



## **Long Range Reader Modul UHF – M700**

### **Module API for C**

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

Issue 1.4  
– 19. May 2025 –

Phone: +49 621 6690094-0  
Fax: +49 621 6690094-9  
E-Mail: [info@idtronic.de](mailto:info@idtronic.de)  
Web: [idtronic.de](http://idtronic.de)

Subject to alteration without prior notice.  
© Copyright iDTRONIC GmbH 2025  
Printed in Germany

## Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction .....</b>                   | <b>5</b>  |
| 1.1      | <b>Functions.....</b>                       | <b>5</b>  |
| 1.2      | <b>Enumerations .....</b>                   | <b>5</b>  |
| 1.2.1    | READER_ERR Enumeration .....                | 5         |
| 1.2.2    | Mtr_Param Enumeration .....                 | 6         |
| 1.2.3    | SL_TagProtocol Enumeration .....            | 7         |
| 1.2.4    | Region_Conf Enumeration .....               | 7         |
| 1.2.5    | Lock_obj Enumeration .....                  | 8         |
| 1.2.6    | Lock_Type Enumeration .....                 | 8         |
| 1.2.7    | GpiTrigger_Type Enumeration .....           | 9         |
| 1.3      | <b>Structures.....</b>                      | <b>9</b>  |
| 1.3.1    | AntPortsVSWR Structure .....                | 9         |
| 1.3.2    | TAGINFO Structure .....                     | 9         |
| 1.3.3    | TagSelector_ST Structure.....               | 9         |
| 1.3.4    | GpiState_ST Structure.....                  | 10        |
| 1.3.5    | GpiTrigger Structure .....                  | 10        |
| 1.3.6    | Inv_Potl Structure - Deprecated .....       | 10        |
| 1.3.7    | Inv_Potls_ST Structure - Deprecated .....   | 10        |
| 1.3.8    | GpiInfo_ST Structure.....                   | 10        |
| 1.3.9    | TagFilter_ST Structure .....                | 10        |
| 1.3.10   | EmbededData_ST Structure.....               | 11        |
| 1.3.11   | AntPower Structure .....                    | 11        |
| 1.3.12   | AntPowerConf Structure.....                 | 11        |
| 1.3.13   | ConnAnts_ST Structure.....                  | 11        |
| 1.3.14   | HoptableData_ST Structure .....             | 12        |
| 1.3.15   | MultiTagSelectors_ST Structure .....        | 12        |
| 1.3.16   | OnTagReadBlock Structure .....              | 12        |
| 1.3.17   | OnGpiTriggerBlock Structure .....           | 12        |
| 1.3.18   | OnReadingErrorBlock Structure.....          | 12        |
| 1.3.19   | CustomParam_ST Structure.....               | 12        |
| 1.3.20   | Default_Param Structure .....               | 13        |
| 1.3.21   | Reader_Ip Structure.....                    | 14        |
| 1.3.22   | BackReadOption Structure .....              | 14        |
| 1.3.23   | TagMetaFlags Structure .....                | 14        |
| 1.3.24   | HardwareDetails Structure .....             | 14        |
| <b>2</b> | <b>Reader Life Cycle .....</b>              | <b>15</b> |
| 2.1      | <b>InitReader_Notype Function .....</b>     | <b>15</b> |
| 2.2      | <b>CloseReader Function .....</b>           | <b>15</b> |
| <b>3</b> | <b>Reader Parameters.....</b>               | <b>15</b> |
| 3.1      | <b>Get Parameter .....</b>                  | <b>15</b> |
| 3.2      | <b>Set Parameter .....</b>                  | <b>16</b> |
| 3.3      | <b>Default Configuration .....</b>          | <b>16</b> |
| 3.4      | <b>Custom Parameter Configuration .....</b> | <b>16</b> |
| 3.5      | <b>Parameters.....</b>                      | <b>16</b> |
| 3.5.1    | Read/write parameters.....                  | 16        |
| 3.5.2    | Read-only Parameters .....                  | 18        |

|           |  |           |
|-----------|--|-----------|
| <b>4</b>  | <b>Synchronous Inventory .....</b>       | <b>18</b> |
| 4.1       | TagInventory_Raw Function.....           | 19        |
| 4.2       | GetNextTag Function .....                | 19        |
| <b>5</b>  | <b>Fast Mode Inventory .....</b>         | <b>19</b> |
| 5.1       | AsyncStartReading Method.....            | 20        |
| 5.2       | AsyncGetTagCount Method .....            | 20        |
| 5.3       | AsyncGetNextTag Method .....             | 20        |
| 5.4       | AsyncStopReading Method .....            | 21        |
| <b>6</b>  | <b>Asynchronous Inventory .....</b>      | <b>22</b> |
| 6.1       | Start Inventory .....                    | 22        |
| 6.1.1     | TagReadHandler Callback.....             | 23        |
| 6.1.2     | SetTagReadHandler Function .....         | 23        |
| 6.1.3     | ReadingErrorHandler Callback .....       | 23        |
| 6.1.4     | SetReadingErrorHandler Function .....    | 23        |
| 6.1.5     | GpiStateHandler Callback.....            | 23        |
| 6.1.6     | GpiTriggerBoundaryHandler Callback ..... | 24        |
| 6.1.7     | SetGpiTriggerHandler Function .....      | 24        |
| 6.1.8     | StartReading Function .....              | 24        |
| 6.2       | Stop Inventory .....                     | 25        |
| 6.2.1     | StopReading Function .....               | 25        |
| 6.3       | Additional Data Settings .....           | 25        |
| <b>7</b>  | <b>Tag Access.....</b>                   | <b>25</b> |
| 7.1       | GetTagData Function .....                | 26        |
| 7.2       | GetTagDataEx Function .....              | 26        |
| 7.3       | WriteTagData Function .....              | 27        |
| 7.4       | WriteTagEpcEx Function.....              | 27        |
| 7.5       | LockTag Function .....                   | 28        |
| 7.6       | Lock180006BTag Function .....            | 29        |
| 7.7       | KillTag Function.....                    | 29        |
| <b>8</b>  | <b>Peripheral Functions .....</b>        | <b>29</b> |
| 8.1       | GetGPI Function .....                    | 30        |
| 8.2       | GetGPIEx Function .....                  | 30        |
| 8.3       | SetGPO Function .....                    | 30        |
| <b>9</b>  | <b>String Functions .....</b>            | <b>30</b> |
| 9.1       | Hex2Str Function.....                    | 31        |
| 9.2       | Str2Hex Function.....                    | 31        |
| <b>10</b> | <b>Error Handling.....</b>               | <b>31</b> |
| 10.1      | GetLastDetailError Function.....         | 32        |
| 10.2      | Error Codes .....                        | 32        |

# 1 Introduction

Include the header file `ModuleReader.h` for access to the library.

`ModuleAPI_C.dll` internally calls `ACE.dll` (only for 32-bit version) and `PCOMM.DLL`, therefore put them in the same catalogue.

All sample code snippets assume `hreader` to be the reader instance handle created by the `InitReader_Notype` function and `err` to be of the enumeration type `READER_ERR`.

## 1.1 Functions

All functions will return a value of type `READER_ERR`, indicating whether the execution was successful or why it failed. Many functions also have a timeout parameter to specify the maximum time in milliseconds that is spend in the function. Valid values for the timeout are in the range of 50 – 65535.

| Function                               | Description  |
|--|--|
| <code>InitReader_Notype</code>         | Initialize the reader and create a handle  |
| <code>CloseReader</code>               | Free the resource that is occupied by the handle for the reader  |
| <code>ParamGet</code>                  | Get the parameters of the reader   |
| <code>ParamSet</code>                  | Set the parameters of the reader   |
| <code>TagInventory_Raw</code>          | Synchronous inventory operation  |
| <code>GetNextTag</code>                | Get a single tag from the set of tags inventoried by the <code>TagInventory_Raw</code> function                                |
| <code>SetTagReadHandler</code>         | Set the <code>TagReadHandler</code> as callback function   |
| <code>TagReadHandler</code>            | Will be called in the event of tags being inventoried during asynchronous inventory  |
| <code>SetReadingErrorHandler</code>    | Set the <code>ReadingErrorHandler</code> as callback function  |
| <code>ReadingErrorHandle</code>        | Will be called in the event of an error during asynchronous inventory  |
| <code>SetGpiTriggerHandler</code>      | Set the <code>GpiStateHandler</code> and <code>GpiTriggerBoundaryHandler</code> as callback functions                          |
| <code>GpiStateHandler</code>           | Will be called if asynchronous inventory is configured to be triggered by GPI; to retrieve information on GPI states           |
| <code>GpiTriggerBoundaryHandler</code> | Will be called if asynchronous inventory is configured to be triggered by GPI; to get cause for starting or stopping inventory |
| <code>StartReading</code>              | Start asynchronous inventory   |
| <code>StopReading</code>               | Stop asynchronous inventory  |
| <code>GetTagDataEx</code>              | Read a block of data from the tag memory   |
| <code>WriteTagData</code>              | Write a block of data to the tag memory  |
| <code>WriteTagEpcEx</code>             | Write the EPC code of a tag  |
| <code>LockTag</code>                   | Lock the tag memory  |
| <code>KillTag</code>                   | Kill a tag   |
| <code>GetGPI</code>                    | Get the state of a GPI pin   |
| <code>GetGPIEx</code>                  | Get the states of multiple GPI pins  |
| <code>SetGPO</code>                    | Set the state of a GPO pin   |
| <code>GetLastDetailError</code>        | Get a more detailed error information  |

## 1.2 Enumerations

### 1.2.1 READER\_ERR Enumeration

All API functions return a value of this enumeration type. If the return value is not `MT_OK_ERR` and it is not a communication error such as `MT_IO_ERR` or `MT_INVALID_READER_HANDLE`, the interface function `GetLastDetailError` can be called to further understand the error details.

