets the baud rate to 9600.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x42 | 0x4B7E |

The request was successful.

Both examples set the baud rate to 9600. The reply is always sent with the original baud rate before the new setting was made. In case the new setting was written into volatile runtime memory any further communication will use the new setting. Writing the new setting into permanent system parameter memory, however, will only have an effect after rebooting the reader.

Baud rate can not be read as the reading process requires data communication which in turn requires the correct baud rate setting to work. Thus the correct baud rate is already known and needn’t be read.

## 5.19 Setting Operating Mode

The RFID reader has three operating modes: loop mode, activated mode and sleep mode. To save energy – especially if operated with a battery – the reader should be put into sleep mode when not needed. If there is a focus on saving energy it is even more important to set the RF_F flag bit as rarely as possible as energy consumption is highest when the reader is keeping the RF field powered. This is especially the case for high power readers where the operation of all other components will only have a relatively small effect when compared to the amplifier. As long as the RF_F bit is set the RF field is powered by the reader.

If the host writes a value of 0x00 into volatile runtime memory at block 0x04 (using WRITE_SYS) the reader enters sleep mode after replying to the request with a reply indicating success. Sending an arbitrary byte of data to the reader in sleep mode will wake it and lead to the same reply as before. The byte sent for waking up the reader is ignored otherwise. It is not possible to change directly from sleep to loop mode.

If, however, the host writes into block 0x04 of the permanent system parameter memory of the reader (using WRITE_MEM) this controls start up behaviour of the reader. The settings made here are executed as long as the Start up Request field is not activated (see Chapter 5.22). The reader can be put into any of the three modes upon startup. What the host has to send as a request can be seen in the following table:

Table 33 – Coding of Operating Mode on Start up

| | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| **Sleep Mode** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Loop Mode** | 1 | 0 | MIFARE Ultralight | PicoTag | MIFARE Classic | Tag-It HF | ICode1 | ISO 15693 |
| **Activated Mode** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

Examples fort his type of coding can be 0x00 for sleep mode, 0x40 for activated mode and 0x81 for loop mode for ISO 15693 tags.

## 5.19.1 ASCII Version: Sleep Mode at Runtime

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 42 | 04 | 01 | 00 | 35E9 | <CR> |

This request will put the reader into sleep mode if it was in activated mode before and vice versa.

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 42 | 6116 | <CR><LF> |

The reply indicates success for both cases.

### 5.19.2 Binary Version: Sleep Mode at Runtime

Request:

| | Request Length | Flags | Request | Starting block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|
| <STX> | 0x07 | 0x20 | 0x42 | 0x04 | 0x01 | 0x00 | 0x35E9 |

This request will put the reader into sleep mode if it was in activated mode before and vice versa.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x42 | 0x4B7E |

The reply indicates success for both cases.

### 5.19.3 ASCII Version: Sleep Mode after Booting

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 41 | 04 | 01 | 00 | 1024 | <CR> |

This request will put the reader into sleep mode right after booting if the Startup Request field is not set.

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 41 | 538D | <CR><LF> |

The reply indicates that the request was successful.

### 5.19.4 Binary Version: Loop Mode after Booting

Request:

| | Request Length | Flags | Request | Starting block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|
| <STX> | 0x07 | 0x20 | 0x41 | 0x04 | 0x01 | 0x81 | 0x9974 |

In this case the reader will automatically go into loop mode after booting and will be searching for ISO 15693 tags if the Start up Request field is not set.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x41 | 0x79E5 |

The reply indicates a successful request.

## 5.20 GPIO-Pin Control

The reader has eight pins that can be used for general purpose inputs or outputs. The host can configure each of these pins to be either an input or an output. Depending on whether the according pins are configured as inputs or outputs they can be read from or written to respectively.

System parameters I/O Pin Direction (block 0x07 in reader memory) and I/O Values (block 0x08 in reader memory) define whether an individual pin is an input or output and which values are present there.

The GPIO pins can source and sink up to 25 mA, each individual pin can sink or source up to 15 mA.