| Member | Description |
|--------|-------------|
|--------|-------------|

|   |  |
|---|--|
| MT_OK_ERR                                 | Success  |
| MT_IO_ERR                                 | IO error; Network or serial port connection, the reader must be closed and reinitialized         |
| MT_INTERNAL_DEV_ERR                       | Deprecated error code  |
| MT_CMD_FAILED_ERR                         | Operation failed; not a fatal error  |
| MT_CMD_NO_TAG_ERR                         | No tag found   |
| MT_M5E_FATAL_ERR                          | Deprecated error code  |
| MT_OP_NOT_SUPPORTED                       | Unsupported operation  |
| MT_INVALID_PARA                           | Invalid parameter  |
| MT_INVALID_READER_HANDLE                  | Invalid reader handle  |
| MT_HARDWARE_ALERT_ERR_BY_HIGN_RETURN_LOSS | High return loss; check for surrounding metal environment, poor antenna connection, poor contact |
| MT_HARDWARE_ALERT_ERR_BY_TOO_MANY_RESET   | RFID engine reset too many times   |
| MT_HARDWARE_ALERT_ERR_BY_NO_ANTENNAS      | The reader did not detect an antenna   |
| MT_HARDWARE_ALERT_ERR_BY_HIGH_TEMPERATURE | Reader temperature is too high   |
| MT_HARDWARE_ALERT_ERR_BY_READER_DOWN      | Reader down, no response from the reader   |
| MT_HARDWARE_ALERT_ERR_BY_UNKNOWN_ERR      | Unknown error  |
| M6E_INIT_FAILED                           | Deprecated error code  |
| MT_OP_EXECING                             | The reader is busy with executing another operation  |
| MT_UNKNOWN_READER_TYPE                    | Unknown reader type  |
| MT_OP_INVALID                             | Invalid operation  |
| MT_MAX_ERR_NUM                            | Enumeration keyword error  |
| MT_HARDWARE_ALERT_BY_FAILED_RESET_MODLUE  | Reset RFID engine failed   |
| MT_FREQUENT_ERR                           | Reader error too often   |
| MT_UPDFWFROMSP_OPENFILE_FAILED            | Failed to open file while upgrading firmware by serial port                                      |
| MT_UPDFWFROMSP_FILE_FORMAT_ERR            | File format error while upgrading firmware by serial port  |
| MT_JNI_INVALID_PARA                       | Invalid JNI call parameter   |

### 1.2.2 Mtr\_Param Enumeration

Used by the functions ParamGet and ParamSet to specify the parameter that will be modified.

| Member                              | Description  |
|-------------------------------------|--|
| MTR_PARAM_POTL_GEN2_SESSION         | Gen2 session   |
| MTR_PARAM_POTL_GEN2_Q               | Gen2 Q value   |
| MTR_PARAM_POTL_GEN2_TAGENCODING     | Gen2 tag encoding                                    |
| MTR_PARAM_RF_ANTPOWER               | Tx power of the reader's antennas in centi-dbm       |
| MTR_PARAM_RF_MAXPOWER               | Maximum tx power in centi-dbm                        |
| MTR_PARAM_RF_MINPOWER               | Minimum tx power in centi-dbm                        |
| MTR_PARAM_TAG_FILTER                | Tag filter   |
| MTR_PARAM_TAG_EMBEDDEDATA           | Read data of another bank during inventory operation |
| MTR_PARAM_TAG_INVPOTL               | Deprecated - Air protocol of inventory               |
| MTR_PARAM_READER_CONN_ANTs          | The connected antennas                               |
| MTR_PARAM_READER_AVAILABLE_ANTPORTS | Number of antenna ports on the reader                |
| MTR_PARAM_READER_IS_CHK_ANT         | Antenna detection configuration                      |
| MTR_PARAM_READER_VERSION            | Reader version number                                |
| MTR_PARAM_READER_IP                 | Reader IP address                                    |
| MTR_PARAM_FREQUENCY_REGION          | Frequency regulatory of the reader                   |
| MTR_PARAM_FREQUENCY_HOPTABLE        | The frequency hopping table of the reader            |
| MTR_PARAM_POTL_GEN2_TARGET          | Gen2 target  |

|                                     |  |
|-------------------------------------|--|
| MTR_PARAM_POTL_GEN2_WRITEMODE       | Gen2 writing mode  |
| MTR_PARAM_TAGDATA_UNIQUEBYANT       | Whether the buffer tag list is unique by antenna         |
| MTR_PARAM_TAGDATA_UNIQUEBYEMDDATA   | Whether the buffer tag list is unique by additional data |
| MTR_PARAM_TAGDATA_RECORDHIGHESTRSSI | Whether to record only the highest RSSI                  |
| MTR_PARAM_RF_TEMPERATURE            | Reader temperature                                       |
| MTR_PARAM_RF_HOPTIME                | Frequency hopping time                                   |
| MTR_PARAM_RF_LBT_ENABLE             | Whether to enable LBT function                           |
| MTR_PARAM_POTL_ISO180006B_BLF       | ISO-6B-BLF backward link rate                            |
| MTR_PARAM_TRANS_TIMEOUT             | Transmission time in milliseconds                        |
| MTR_PARAM_TAG_EMDSECUREREAD         | Embedded encrypted security data                         |
| MTR_PARAM_POWERSAVE_MODE            | RF power mode  |
| MTR_PARAM_RF_ANTPORTS_VSWR          | Antenna port VSWR value                                  |
| MTR_PARAM_INTERNALSAVECONFIGURATION | Default configuration of the integrated device           |
| MTR_PARAM_CUSTOM                    | Extended custom parameters                               |
| MTR_PARAM_READER_WATCHDOG           | Watchdog parameters                                      |
| MTR_PARAM_READER_ERRORDATA          | Error message  |
| MTR_PARAM_RF_HOPANTTIME             | Antenna dwell time during fast mode inventory            |
| MTR_PARAM_TAG_MULTISELECTORS        | Multiple tag filtering rules, maximum number is 16       |
| MTR_PARAM_SAVEINMODULE              | Module power-on default value                            |
| MTR_PARAM_SAVEINMODULE_BAUD         | Module power-on default baud rate value                  |
| MTR_PARAM_TAG_OPOTL                 | Current tag protocol                                     |

### 1.2.3 SL\_TagProtocol Enumeration

Air interface protocol.

| Member                           | Description           |
|----------------------------------|-----------------------|
| SL_TAG_PROTOCOL_NONE             | None                  |
| SL_TAG_PROTOCOL_ISO180006B       | ISO 18000-6b          |
| SL_TAG_PROTOCOL_GEN2             | GEN2                  |
| SL_TAG_PROTOCOL_ISO180006B_UCODE | ISO 18000-6B-UCODE    |
| SL_TAG_PROTOCOL_IPX64            | IPX64                 |
| SL_TAG_PROTOCOL_IPX256           | IPX256                |
| SL_TAG_PROTOCOL_GB               | GB, national standard |

### 1.2.4 Region\_Conf Enumeration

Frequency region in which the reader operates.

| Member      | Description                                     |
|-------------|---|
| RG_NA       | North America (902-928 MHz)                     |
| RG_EU       | Europe, ETSI EN 302 208                         |
| RG_EU2      | Europe, ETSI EN 300 220                         |
| RG_EU3      | Europe, revised ETSI EN 302 208 (865-867 MHz)   |
| RG_KR       | Korean (917-921 MHz)                            |
| RG_PRC      | Chinese (920-925 MHz)                           |
| RG_OPEN     | No regulatory compliance enforced (860-960 MHz) |
| RG_IN       | India (865-867 MHz)                             |
| RG_JP       | Japan (916-921 MHz, no LBT)                     |
| RG_HK       | Hong Kong (920-925 MHz)                         |
| RG_TAIWAN   | Taiwan (920-927 MHz)                            |
| RG_MALAYSIA | Malaysia (919-923 MHz)                          |

|                 |                                  |
|-----------------|----------------------------------|
| RG_SOUTH_AFRICA | South Africa (915-918 MHz)       |
| RG_BRAZIL       | Brazil (902-927 MHz)             |
| RG_THAILAND     | Thailand (920-925 MHz)           |
| RG_SINGAPORE    | Singapore (920-925 MHz)          |
| RG_AUSTRALIA    | Australia (920 – 925 MHz)        |
| RG_URUGUAY      | Uruguay (916-927 MHz)            |
| RG_VIETNAM      | Vietnam (918-922 MHz)            |
| RG_ISRAEL       | Israel (916 MHz)                 |
| RG_PHILIPPINES  | Philippines (918-920 MHz)        |
| RG_INDONESIA    | Indonesia (917-922 MHz)          |
| RG_NEW_ZEALAND  | New Zealand (922-927 MHz)        |
| RG_PERU         | Peru (916-927 MHz)               |
| RG_RUSSIA       | Russia (916-920 MHz)             |
| RG_JP2_LBT6     | Japan (916-921 MHz, with LBT)    |
| RG_JP3_NLBT19   | Japan (916-924 MHz, without LBT) |

### 1.2.5 Lock\_obj Enumeration

To indicate which memory objects of a tag are to be locked in the function LockTag.

| Member                    | Description     |
|---------------------------|-----------------|
| LOCK_OBJECT_KILL_PASSWORD | Kill password   |
| LOCK_OBJECT_ACCESS_PASSWD | Access password |
| LOCK_OBJECT_BANK1         | Bank 1          |
| LOCK_OBJECT_BANK2         | Bank 2          |
| LOCK_OBJECT_BANK3         | Bank 3          |

### 1.2.6 Lock\_Type Enumeration

To indicate how to lock a tag in the function LockTag. After locking, the specified area cannot be read or written. Without unlocking, a password is required for reading and writing. When locking the EPC or User area, the area can be read but not written. The TID area is generally permanently locked at the factory and can only be read.

| Member                    | Description                            |
|---------------------------|--|
| KILL_PASSWORD_UNLOCK      | Unlock kill password                   |
| KILL_PASSWORD_LOCK        | Temporarily lock kill password         |
| KILL_PASSWORD_PERM_LOCK   | Permanently lock kill password         |
| KILL_PASSWORD_PERM_UNLOCK | Permanently unlock the kill password   |
| ACCESS_PASSWD_UNLOCK      | Unlock access password                 |
| ACCESS_PASSWD_LOCK        | Temporarily lock access password       |
| ACCESS_PASSWD_PERM_LOCK   | Permanently lock access password       |
| ACCESS_PASSWD_PERM_UNLOCK | Permanently unlock the access password |
| BANK1_UNLOCK              | Unlock bank 1                          |
| BANK1_LOCK                | Temporarily lock bank 1                |
| BANK1_PERM_LOCK           | Permanently lock bank 1                |
| BANK1_PERM_UNLOCK         | Permanently unlock bank 1              |
| BANK2_UNLOCK              | Unlock bank 2                          |
| BANK2_LOCK                | Temporarily lock bank 2                |
| BANK2_PERM_LOCK           | Permanently lock bank 2                |
| BANK2_PERM_UNLOCK         | Permanently unlock bank 2              |
| BANK3_UNLOCK              | Unlock bank 3                          |
| BANK3_LOCK                | Temporarily lock bank 3                |

|                   |                           |
|-------------------|---------------------------|
| BANK3_PERM_LOCK   | Permanently lock bank 3   |
| BANK3_PERM_UNLOCK | Permanently unlock bank 3 |

### 1.2.7 GpiTrigger\_Type Enumeration

Defines the trigger type for GPI triggered inventory.

| Member                                    | Description  |
|---|--|
| GPITRIGGER_NONE                           | Don't use GPI trigger  |
| GPITRIGGER_TRI1START_TRI2STOP             | Start triggered by condition 1 and stop triggered by condition 2           |
| GPITRIGGER_TRI1START_TIMEOUTSTOP          | Start triggered by condition 1 and stop after timeout                      |
| GPITRIGGER_TRI1ORTRI2START_TIMEOUTSTOP    | Start triggered by condition 1 or 2 and stop after timeout                 |
| GPITRIGGER_TRI1ORTRI2START_TRI1ORTRI2STOP | Start triggered by condition 1 or 2 and stop triggered by condition 1 or 2 |

## 1.3 Structures

### 1.3.1 AntPortsVSWR Structure

| Member      | Type            | Description   |
|-------------|-----------------|---|
| andid       | int             | The antenna ID for detecting standing wave, numbered from 1 |
| power       | unsigned short  | Detected transmission power                                 |
| region      | Region_Conf     | Detected frequency band region                              |
| frequencies | int[]           | Frequency values  |
| vswrs       | unsigned char[] | Returned standing wave value                                |
| frecount    | int             | Frequency value element group length                        |

### 1.3.2 TAGINFO Structure

The detailed information of the inventoried tag.

| Member          | Type             | Description  |
|-----------------|------------------|--|
| ReadCnt         | unsigned char    | The number of times the tag is inventoried.  |
| RSSI            | unsigned char    | Received signal strength of tag.   |
| AntennaID       | unsigned char    | The antenna ID by which the tag is inventoried.  |
| Frequency       | unsigned int     | The frequency on which the tag was inventoried.  |
| TimeStamp       | unsigned int     | The time the tag was inventoried, relative to the time the command of inventory was issued in millisecond. |
| EmbeddedDataLen | unsigned short   | The length of embedded data, in bytes.   |
| EmbeddedData    | unsigned char[]  | Embedded data, another bank data read when inventory.  |
| Res             | unsigned char[2] | Reserved   |
| Epclen          | unsigned short   | The length of EPC code, in bytes.  |
| PC              | unsigned char[2] | PC field of EPC bank   |
| CRC             | unsigned char[2] | CRC field of EPC bank  |
| EpcId           | unsigned char[]  | EPC code   |
| Phase           | int              | The phase on which the tag was inventoried.  |
| protocol        | SL_TagProtocol   | Air interface protocol of tag  |

### 1.3.3 TagSelector\_ST Structure

Similar to TagFilter\_ST, with the only difference being that there is no isInvert member.

| Member | Type | Description |
|--------|------|-------------|
|--------|------|-------------|

|           |                 |   |
|-----------|-----------------|---|
| bank      | int             | The memory bank to be matched, legal values are 0,1,2,3,4. 0—3 means gen2 tag's bank0-3; 4 means iso180006b memory. |
| startaddr | unsigned int    | The memory bank offset, in bits, at which to begin comparing the fdata.   |
| slen      | int             | The length, in bits, of the sdata   |
| sdata     | unsigned char[] | The data to be compared   |

### 1.3.4 GpiState\_ST Structure

The status of a single GPI pin.

| Member | Type | Description                  |
|--------|------|------------------------------|
| GpiId  | int  | GPI number (starting from 1) |
| State  | int  | GPI state                    |

### 1.3.5 GpiTrigger Structure

Used to set the parameters of the GPI trigger inventory interface.

| Member             | Type            | Description   |
|--------------------|-----------------|---|
| GpiTrigger1States  | GpiInfo_ST      | Trigger configuration for condition 1   |
| GpiTrigger2States  | GpiInfo_ST      | Trigger configuration for condition 2   |
| GpiTriggerType     | GpiTrigger_Type | Trigger type  |
| StopTriggerTimeout | int             | When the TriggerType has _TIMEOUTSTOP, this defines the timeout in milliseconds |

### 1.3.6 Inv\_Potl Structure - Deprecated

The air protocol for inventory operations.

| Member | Description   |
|--------|---|
| potl   | The air protocol of a tag   |
| weight | The proportion of inventory time occupied by the protocol when using multiple protocols |

### 1.3.7 Inv\_Potls\_ST Structure - Deprecated

| Member  | Description                    |
|---------|--------------------------------|
| potlcnt | The number of elements / potls |
| potls   | The air protocol of a tag      |

### 1.3.8 GpiInfo\_ST Structure

| Member   | Type          | Description                       |
|----------|---------------|-----------------------------------|
| gpiCount | int           | The number of elements / gpiStats |
| gpiStats | GpiState_ST[] | The status of GPI pins            |

### 1.3.9 TagFilter\_ST Structure

Rules for filtering tags. Tag inventory and access operations can only be applied to tags matching the filters.

| Member    | Type           | Description   |
|-----------|----------------|---|
| bank      | int            | The memory bank to be matched, legal values are 0,1,2,3,4. 0—3 means gen2 tag's bank0-3; 4 means iso180006b memory. |
| startaddr | unsigned int   | The memory bank offset, in bits, at which to begin comparing the fdata.   |
| flen      | int            | The length, in bits, of the fdata   |
| fdata     | unsigned char* | The data to be compared   |

|          |     |  |
|----------|-----|--|
| isInvert | int | 0 means matching the filter criteria, 1 means matching the inverted filter criteria. |
|----------|-----|--|

Example: Filter tag with EPC code starting with 0x1234

```

1. TagFilter_ST tf;
2. unsigned char data[2];
3. data[0] = 0x12;
4. data[1] = 0x34;
5. tf.bank = 1;
6. tf.startaddr = 32;
7. tf.flen = 16;
8. tf.fdata = data;
9. tf.isInvert = 0;

```

### 1.3.10 EmbeddedData\_ST Structure

Used to specify the details of reading a memory bank during inventory.

| Member    | Type           | Description  |
|-----------|----------------|--|
| bank      | int            | Specifies which bank to read during inventory, legal values are 0, 1, 2, 3 |
| startaddr | int            | The memory bank offset, in blocks, at which to begin reading               |
| bytecnt   | int            | The number of bytes to read starting at startaddr, max. 64 bytes           |
| accesspwd | unsigned char* | Access password, if no password is required it can be set to null          |

**Example:** Read two blocks from the USER bank starting at the 4<sup>th</sup> block with access password 0x11112222

```

1. EmbeddedData_ST emd;
2. unsigned char accpwd[4];
3. accpwd[0] = 0x11;
4. accpwd[1] = 0x11;
5. accpwd[2] = 0x22;
6. accpwd[3] = 0x22;
7. emd.bank = 3;
8. emd.startaddr = 4;
9. emd.bytecnt = 4;
10. emd.accesspwd = accpwd;

```

### 1.3.11 AntPower Structure

The Tx power of one antenna.

| Member     | Type           | Description   |
|------------|----------------|---|
| antid      | int            | The antenna ID, numbered from 1   |
| readPower  | unsigned short | Transmit power in centi-dbm when the reader performs a read-related operation.  |
| writePower | unsigned short | Transmit power in centi-dbm when the reader performs a write-related operation. |

### 1.3.12 AntPowerConf Structure

Tx power configuration of antennas

| Member | Type        | Description                        |
|--------|-------------|------------------------------------|
| antcnt | int         | The number of elements of Powers   |
| Powers | AntPower[ ] | Tx power configuration of antennas |

### 1.3.13 ConnAnts\_ST Structure

The antennas to which the reader is connected

| Member        | Type   | Description                                   |
|---------------|--------|---|
| antcnt        | int    | The number of elements of connected ants      |
| connectedants | int[ ] | The antennas to which the reader is connected |

#### 1.3.14 HoptableData\_ST Structure

The frequency hopping table of the reader.