The bits set in system parameter I/O Pin Direction (0x07) define for each pin individually whether it is an input or output. Using WRITE_SYS and WRITE_MEM requests these values can be changed.

The bits set in system parameter I/O Values can either be read or written depending on whether the respective pins are input or output. To read the bit values of pins that are inputs the READ_SYS request is used. The output values can be set using either WRITE_SYS or WRITE_MEM. Only output values are affected by these two requests.

### 5.20.1 ASCII Version: Set I/O PIN Direction at Runtime

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 42 | 07 | 01 | 0F | 227A | <CR> |

This request sets the four least significant bits as inputs and the four most significant bits as outputs. The request writes 0x0F into block 0x07 of volatile runtime memory which is 0000 1111 in binary notation. A bit that is set to 1 defines an input. This setting will be lost when the reader is rebooted as it was only written to runtime memory.

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 42 | 6116 | <CR><LF> |

The reply indicates a successful request.

### 5.20.2 ASCII Version: Writing Values into Pins Defined as Outputs

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 42 | 08 | 01 | F0 | 67C5 | <CR> |

With this request the host wrote ones to the pins which were set as output pins in the previous example. The data byte is 1111 0000 when written in binary format – the four upper pins are set to 1 by writing it to block 0x08. As the lower four pins are set to being inputs writing zeros into them as shown here has no effect. If the same pins are read in a subsequent request they can contain different values than zero. Again the values written to the pins are lost when rebooting the reader as they were written to volatile runtime memory.

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 42 | 6116 | <CR><LF> |

The reply indicates a successful request.

### 5.20.3 ASCII Version: Reading Input Pin Values

Request:

| | Flags | Request | Starting Block | Number of Blocks | |
|---|---|---|---|---|---|
| <CR> | 00 | 22 | 08 | 01 | <CR> |

This request reads the values currently found on the input pins. Again, it is assumed that the same pin direction is valid as was set in the example of Chapter 5.20.1. Reading is always done from volatile runtime memory (request 22).

Reply:

| | Reply Code | Data | |
|---|---|---|---|
| <LF> | 22 | F0 | <CR><LF> |

The reply indicates that the request was successful. The data returned by the reply is what was found at the input pins. All pins in this example have a value of one set (0000 1111 is the binary coding of the data byte). The four upper pins were configured as outputs and do not supply sensible reply data. They could have been set to one and still report zero here. Please make sure the pin direction is set correctly for input and output pins as otherwise the values read or written will not make sense.

## 5.21 Control of metraTec Multiplexers

The programming instructions for the input and output pins described in the previous example of Chapter 5.20 can be used to write code that will give you control over the many metraTec multiplexers available. The example used in this chapter assumes that a maximum number of antenna ports are to be controlled. To achieve this setup a primary 16 port multiplexer is connected to the four upper pins (pins 5-8) of the reader (control lead 1 to pin 5, 2 to 6, etc.). The 16 secondary multiplexers are connected with their reader ports to the 16 antenna ports of the primary multiplexer. Their control leads are bundled according to color and then connected to pins 1-4 of the reader in the same order as those of the primary multiplexer. After switching on power for the multiplexers the reader is started. As currently no settings have been made both primary and secondary multiplexers switch to antenna port 1 which results in the reader being connected to the first antenna.

The application that is planned in this example calls for switching between all antennas sequentially and finding all ISO 15693 tags within reach of the according antenna.

To do this the first step is to set up multiplexer control. All eight GPIO pins are configured to be output pins (see Chapter 5.20.1):

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 41 | 07 | 01 | 00 | FF40 | <CR> |

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 41 | 538D | <CR><LF> |

The reply indicates a successful request. All eight GPIO pins are now configured to be outputs. Please not that this new configuration was written into permanent system parameter memory so that they will not be lost with a reboot. For this change to take effect, the reader has to be rebooted.

2. The next step is to write values to the output pins. As at the beginning antenna one is to be read the following sequence results (see also Chapter 5.20.2):

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 42 | 08 | 01 | 00 | 904A | <CR> |

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 42 | 6116 | <CR><LF> |

All output pins were set to zero which results in the first antenna to be connected to the reader as described before. This time the values were written to runtime volatile memory as these values will frequently change and do not have to be stored if the reader is powered down.