| Member | Type            | Description  |
|--------|-----------------|--|
| htb    | unsigned int[ ] | The frequency hopping table, the frequency values must be within the defined frequency band (in KHz) |
| lenhtb | int             | The number of elements in htb, maximum length is 100   |

#### 1.3.15 MultiTagSelectors\_ST Structure

Filter rule group can define up to 16 filter rules that are logically related to each other.

| Member         | Type              | Description                          |
|----------------|-------------------|--------------------------------------|
| multiselectors | TagSelector_ST[ ] | Tag filter rule array                |
| selectorcnt    | int               | Number of elements in multiselectors |

#### 1.3.16 OnTagReadBlock Structure

| Member  | Type           | Description   |
|---------|----------------|---|
| handler | TagReadHandler | Callback for when a tag is inventoried (asynchronous inventory) |
| cookie  | void*          | User data, will be passed to handler as parameter               |

#### 1.3.17 OnGpiTriggerBlock Structure

| Member     | Type                      | Description                                      |
|------------|---------------------------|--|
| gshandler  | GpiStateHandler           | Notifies when trigger conditions 1 and 2 are met |
| gtbhandler | GpiTriggerBoundaryHandler | Notifies the reader to start and stop inventory  |
| cookie     | void*                     | User-defined data                                |

#### 1.3.18 OnReadingErrorBlock Structure

Asynchronous storage error event handling.

| Member  | Type                | Description                                 |
|---------|---------------------|---|
| handler | ReadingErrorHandler | Function pointer to the processing function |
| cookie  | void*               | User-defined data                           |

#### 1.3.19 CustomParam\_ST Structure

Parameter value type when the keyword is MTR\_PARAM\_CUSTOM.

| Member    | Type  | Description   |
|-----------|-------|---|
| pCusParam | void* | <p>The first byte determines the custom operation in the following three cases:</p> <ul style="list-style-type: none"> <li>“0x01”: Indicates a carrier test and the parameter value byte sequence is: on or off (1 byte, 1 on, 0 off) + antenna id (1 byte) + power (2 bytes) + frequency (4 bytes)</li> <li>“0x02”: Switching the module to the BootLoader layer</li> <li>“0x03”: Reading and writing registers: 4-byte address + 4-byte value</li> </ul> <p>In all other cases, the first 50 bytes are the parameter name and after these the specific parameter values:</p> <ul style="list-style-type: none"> <li>“reader/rdrdetails”: 16 bytes reserved + 4 bytes hardware version + 12 bytes serial number</li> </ul> |

|           |     |   |
|-----------|-----|---|
|           |     | <ul style="list-style-type: none"> <li>“reader/moduleinfo”: 16 bytes reserved + 16 bytes version information (4 bytes bootloader + 4 bytes hardware version + 4 bytes firmware date + 4 bytes firmware version) + 12 bytes serial number</li> <li>“reader/macaddr”: 6 bytes</li> <li>“Reader/Ex10fastmode”: mode (1 byte, 1 is Ex fast mode, 0 is normal mode) + fixed byte to 20 + use case (1 byte, 0 is many labels, 1 is small number of labels) + 19 bytes (all 0)</li> <li>“tagcustomcmd/fastid”: 1 byte (1 on, 0 off)</li> <li>“tagcustomcmd/tagfocus”: 1 byte (1 on, 0 off)</li> <li>“gen2op/multiembeddeddata”: number of additional data items (1 byte) + password (4 bytes) + additional 1 bank (1 byte) + additional 1 address (4 bytes) + additional 1 block number (1 byte) + additional 2 bank (1 byte) + ...</li> <li>“Reader/Savestandby”: 1 byte (1 open, 0 closed)</li> <li>“R2000/oemregister”: 4-byte address + 4-byte value</li> <li>“Reader/dutycycle”: duty cycle execution before inventory (2 bytes, in milliseconds) + duty cycle execution base time (2 bytes, in milliseconds)</li> <li>“Reader/powermode”: 1 byte (0x00 full power, 0x01 min saving, 0x02 med saving, 0x03 max saving)</li> <li>“Reader/rssifilter”: when getting parameters 4 bytes all 0, when setting parameters to cancel internal filtering 1 byte 0x01 + 3 bytes all 0, when setting to enable internal filtering 1 byte 0x01 + 1 byte 0xAA + 1 byte signed RSSI value + 1 byte 0x00</li> </ul> |
| CParamlen | int | Byte length of pCusParam. When pCusParam is an incoming parameter, CParamlen is only the length of the specific parameter value after 50 bytes, if pCusParam is an outgoing parameter, CParamlen includes the length of the first 50 bytes.   |

### 1.3.20 Default\_Param Structure

Used to set the default parameters for when the module is powered on.

| Member    | Type      | Description  |
|-----------|-----------|--|
| key       | Mtr_Param | Defined power-on default parameter keyword: <ul style="list-style-type: none"> <li>MTR_PARAM_SAVEINMODULE_BAUD: val type is int[1] with possible values 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800 or 921600</li> <li>MTR_PARAM_FREQUENCY_REGION: val type is byte[1]</li> <li>MTR_PARAM_RF_ANTPOWER: val type is unsigned short[], serial antenna number 1, read power 1, write power 1, antenna switching time (fixed to 0), antenna number 2, read power 2, ...</li> <li>MTR_PARAM_POTL_GEN2_SESSION: val type is int[1]</li> <li>MTR_PARAM_POTL_GEN2_Q: val type is int[1]</li> <li>MTR_PARAM_POTL_GEN2_TARGET: val type is int[1]</li> <li>MTR_PARAM_SAVEINMODULE: val type is int[1]</li> </ul> |
| customkey | void*     | When key is MTR_PARAM_SAVEINMODULE, subkey can be <ul style="list-style-type: none"> <li>“modulesave/hpupload”: Customize HP function and upload actively</li> <li>“modulesave/antport”: The default antenna port is used when powered on</li> <li>“modulesave/default”: Restore the initial power-on default value</li> </ul>   |
| isdefault | int       | When modifying the custom default configuration, it should be set to false. When set to true, the default value will be used and the value of val will be ignored.   |
| val       | void*     | Parameter value  |

### 1.3.21 Reader\_Ip Structure

| Member  | Type     | Description            |
|---------|----------|------------------------|
| ip      | char[50] | Reader IP address      |
| mask    | char[50] | Reader network mask    |
| gateway | char[50] | Reader network gateway |

### 1.3.22 BackReadOption Structure

The detailed configuration of asynchronous inventory.

| Member             | Type           | Description   |
|--------------------|----------------|---|
| ReadDuration       | unsigned short | Inventory cycle in milliseconds when using common asynchronous inventory mode. If using high-speed asynchronous inventory mode the attribute does nothing.  |
| ReadInterval       | unsigned int   | The idle interval in milliseconds between two inventory cycles when using common asynchronous inventory mode. If using high-speed asynchronous inventory mode the attribute does nothing.   |
| IsFastRead         | int            | If 1, it uses high speed asynchronous inventory mode, otherwise it uses common asynchronous inventory mode.   |
| FastReadDutyRation | unsigned char  | The upper 4 bits must be 0, the lower 4 bits are valid, and the lower 4 bits are used to indicate the proportion of time spent resting during the reader's inventory. If the FastReadDutyRation is 0-9 the proportion is FastReadDutyRation * 5%. If the FastReadDutyRation is 10-15 the proportion is 50%. |
| TMFlags            | TagMetaFlags   | Specifies which metadata information the tag carries when using the high-speed asynchronous inventory mode. If using common asynchronous inventory mode the attribute does nothing.   |
| GpiTrigger         | GPITrigger     | The GPI trigger condition used for starting or stopping asynchronous inventory if IsGPITrigger is true.   |
| IsGPITrigger       | int            | If true, starting and stopping asynchronous inventory is triggered by GPI trigger condition.  |

### 1.3.23 TagMetaFlags Structure

Specifies which metadata information the tag carries when using the high-speed asynchronous inventory mode. Setting the members in the following table to true means asking the reader to return the corresponding information data, otherwise set to false.

| Member      | Type | Description  |
|-------------|------|--|
| IsAntennaID | int  | The antenna ID on which the tag was inventoried  |
| IsReadCnt   | int  | The read count of the tag  |
| IsRSSI      | int  | Received Signal Strength Indication of the tag   |
| IsFrequency | int  | The frequency on which the tag was inventoried   |
| IsTimestamp | int  | The time stamp when the tag was inventoried  |
| IsRFU       | int  | The reserved field   |
| IsEmdData   | int  | The bank data of the tag. During inventory the reader can read tag data of another bank depending on the EmbeddedCmdOfInventory parameter. |
| IsProtocol  | int  | The protocol   |

### 1.3.24 HardwareDetails Structure

Refer to the method GetHardwareDetails.

| Member        | Type           | Description                           |
|---------------|----------------|---------------------------------------|
| module        | Module_Type    | Module type of the reader             |
| board         | MainBoard_Type | Mainboard type of the reader          |
| logictype     | Reader_Type    | Type of the reader                    |
| selfcheckants | int            | Number of antenna ports for self-test |

## 2 Reader Life Cycle

First call `InitReader_Notype` to initialize the reader. Afterwards call the functions `ParamSet` and `ParamGet` to configure the reader. Once configured, tag operation functions can be called. When the reader is no longer used, call `CloseReader`.

### 2.1 InitReader\_Notype Function

Initialize a reader and create a handle.

```
1. READER_ERR InitReader_Notype(int* hReader, char* src, int antscnt)
```

| Parameter | Description  |
|-----------|--|
| hReader   | Handle of the reader if the function executes successfully; used as return parameter |
| src       | IP address or serial port number of a reader, eg COM1                                |
| antscnt   | Number of physical ports of the reader   |

Example:

```
1. int hreader;  
2. READER_ERR err = InitReader_Notype(&hreader, "192.168.1.100", 4);
```

### 2.2 CloseReader Function

Close reader and release related resources.

```
1. void CloseReader(int hReader);
```

| Parameter | Description          |
|-----------|----------------------|
| hReader   | Handle of the reader |

## 3 Reader Parameters

The `ParamGet` and `ParamSet` functions are to get and set the readers' parameters. Every parameter of the reader is represented by a key value which is of the `Mtr_Param` enumeration type. The parameters can be divided into read only and read/write parameters. A parameter of the reader will be in effect until it is set to another value.

### 3.1 Get Parameter

`ParamGet` can be called anytime during the lifetime of a reader.

```
1. READER_ERR ParamGet(int hReader, Mtr_Param key, void* val)
```

| Parameter | Description                               |
|-----------|---|
| hReader   | Handle of the reader                      |
| key       | Key of a parameter                        |
| val       | Used to return the value of the parameter |

Example: Get the Gen2 Session parameter

```
1. int sess;
```

```
2. READER_ERR err = ParamGet(hReader, MTR_PARAM_POTL_GEN2_SESSION, &sess);
```

### 3.2 Set Parameter

ParamSet can be called anytime during the lifetime of a reader. Once a parameter was set it will stay in effect until the end of the reader's lifetime or until it is modified.

```
1. READER_ERR ParamSet(int hReader, Mtr_Param key, void* val)
```

| Parameter | Description                               |
|-----------|---|
| hReader   | Handle of the reader                      |
| key       | Key of a parameter                        |
| val       | Used to return the value of the parameter |

Example: Set the Gen2 Session parameter as Session 0

```
1. int sess = 0;
2. READER_ERR err = ParamSet(hReader, MTR_PARAM_POTL_GEN2_SESSION, &sess);
```

### 3.3 Default Configuration

When the reader is powered on and initialized, the module reads the parameters stored in the internal registers and uses the values to configure the reader after powering on. Modifying the default configuration is permanent, meaning the parameter values won't be lost due to power failure of the reader.

**Example:** Set the serial baud rate of the reader to 921600.

```
1. Default_Param dp;
2. dp.isdefault = false;
3. dp.key = MTR_PARAM_SAVEINMODULE_BAUD;
4. dp.val = 921600;
5. err = ParamSet(hreader, MTR_PARAM_SAVEINMODULE, &dp);
```

### 3.4 Custom Parameter Configuration

Mainly for adapting to the expansion of reader functions and related differences between different products.

**Example:** Enable fastid.

```
1. CustomParam_ST cpara;
2. unsigned char paraBuf_fastid[51] = "tagcustomcmd/fastid";
3. char out[100];
4. paraBuf_fastid[50] = 1;
5. cpara.pCusParam = paraBuf_fastid;
6. cpara.CParamlen = 1;
7. err = ParamSet(hreader, MTR_PARAM_CUSTOM, &cpara);
```

### 3.5 Parameters

#### 3.5.1 Read/write parameters

| Key                         | Value Type | Description   |
|-----------------------------|------------|---|
| MTR_PARAM_POTL_GEN2_SESSION | int*       | Legal values: 0,1,2,3. For few but fast-moving tags use Session 0; for many but slow-moving tags use Session 1. |
| MTR_PARAM_POTL_GEN2_Q       | int*       | Legal values: -1 to 15. (-1: automatically adjust Q value; 0 to 15: static Q value)                             |

|                                     |                       |  |
|-------------------------------------|-----------------------|--|
| MTR_PARAM_POTL_GEN2_TAGENCODING     | int*                  | R2000 chip supports profile1-profile5:<br>value (0x10-0x15)<br>The E series code value supports 203,<br>111 (RF_MODE_11), 220, 101<br>(RF_MODE_1), 45, 115 (RF_MODE_15),<br>112 (RF_MODE_12), 103 (RF_MODE_3),<br>105 (RF_MODE_5), 107 (RF_MODE_7),<br>113 (RF_MODE_13)<br>The SFM series code value, 6C protocol<br>(100-109), GB protocol (110-118), 6B<br>protocol (119),<br>GB2 protocol (120-132) |
| MTR_PARAM_RF_ANTPOWER               | AntPowerConf          | Note: the default power may not be the<br>maximum transmitting power of reader   |
| MTR_PARAM_TAG_FILTER                | TagFilter_ST          | When you don't use filter criteria you<br>could set to NULL.   |
| MTR_PARAM_TAG_EMBEDDEDATA           | EmbeddedData_ST       | To disable this feature please set it to<br>NULL.  |
| MTR_PARAM_READER_IS_CHK_ANT         | int*                  | 1 is enabled, 0 is disabled. After<br>enabling, the antenna is detected<br>before transmitting power, if no<br>antenna is connected, an error is<br>returned   |
| MTR_PARAM_READER_IP                 | Reader_Ip             | IP address information   |
| MTR_PARAM_FREQUENCY_REGION          | Region_Conf           | Operating frequency band   |
| MTR_PARAM_FREQUENCY_HOPTABLE        | HoptableData_ST       | The frequency point must be within the<br>spectrum of region regulatory.   |
| MTR_PARAM_POTL_GEN2_WRITEMODE       | int*                  | Legal values: 0, 1<br>(0: write in words;1: write in blocks)   |
| MTR_PARAM_POTL_GEN2_TARGET          | int*                  | 0: A; 1: B; 2: A->B; 3: B->A   |
| MTR_PARAM_TAGDATA_UNIQUEBYANT       | int*                  | Legal values: 0,1 (0: no matter how<br>many antennas read the tag there<br>would be only one tag record; 1:<br>different tag records when the same tag<br>was read by different antennas)  |
| MTR_PARAM_TAGDATA_UNIQUEBYEMDDATA   | int*                  | Legal values: 0,1 (0: regard as one tag<br>record; 1: regard as multiply tag<br>records)   |
| MTR_PARAM_TAGDATA_RECORDHIGHESTRSSI | int*                  | Legal value: 0,1 (0: record the highest<br>RSSI value; 1 : not to record the highest<br>RSSI value)  |
| MTR_PARAM_RF_HOPTIME                | int*                  | Frequency hopping time in milliseconds   |
| MTR_PARAM_TRANS_TIMEOUT             | int*                  | Transmission time in milliseconds  |
| MTR_PARAM_TAG_EMDSECUREREAD         | EmbeddedSecureRead_ST | The member tagtype of<br>EmbeddedSecureRead_ST indicates the<br>tag type, 1: Alien Higgs3; 2: Impinj<br>Monza 4, other values are illegal;<br>pwdtype indicates the password type,<br>1: fixed password; 2: calculated<br>password, other values are illegal;  |

|                              |                      |   |
|------------------------------|----------------------|---|
|                              |                      | ApIndexStartBitsInEpc indicates the starting epc bit position as the password index; ApIndexBitsNumInEpc indicates the number of bits as the password index; bank indicates which bank to read; address indicates the starting address in the bank (in blocks); blkcnt indicates the number of blocks to read; accesspwd indicates the access password (if fixed password mode is used) |
| MTR_PARAM_POWERSAVE_MODE     | int*                 | Power saving level, 0 no power saving, level 3 highest power saving   |
| MTR_PARAM_RF_ANTPORTS_VSWR   | AntPortsVSWR         | VSWR value reflects the compatibility of the antenna port and the antenna. The smaller the value the better   |
| MTR_PARAM_CUSTOM             | CustomParam_ST       | Extended custom parameters  |
| MTR_PARAM_READER_WATCHDOG    | unsigned char*       | Watchdog parameters   |
| MTR_PARAM_READER_ERRORDATA   | unsigned char*       | Error message   |
| MTR_PARAM_RF_HOPANTTIME      | int*                 | Antenna dwell time during fast mode inventory   |
| MTR_PARAM_TAG_MULTISELECTORS | MultiTagSelectors_ST | Multi-tag filtering rules, max. 16 rules  |
| MTR_PARAM_SAVEINMODULE       | Default_Param        | Reader power-on default value   |
| MTR_PARAM_TAG_OPPOTL         | int*                 | Current tag protocol  |
| MTR_PARAM_TAG_INVPOTL        | Inv_Potls_ST         | This parameter must be set before calling the StartReading and TagInventory_Raw methods. Then the reader only supports the Gen2 protocol. - Deprecated  |

### 3.5.2 Read-only Parameters

| Key                                 | Value Type      | Description  |
|-------------------------------------|-----------------|--|
| MTR_PARAM_READER_CONN_ANTs          | ConnAnts_ST     | Not all kinds of antennas can be detected.                         |
| MTR_PARAM_READER_AVAILABLE_ANTPORTS | int*            | The maximum number of antennas that can be connected to the reader |
| MTR_PARAM_RF_MAXPOWER               | unsigned short* | The maximum power [centi-dBm]                                      |
| MTR_PARAM_RF_MINPOWER               | unsigned short* | The minimum power [centi-dBm]                                      |
| MTR_PARAM_READER_VERSION            | Reader_Ver      | Reader version number  |
| MTR_PARAM_RF_TEMPERATURE            | int*            | Reader temperature   |

## 4 Synchronous Inventory

Synchronous inventory is implemented using the TagInventory\_Raw and GetNextTag functions. TagInventory\_Raw blocks until the end of the inventory. When the user needs to perform a short inventory operation, and the number of tags is small, the synchronous inventory is a good choice.

Side note: the MTR\_PARAM\_TAG\_INVPOTL parameter must be set before calling the above methods, otherwise it will return an error.