3. Now, all ISO 15693 tags within range of antenna 1 have to be found (see Chapter 5.2.2):

Request:

| | Flags | Request | Tag Type | |
|---|---|---|---|---|
| <CR> | 02 | 14 | 01 | <CR> |

Reply:

| | Reply Code | Data (TID) | |
|---|---|---|---|
| <LF> | 14 | E0 07 00 00 01 64 5E 37 | <CR><LF> |
| <LF> | 14 | E0 07 00 00 01 54 65 31 | <CR><LF> |
| <LF> | 14 | E0 07 00 00 01 54 41 32 | <CR><LF> |

| | | |
|---|---|---|
| <LF> | 94 | <CR><LF> |

4. After reply 0x94 was received (no further tags found) the reader can switch to antenna 2. This is done by changing the request of step 2 by setting the data in the Data field to 0x01. This loop comprising Steps 2 to 4 is repeated and the data value incremented until 0xFF is reached. This is when antenna 256 is being read from. If desired, the overall lop can then be restarted by setting the data value to 0x00.

## 5.22 Setting the Startup Request

The reader offers the user to store a request that is automatically executed when it is turned on. This request has to be written to volatile runtime memory using the WRITE_SYS request addressing block 0x12. The reader will execute whichever request it finds there at start up and replies to it in ASCII or binary notation depending on the way the request was stored.

The request written to that block has to be complete and correct to work. Depending on the flags set, the TID, RID and CRC fields have to be present in it. For binary coding the request length field should not be forgotten. The control characters (e.g. <CR>) are not part of such a request.

In case this feature is not to be used, an arbitrary byte should be written to the according block address. If this feature is deactivated this way, the operating mode settings described in Chapter 5.19 can be used.

### 5.22.1 ASCII Version: SELECT_TAG on Start up

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 42 | 12 | 01 | 00 14 01 | 340F | <CR> |

In this example the SELECT_TAG request is automatically executed when switching on the reader. It is set to search for ISO 15693 tags (0x01). Since the request was sent to the reader in ASCII notation the reply of the reader on start up will also be in this notation. As with all WRITE_SYSTEM commands, the CRC field is necessary.

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 42 | 6116 | <CR><LF> |

The reply indicates a successful request.

## 5.22.2 Binary Version: SELECT_TAG on Start up

Request:

| | Request Length | Flags | Request | Starting block | Number of Blocks | Data | CRC |
|---|---|---|---|---|---|---|---|
| <STX> | 0x0C | 0x20 | 0x42 | 0x12 | 0x01 | 0x05211400C541 | 0xD591 |

This example is a bit more complex than the previous one. On startup the reader goes into loop mode (LOOP_F flag bit is set) and transmits the data including CRC field (CRC_F flag bit is set). It executed the SELECT_TAG request (0x14) with Auto Detect (0x00) as the tag type.

Reply:

| | Reply Length | Reply Code | CRC |
|---|---|---|---|
| <STX> | 0x03 | 0x42 | 0x4B7E |

The reply indicates a successful request.

## 5.22.3 ASCII Version: Deactivate Startup Request

Request:

| | Flags | Request | Starting Block | Number of Blocks | Data | CRC | |
|---|---|---|---|---|---|---|---|
| <CR> | 20 | 42 | 12 | 01 | 00 | 66A5 | <CR> |

This request deactivates the start up request feature.

Reply:

| | Reply Code | CRC | |
|---|---|---|---|
| <LF> | 42 | 6116 | <CR><LF> |

The reply indicates a successful request.

# 6. Tag Type Descriptions

## 6.1 ISO 15693

The ISO/IEC 15693 Standard was developed for "Contactless Vicinity Cards" and published in 1998. It had significant influence on the global acceptance of RFID solutions at 13.56 MHz. The technical foundations of the standard are tag developments by Texas Instruments and Philips. The standard basically comprises the features of both the Tag-It HF and I-Code1 transponders.