## 4.1 TagInventory\_Raw Function

Implements synchronous inventory operation. After calling this function, one should immediately call the `GetNextTag` function to get the inventoried tags.

```
1. READER_ERR TagInventory_Raw(int hReader, int* ants, int antcnt, unsigned short timeout, int* tagcnt)
```

| Parameter            | Description  |
|----------------------|--|
| <code>hReader</code> | Handle of the reader   |
| <code>ants</code>    | Antenna ids used for this operation  |
| <code>antcnt</code>  | The number of antennas in the <code>ants</code> array                                    |
| <code>timeout</code> | The time period for the operation [ms]; the function blocks until the timeout is expired |
| <code>tagcnt</code>  | Output parameter, the number of read tags  |

**Example:** TagInventory with antenna 1, 3 and 4

```
1. int ants[] = {1, 3, 4};
2. int antcnt = 3;
3. int tagnum;
4. if (TagInventory_Raw(hreader, ants, antcnt, 1000, &tagnum) != MT_OK_ERR)
5. {
6.     printf("TagInventory_Raw failed");
7. }
```

## 4.2 GetNextTag Function

Get the next tag. Users could get the number of tags inventoried by calling `TagInventory_Raw` function and then execute the `GetNextTag` function as many times as the number of tags to get all tags inventoried.

```
1. READER_ERR GetNextTag(int hReader, TAGINFO* pTInfo)
```

| Parameter            | Description                                  |
|----------------------|--|
| <code>hReader</code> | Handle of the reader                         |
| <code>pTInfo</code>  | Output parameter, used to store the tag data |

**Example:**

```
1. TAGINFO tag;
2. if (GetNextTag(hreader, &tag) != MT_OK_ERR)
3. {
4.     printf("GetNextTag failed");
5. }
```

## 5 Fast Mode Inventory

The fast mode inventory does not buffer the read tags inside the reader but uploads them while reading and is therefore real-time. Once the fast mode is started using the method `AsyncStartReading`, the reader will always be in working state. The `AsyncGetTagCount` method can operate in intervals of 50 ms to 1000 ms. Therefore, a timer or thread is required in the program design to maintain this frequency. When the number of tags returned is greater than 0, the method `AsyncGetNextTag` can be called to retrieve the tags. The method returns data of one tag each time. To stop the fast mode, the method `AsyncStopReading` method must be called.

The fast mode is divided into normal fast mode and EX special fast mode. The later must be supported by the E series reader. The default is the normal fast mode and can be switched by custom parameter configurations.

## 5.1 AsyncStartReading Method

After starting fast mode inventory, the reader is always in inventory state. It does not stop unless there is an exception or the AsyncStopReading method was called.

```
1. READER_ERR AsyncStartReading(int hreader, int* ants, int antcnt, int option)
```

| Parameter | Description  |
|-----------|--|
| hreader   | Handle of the reader   |
| ants      | Operating antennas   |
| antcnt    | Number of elements in ants   |
| option    | The first byte controls whether to enable the antenna group polling notification method. If enabled (0x01), a virtual tag will be generated after polling the antenna once and its epc length is 1.<br>The second and third byte are metaflags as explained in the example below.<br>The lower four bits of the fourth byte are the pause percentage (0 is 0%, 1 is 5%, 2 is 10% ... 10 is 50%, 11 is 55 %, 12 is 60 %, 13 is 70%, 14 is 80% and 15 is 90 %). The highest bit of the fourth byte controls whether to enable the method of sending heartbeat packets to the host every 15 seconds, and one bit lower (bit6) controls whether to enable the function of automatically stopping inventory without new tags. |

### Example:

```
1. // Return all data items
2. int metaflag = 0;
3. metaflag |= 0x0001; // return the number of reads
4. metaflag |= 0x0002; // return the RSSI signal
5. metaflag |= 0x0004; // Return the antenna number
6. metaflag |= 0x0008; // Return the frequency
7. metaflag |= 0x0010; // Return the timestamp
8. metaflag |= 0x0020; // Return the phase value
9. metaflag |= 0x0040; // Return the tag protocol value
10. metaflag |= 0x0080; // Return additional data
11. // Pause ratio: 50% (ratio of read time and interval)
12. int option = (metaflag << 8) | 10;
13. // enable heartbeat
14. option |= 0x80;
15. // enable automatic stop function
16. option |= 0x40;
17. // enable antenna polling notification function
18. option |= (0x01 << 24);
```

## 5.2 AsyncGetTagCount Method

Get the number of tags read in the buffer in fast mode. It is recommended that the query time interval is more than 50 ms.

```
1. READER_ERR AsyncGetTagCount(int hreader, int[] tagcnt)
```

| Parameter | Description          |
|-----------|----------------------|
| hreader   | Handle of the reader |
| tagcnt    | Number of tags read  |

## 5.3 AsyncGetNextTag Method

Gets the buffered tag data in fast mode.

```
1. READER_ERR AsyncGetNextTag(int hreader, TAGINFO pTInfo)
```

| Parameter | Description                          |
|-----------|--------------------------------------|
| hreader   | Handle of the reader                 |
| pTInfo    | Stores a tag and its data item value |

## 5.4 AsyncStopReading Method

Stop fast mode inventory.

```
1. READER_ERR AsyncStopReading(int hreader)
```

| Parameter | Description          |
|-----------|----------------------|
| hreader   | Handle of the reader |

Example:

```
1. int ants[] = {1};
2. time_t startt, endt;
3. int tagcount = 0;
4. int testseconds = 0;
5. int isbreak = 0;
6. err = AsyncStartReading(hreader, ants, 1, option);
7. if (err == MT_OK_ERR) {
8.     printf("AsyncStartReading successfull\n");
9.     startt = time((time_t*) NULL);
10.    endt = startt;
11. } else {
12.     printf("AsyncStartReading failed. Err: %d\n", err);
13.     return -1;
14. }
15. while (1) {
16.     endt = time((time_t*) NULL);
17.     if ((endt-startt) >= testseconds || isbreak) {
18.         AsyncStopReading(hreader);
19.         break;
20.     } else {
21.         err = AsyncGetTagCount(hreader, &tagcount);
22.         if (err == MT_OK_ERR) {
23.             for (int i = 0; i < tagcount; ++i) {
24.                 TAGINFO pTInfo;
25.                 err = AsyncGetNextTag(hreader, &pTInfo);
26.                 if (err == MT_OK_ERR) {
27.                     char out[124];
28.                     Hex2Str(pTInfo.EpcId, pTInfo.EpcIdLen, out);
29.                     printf("epcid: %s\n", out);
30.                 } else {
31.                     isbreak = 1;
32.                     break;
33.                 }
34.             }
35.         } else {
36.             isbreak = 1;
37.         }
38.     }
39. }
```

```

37.     }
38.     MSLEEP(1000); // sleep 1 second
39. }

```

## 6 Asynchronous Inventory

The SDK can implement asynchronous inventory in two ways. One is to internally use a thread to call the synchronous inventory `TagInventory_Raw` function. This asynchronous inventory is called common asynchronous inventory mode. The other is a true asynchronous inventory, where the reader is in a continuous inventory state. This way, the inventory performance of the reader is optimal, and applications with higher inventory performance requirements should use this mode of asynchronous inventory. This asynchronous inventory is called high-speed asynchronous inventory mode or short: fast mode inventory. The `IsFastRead` parameter in the `BackReadOption` switches between normal inventory and fast mode.

Attention: Not all devices support asynchronous high-speed inventory mode. Only those readers built with R2000, E series and sfm series chips can support this inventory mode.

During asynchronous inventory, if tags are inventoried, the `TagReadHandler` callback will be called to pass the tag data to the user program; if the reader has an abnormal condition, the `ReadingErrorHandler` callback will be called to pass the exception details to the user program; if the asynchronous inventory is configured to start or stop with a GPI trigger, the `GpiStateHandler` and `GpiTriggerBoundaryHandler` callbacks are called when the GPI condition is met.

Asynchronous Inventory Functions:

| Method                    | Description                  |
|---------------------------|------------------------------|
| <code>StartReading</code> | Start asynchronous inventory |
| <code>StopReading</code>  | Stop asynchronous inventory  |

Asynchronous Inventory Callback Interfaces

| Callback interface                     | Description   |
|--|---|
| <code>TagReadHandler</code>            | Called when tags are inventoried  |
| <code>ReadingErrorHandler</code>       | Called when exception occurs  |
| <code>GpiStateHandler</code>           | Called when GPI trigger condition is met if asynchronous Inventory is configured to start by <code>GpiTrigger</code> .                |
| <code>GpiTriggerBoundaryHandler</code> | Called when starting or stopping asynchronous inventory if asynchronous inventory is configured to start by <code>GpiTrigger</code> . |

### 6.1 Start Inventory

Before calling the `StartReading` function to start the asynchronous inventory, the following callbacks and parameters must be set correctly:

- `MTR_PARAM_TAG_INV POTL` parameter must be set, or it will return an error.
- The `TagReadHandler` callback must be set or else the inventoried tags cannot be passed to the user program.
- The `ReadingErrorHandler` callback must be set, else the reader exception state cannot be passed to the user program.

If the reader is currently running an asynchronous inventory, calling the `StartReading` function will return an error. When in doubt of the current state of the reader, first call the `StopReading` function before starting asynchronous inventory.

### 6.1.1 TagReadHandler Callback

Will be called in the event of tags being inventoried during asynchronous inventory operation.

```
1. typedef void (*TagReadHandler)(int hReader, TAGINFO* tag, void* cookie)
```

| Parameter | Description   |
|-----------|---|
| hReader   | Handle of the reader                                    |
| tag       | The inventoried tag                                     |
| cookie    | User data that is set by the SetTagReadHandler function |

### 6.1.2 SetTagReadHandler Function

Set the TagReadHandler callback function for asynchronous inventory.

```
1. READER_ERR SetTagReadHandler(int hReader, OnTagReadBlock bok)
```

| Parameter | Description                      |
|-----------|----------------------------------|
| hReader   | Handle of the reader             |
| bok       | The TagReadHandler and user data |

### 6.1.3 ReadingErrorHandler Callback

Will be called in the event of an error during asynchronous inventory.

```
1. typedef void (*ReadingErrorHandler)(int hReader, READER_ERR errcode, void* cookie)
```

| Parameter | Description  |
|-----------|--|
| hReader   | Handle of the reader   |
| errcode   | Error code that was thrown                                   |
| cookie    | User data that is set by the SetReadingErrorHandler function |

### 6.1.4 SetReadingErrorHandler Function

Set the ReadingErrorHandler callback function for asynchronous inventory.

```
1. READER_ERR SetReadingErrorHandler(int hReader, OnReadingErrorBlock bok)
```

| Parameter | Description                           |
|-----------|---------------------------------------|
| hReader   | Handle of the reader                  |
| bok       | The ReadingErrorHandler and user data |

### 6.1.5 GpiStateHandler Callback

Will be called if asynchronous inventory is configured to be triggered by GPI to retrieve information on GPI states.

```
1. typedef void (*GpiStateHandler)(int hReader, int triggerid, GpiInfo_ST *gpist, void* cookie)
```

| Parameter | Description   |
|-----------|---|
| hReader   | Handle of the reader  |
| triggerid | The condition number that triggered the start or stop of asynchronous inventory |
| gpist     | GPI status when triggering start or stop of asynchronous inventory              |
| cookie    | User data that is set by the SetGpiTriggerHandler function                      |

### 6.1.6 GpiTriggerBoundaryHandler Callback

Will be called to pass the cause of starting or stopping inventory if asynchronous inventory is configured to start by GPI trigger.

```
1. typedef void (*GpiTriggerBoundaryHandler)(int hReader, int bordertype, int reason, void* cookie)
```

| Parameter  | Description  |
|------------|--|
| hReader    | Handle of the reader                                       |
| bordertype | 1 – starting inventory; 2 – stopping inventory             |
| reason     | 1 – by GPI trigger; 2 – by timeout                         |
| cookie     | User data that is set by the SetGpiTriggerHandler function |

### 6.1.7 SetGpiTriggerHandler Function

Set the GpiStateHandler and GpiTriggerBoundaryHandler as callback functions.

```
1. READER_ERR SetGpiTriggerHandler(int hReader, OnGpiTriggerBlock bok)
```

| Parameter | Description  |
|-----------|--|
| hReader   | Handle of the reader   |
| bok       | The GpiTriggerHandler, GpiTriggerBoundaryHandler and user data |

### 6.1.8 StartReading Function

Start the asynchronous inventory.

```
1. READER_ERR StartReading(int hReader, int* ants, int antcnt, BackReadOption* pBRO)
```

| Parameter | Description  |
|-----------|--|
| hReader   | Handle of the reader   |
| ants      | Operating antennas   |
| antcnt    | Number of operating antennas in the ants array                                 |
| pBRO      | Detailed configuration of asynchronous inventory; see BackReadOption structure |

Example: Use antenna 1, 3 and 4 to start common asynchronous inventory.

```
1. int ants[] = {1, 3, 4};
2. int antcnt = 3;
3. BackReadOption Brop;
4. Brop.ReadDuration = 200;
5. Brop.ReadInterval = 10;
6. Brop.IsFastRead = 0;
7. Brop.IsGPITrigger = 0;
8. READER_ERR err = StartReading(hreader, ants, antcnt, &Brop);
```

Example: Use antenna 1, 3 and 4 to start high-speed asynchronous inventory.

```
1. int ants[] = {1, 3, 4};
2. int antcnt = 3;
3. BackReadOption Brop;
4. Brop.IsFastRead = 1;
5. Brop.IsGPITrigger = 0;
6. Brop.TMFlags.IsAntennaID = 1;
```

```

7. Brop.TMFlags.IsReadCnt = 1;
8. Brop.TMFlags.IsRSSI = 1;
9. Brop.TMFlags.IsFrequency = 1;
10. Brop.TMFlags.IsRFU = 0;
11. Brop.TMFlags.IsTimestamp = 0;
12. Brop.TMFlags.IsEmdData = 0;
13. READER_ERR err = StartReading(hreader, ants, antcnt, &Brop);

```

## 6.2 Stop Inventory

Call the StopReading function to stop the asynchronous inventory.

### 6.2.1 StopReading Function

```
1. READER_ERR StopReading(int hReader)
```

| Parameter | Description          |
|-----------|----------------------|
| hReader   | Handle of the reader |

Example:

```
1. READER_ERR err = StopReading(hreader);
```

## 6.3 Additional Data Settings

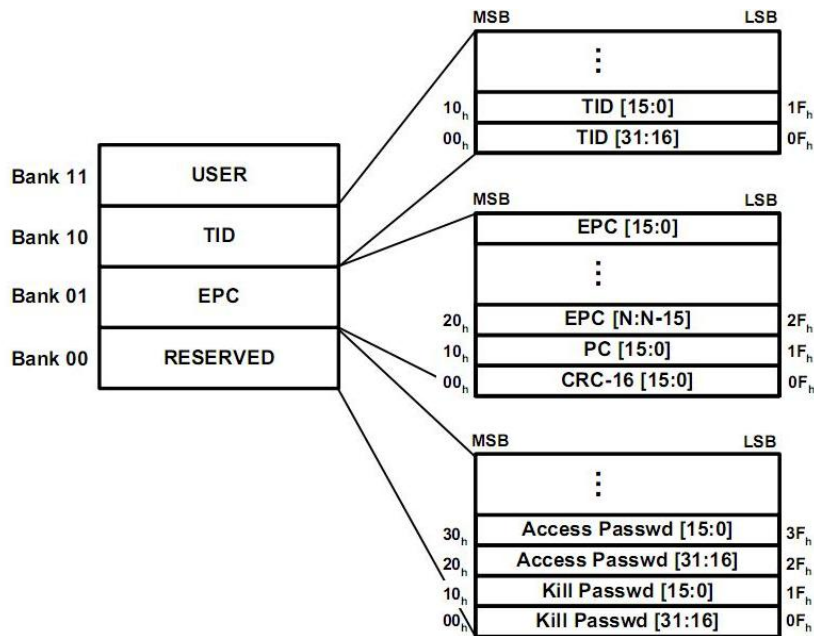
When performing an inventory operation, in addition to obtaining the tag's EPC ID, it is possible to embed additional read instructions to allow reading data in a certain bank. When a read instruction is embedded in an inventory operation, the returned tag data will carry the additional data. The field EmbeddedDataLen of TAGINFO will indicate the number of bytes of additional data. When there is no embedded read instruction or the embedded read instruction fails, this field is 0. This setting only works for Gen2 tag operations.

## 7 Tag Access

Tag access operations include operations such as reading, writing, writing EPC, locking and killing operations. Tag access operations are applied to a single tag. If there are multiple tags in the antenna field, it is not determined which tag the operation ultimately acts on. In this case, if you need to accurately act on one tag, you do it by setting the MTR\_PARAM\_TAG\_FILTER parameter. The tag access operations can only be performed on one antenna at a time. Each tag operation method can specify a timeout, the maximum duration the method will block.

| Function      | Description                                   |
|---------------|---|
| GetTagDataEx  | Read data block of the memory area of the tag |
| WriteTagData  | Write data block to memory area of the tag    |
| WriteTagEpcEx | Write the EPC code of a tag                   |
| LockTag       | Lock the memory areas of a tag                |
| KillTag       | Kill a tag                                    |

A Gen2 tag is divided into four banks which are bank 0, bank 1, bank 2 and bank 3. Bank 0 is also called the reserved bank which contains the access password and the kill password where each password has 32 bits. Bank 1 is called EPC bank, which contains a CRC field (16 bits), PC field (Protocol Control, 16 bits) and EPC field (Electronic Product Code, maximum length is 496 bits and the common length is 96 bits). Bank 2 also known as TID bank, which contains tag- and vendor-specific data (for example, a tag serial number). Bank 3 is called user bank which allows user-specific data storage. Lots of tags do not have bank 3.



## 7.1 GetTagData Function

Read memory area of a bank.

1. `READER_ERR GetTagData(int hReader, int ant, unsigned char bank, unsigned int address, int blkcnt,`
2. `unsigned char* data, unsigned char* accesspasswd, unsigned short timeout, int option, void* pSpec)`

| Parameter    | Description   |
|--------------|---|
| hReader      | Handle of the reader  |
| ant          | Operating antenna   |
| bank         | The bank to read; values 0 – 3 for Gen2 tag, 4 for ISO18000-6b tag                  |
| address      | Starting address in bank [blocks]   |
| blkcnt       | Number of blocks to read  |
| data         | Output parameter with the read data   |
| accesspasswd | Access password if needed (4 bytes); if not needed the parameter can be set to NULL |
| timeout      | Timeout period [ms]   |

## 7.2 GetTagDataEx Function

Read memory area of a bank. Same as GetTagData with two additional parameters to support the QT function of Impinj tags.