ISO 15693-1 defines the physical characteristics of credit card transponders.

ISO 15693-2 defines the RF interface at 13.56 MHz and its modulation methods in agreement with international standards.

ISO 15693-3 specifies the request protocol and the anti collision method for data exchange between tags and readers.

The ISO 15693 standard allows tag producers to make tags that support further optional custom requests and that have different memory architectures, sizes and structures. The protocol described in this programming guide supports all transponders of all IC manufacturers compatible with this norm.

### 6.1.1 Tag-It HF-I ISO 15693 (Texas Instruments)

The complete technical specification of the Tag-It HF-I tag transponders can be found in the according data sheets by Texas Instruments called "Tag-It HF-I Transponder Inlays Reference Guide".

### 6.1.2 I-Code SLI ISO 15693 (Philips)

The complete technical specification of the I-Code SLI tag transponders can be found in the according data sheets by Philips called "I-Code SLI Smart Label IC SL2 ICS20 Functional Specification".

### 6.1.3 my-d SRF55VxxP ISO 15693 (Infineon)

The complete documentation of the my-d SRF55VxxP can be obtained from Infineon.

### 6.1.4 LRI512 ISO 15693 (ST Microelectronics)

The complete transponder specification of the LRI512 can be found in the data sheets by ST

Microelectronics called "LRI512 Memory TAG IC 512 bit High Endurance EEPROM 13.56 MHz, ISO 15693 Standard Compliant with E.A.S..".

## 6.2 Tag-It HF

The first 13.56 MHz transponder developed by Texas Instruments was the Tag-It HF. This transponder was part of the technical development that went into the ISO 15693 standard. However, the original Tag-It HF transponders are not ISO compatible (whereas their successors, the Tag-It HF-I are ISO compatible – please note the small but significant difference!). They are still being produced in great numbers and have an installed base of many millions worldwide. The transponders use a proprietary communication interface of Texas Instruments. The complete documentation to the Tag-It HF transponders can be obtained from Texas Instruments.

## 6.3 I-Code1

The I-Code1 (SL1) was the first 13.56 MHz transponder chip by Philips designed for longer range applications. It is the second IC whose technical capabilities were considered when designing the ISO15693 standard. It is not standard conform however (whereas its successor the I-Code SLI (SL2), introduced in 2002, is ISO compliant – please note the small but significant difference!) and uses a proprietary communication interface by Philips. The I-Code1 (SL1) is widely used worldwide and is still being produced in large numbers. The complete technical specification can be obtained from Philips.

## 6.4 PicoTag

Inside Contactless (previously known as Inside Technologies) produces a series of RFID transponders called PicoTag. There are different types which have different memory sizes (2K and 16 K) and different operating modes (normal and secure).

## 6.5 ISO 14443

ISO/ IEC 14443 is a standard composed of four parts which define "Contactless Proximity Cards" with a short reading / writing range. Introduced in 1999 and 2000 the standard has become the worldwide standard for cashless payment.

ISO 14443-1 defines the physical characteristics of the RFID cards.

ISO 14443-2 defines two types (A and B) of interface and modulation methods at 13.56 MHz for the communication between tag and reader.

ISO 14443-3 defines the anti collision method to be able to select one tag among many in the RF field.

ISO 14443-4 specifies the upper protocol layer and the method for data exchange between tag and reader.

### 6.5.1 14443-A MIFARE Classic 4K Technology (NXP)

The IC using MIFARE® Classic 4K technology by NXP is currently in use in many millions of secure contactless applications. It was introduced in 1995. The complete specification can be obtained from NXP and is documented in the data sheet "MF1S70yyX - MIFARE Classic 4K - Mainstream contactless smart card IC for fast and easy solution development" (current status May 2011).

### 6.5.2 MIFARE Ultralight Technology (NXP)

The complete specification for the MIFARE Ultralight® protocol can be found in a data sheet called "MF0ICU1 - MIFARE Ultralight contactless single-ticket IC", current status December 2010, published by NXP.

## 6.6 Technical Data of the Transponders

The following table is meant to give an overview over the technical specifications of the different transponder types. It contains an overview over the memory architecture of the different transponders, shows which memory areas can be written, read and locked and which tag types have which numbers in the Tag Type field of many requests described in previous chapters.

Table 34: Tag types, properties and addressing

| Tag Name | Tag Type | User Data Memory Size | Writable Address Space | Lockable |
|---|---|---|---|---|
| Tag-It HF-I | 01 | 256 Bytes | 0x00-0x3F<br>4 Bytes each | Yes, blockwise<br>Not unlockable |
| I-Code SLI | 01 | 112 Bytes | 0x00-0x1B<br>4 Bytes each | Yes, blockwise<br>Not unlockable |
| my-d SRF55V02P | 01 | 232 Bytes | 0x03-0x1F<br>8 Bytes each | Yes, blockwise<br>Not unlockable |
| my-d SRF55V10P | 01 | 1000 Bytes | 0x03-0x7F<br>8 Bytes each | Yes, blockwise<br>Not unlockable |
| LRI512 | 01 | 64 Bytes | 0x03-0x0F<br>4 Bytes each | Yes, blockwise<br>Not unlockable |
| Tag-It HF | 03 | 32 Bytes | 0x00-0x07<br>4 Bytes each | Yes, blockwise<br>Not unlockable |
| I-Code1 | 02 | 64 Bytes | 0x03-0x0F<br>4 Bytes each | Yes, blockwise<br>Not unlockable |
| PicoTag 2K | 06 | 232 Bytes | 0x03-0x1F<br>8 Bytes each | Yes, blockwise<br>Not unlockable |
| MIFARE Classic 4K | 04 | approx. 1000 Bytes | 16 Sectors of 4 Blocks with 16 Bytes each, writable are Blocks 0x01, 0x02 in the first Sector, in all other Sectors the first 3 out of 4 Blocks | Yes, sectorwise<br>Last block in each sector contains access control data |
| MIFARE Ultralight | 0A | 48 Bytes | 0x04-0x0F<br>4 Bytes each | Yes, blockwise, Not unlockable; 32 One-Time-Programmable Bits (Block 0x03) |
| Gemwave C210 | 08 | Read Only | Read Only | Everything is locked permanently |

This table shows only part of the currently available transponder types. For further transponder types (and especially their memory configurations) please refer to the technical data sheets supplied by the respective transponder manufacturers.

# 7. Exception Handling

When programming applications, exception handling plays an important role. Good programs should take exceptions into account and treat them correctly to be robust. Especially errors in transmission of single bits in the Flags field can lead to strange behaviour and should be caught and treated correctly. The following is a list of strange reader behaviour that results due to a one bit transmission error (by far the most common transmission error type):

A transmission error for the LOOP_F bit can lead to the loop mode being activated, resulting in a Reply Code of 1C (or 1C1189 if CRC is activated). Further replies will then send TIDs of tags found in the RF field of the reader.

In case the loop mode is activated (and is meant to be) any byte erroneously sent by the host will lead to a termination of the loop mode and a Reply Code of 9C (or 9C9DC1 if CRC is active).

A transmission error fort he CRC_F bit can lead to the reply containing the CRC even though this is not expected or not containing it when it is expected. It is recommended to have CRC activated as this allows catching any transmission errors.

A transmission error for the RID_F bit will lead to no reply or to a reply by the wrong reader.

A transmission error for the TID_F bit or in the TID field will lead to an error code as a reply or to a reply by the wrong tag.

# 8. Version Control

| *Version* | *Change* | *by* | *Date* |
|---|---|---|---|
| 1.0 | created | KD | 01.09.2007 |
| 1.1 | small corrections | CS | 24.02.2015 |

Contact

metraTec GmbH
Werner-Heisenberg-Str. 1
39106 Magdeburg, Germany

Tel.: +49 (0)391 251906-00
Fax: +49 (0)391 251906-01

Email: support@metratec.com
Web: www.metratec.com

We are constantly improving our products. Changes in function, form, features can happen without prior notice.