1. `READER_ERR GetTagDataEx(int hReader, int ant, unsigned char bank, unsigned int address, int blkcnt,`
2. `unsigned char* data, unsigned char* accesspasswd, unsigned short timeout, int option, void* pSpec)`

| Parameter | Description          |
|-----------|----------------------|
| hReader   | Handle of the reader |
| ant       | Operating antenna    |

|              |   |
|--------------|---|
| bank         | The bank to read; values 0 – 3 for Gen2 tag, 4 for ISO18000-6b tag                  |
| address      | Starting address in bank [blocks]   |
| blkcnt       | Number of blocks to read  |
| data         | Output parameter with the read data   |
| accesspasswd | Access password if needed (4 bytes); if not needed the parameter can be set to NULL |
| timeout      | Timeout period [ms]   |
| option       | Reserved, can be set to 0   |
| pSpec        | Reserved, can be set to NULL  |

**Example:** Read two blocks of data from the second block of bank 0 with antenna 1, access password is 0x12345678

```

1. unsigned char bank = 0;
2. unsigned int addr = 2;
3. unsigned int blks = 2;
4. unsigned char data[4];
5. unsigned char pwd[4];
6. pwd[0] = 0x12;
7. pwd[1] = 0x34;
8. pwd[2] = 0x56;
9. pwd[3] = 0x78;
10. if (GetTagData(hreader, 1, bank, addr, blks, data, pwd, 1000) != MT_OK_ERR)
11. {
12.     printf("GetTagData failed");
13. }
```

### 7.3 WriteTagData Function

Write data into memory area of a bank.

```

1. READER_ERR WriteTagData(int hReader, int ant, unsigned char bank, unsigned int address,
2.     unsigned char* data, int datalen, unsigned char* accesspasswd, unsigned short timeout)
```

**Example:** Write data 0x111122223333 to bank 3 starting at the second block, no access password needed.

```

1. int bank = 3;
2. int addr = 2;
3. unsigned char data[6];
4. data[0] = 0x11;
5. data[1] = 0x11;
6. data[2] = 0x22;
7. data[3] = 0x22;
8. data[4] = 0x33;
9. data[5] = 0x33;
10. if (WriteTagData(hreader, 1, bank, addr, data, 6, NULL, 1000) != MT_OK_ERR)
11. {
12.     printf("WriteTagData failed\n");
13. }
```

### 7.4 WriteTagEpcEx Function

Write the EPC code of a tag. It is also possible to modify the EPC code using the WriteTagData function. The difference between these two functions is that the WriteTagEpcEx also changes the EPC length field of the PC field.

```

1. READER_ERR WriteTagEpcEx(int hReader, int ant, unsigned char *Epc, int epcLen,
2.     unsigned char *accesspasswd, unsigned short timeout);
```

| Parameter | Description   |
|-----------|---|
| hReader   | Handle of the reader  |
| ant       | Operating antenna   |
| Epc       | EPC data to write   |
| epclen    | Length of EPC data [bytes]; must be multiple of 2                                   |
| accesspwd | Access password if needed (4 bytes); if not needed the parameter can be set to NULL |
| timeout   | Timeout period [ms]   |

Example: Write EPC code 0x111122223333111122223333 with access password 0x12345678.

```

1. unsigned char epcdata[12];
2. unsigned char pwd[4];
3. pwd[0] = 0x12;
4. pwd[1] = 0x34;
5. pwd[2] = 0x56;
6. pwd[3] = 0x78;
7. epcdata[0] = 0x11;
8. epcdata[1] = 0x11;
9. epcdata[2] = 0x22;
10. epcdata[3] = 0x22;
11. epcdata[4] = 0x33;
12. epcdata[5] = 0x33;
13. epcdata[6] = 0x11;
14. epcdata[7] = 0x11;
15. epcdata[8] = 0x22;
16. epcdata[9] = 0x22;
17. epcdata[10] = 0x33;
18. epcdata[11] = 0x33;
19. if (WriteTagEpcEx(hreader, 1, epcdata, 12, pwd, 1000) != MT_OK_ERR)
20. {
21.     printf("WriteTagEpcEx failed\n");
22. }
```

## 7.5 LockTag Function

Lock the tag's storage areas. The following objects can be locked: kill password, access password, EPC bank, TID bank, USER bank. The accesspasswd parameter cannot be null. This method could lock multiple objects of one tag at the same time. The lock\_type can be unlock, temporarily lock or permanently lock. This method only supports gen2 tags.

For each Lock\_Obj in lockobjects there must be a corresponding Lock\_Type in locktypes.

```

1. READER_ERR LockTag(int hReader, int ant, unsigned char lockobjects, unsigned short locktypes,
2.     unsigned char* accesspasswd, unsigned short timeout)
```

| Parameter    | Description   |
|--------------|---|
| hReader      | Handle of the reader  |
| ant          | Operating antenna   |
| lockobjects  | The objects to lock. In case of multiple Lock_Obj values, use OR operator |
| locktypes    | The lock type. In case of multiple Lock_Type values, use OR operator      |
| accesspasswd | Access password   |
| timeout      | Timeout period [ms]   |

Example: temporarily lock bank1 and bank3, unlock the access password, the access password is 0x12345678.

```

1. unsigned char pwd[4];
2. pwd[0] = 0x12;
3. pwd[0] = 0x34;
4. pwd[0] = 0x56;
5. pwd[0] = 0x78;
6. if (LockTag(hreader, 1,
7.   LOCK_OBJECT_ACCESS_PASSWD | LOCK_OBJECT_BANK1 | LOCK_OBJECT_BANK3,
8.   ACCESS_PASSWD_UNLOCK | BANK1_LOCK | BANK3_LOCK, pwd, 1000) != MT_OK_ERR)
9. {
10.     printf("LockTag failed\n");
11. }

```

## 7.6 Lock180006BTag Function

Lock a 18000-6b tag.

```
1. READER_ERR Lock180006BTag(int hreader, int ant, int startblk, int blkcnt, unsigned short timeout)
```

| Parameter | Description              |
|-----------|--------------------------|
| hReader   | Handle of the reader     |
| ant       | Operating antenna        |
| startblk  | Starting block to lock   |
| blkcnt    | Number of blocks to lock |
| timeout   | Timeout period [ms]      |

## 7.7 KillTag Function

Kill a tag.

```
1. READER_ERR EXTERN KillTag(int hReader, int ant, unsigned char* killpasswd, unsigned short timeout)
```

| Parameter  | Description          |
|------------|----------------------|
| hReader    | Handle of the reader |
| ant        | Operating antenna    |
| killpasswd | Kill password        |
| timeout    | Timeout period [ms]  |

Example: Kill a tag with kill password 0x43215678.

```

1. unsigned char kpwd[4];
2. kpwd[0] = 0x43;
3. kpwd[1] = 0x21;
4. kpwd[2] = 0x56;
5. kpwd[3] = 0x78;
6. if (KillTag(hReader, 1, kpwd, 1000) != MT_OK_ERR)
7. {
8.     printf("KillTag failed \n");
9. }

```

# 8 Peripheral Functions

Peripheral functions currently only support getting GPI status and setting GPO status.

There are two methods to get the GPI state, one is to get only one GPI pin's state at a time, the other is to get all the reader's GPI pins states at a time.

## 8.1 GetGPI Function

Get the state of a single GPI pin.

```
1. READER_ERR GetGPI(int hReader, int gpoid, int* state)
```

| Parameter | Description                            |
|-----------|--|
| hReader   | Handle of the reader                   |
| gpoid     | GPI id, starting at 1                  |
| state     | Output parameter, state of the GPI pin |

Example: Get the state of GPI pin 1.

```
1. int state;
2. if (GetGPI(hreader, 1, &state) != MT_OK_ERR)
3. {
4.     printf("GetGPI failed \n");
5. }
```

## 8.2 GetGPEx Function

Get the states of all the GPI pins of a reader.

```
1. READER_ERR GetGPEx(int hReader, GpiInfo_ST* gpinfo)
```

| Parameter | Description  |
|-----------|--|
| hReader   | Handle of the reader   |
| gpinfo    | Output parameter, the states of all the GPI pins of a reader |

Example:

```
1. GpiInfo_ST ginfo;
2. if (GetGPEx(hreader, &ginfo) != MT_OK_ERR)
3. {
4.     printf("GetGPEx failed \n");
5. }
```

## 8.3 SetGPO Function

Set the state of a GPO pin.

```
1. READER_ERR SetGPO(int hReader, int gpoid, int state)
```

| Parameter | Description              |
|-----------|--------------------------|
| hReader   | Handle of the reader     |
| gpoid     | GPO id, starting at 1    |
| state     | The state to set; 0 or 1 |

Example: Set the state of GPO pin 1 to value 1.

```
1. if (SetGPO (hreader, 1, 1) != MT_OK_ERR)
2. {
3.     printf("SetGPO failed \n");
4. }
```

## 9 String Functions

Convert byte data into hexadecimal string and vice versa.

## 9.1 Hex2Str Function

```
1. void Hex2Str(unsigned char* buf, int len, char* out)
```

| Parameter | Description                                     |
|-----------|---|
| buf       | Byte data                                       |
| len       | Number of bytes in buf                          |
| out       | Output buffer to store the resulting hex string |

**Example:**

```
1. unsigned char hexbuf[] = {1, 2, 3, 4};
2. char str[9];
3. Hex2Str(hexbuf, 4, str);
4. // Content of str = "01020304";
```

## 9.2 Str2Hex Function

```
1. void Str2Hex(const char* buf, int len, unsigned char* hexbuf);
```

| Parameter | Description  |
|-----------|--|
| buf       | Hexadecimal string                                   |
| len       | Length of the string in buf, must be a multiple of 2 |
| hexbuf    | Output parameter to store the binary byte data       |

**Example:**

```
1. char str[] = "12345678abcd";
2. unsigned char out[100];
3. Str2Hex(str, strlen(str), out);
```

## 10 Error Handling

All the API functions with a return value will return MT\_OK\_ERR on success. If the return value is MT\_IO\_ERR it means, there is something wrong with the network connection or serial port connection. You should call CloseReader and check these connections, then try to call InitReader\_Notype. As for MT\_CMD\_FAILED\_ERR, it just means the failure of the function, it is not a fatal error, and you can do any operation next. As for MT\_CMD\_NO\_TAG\_ERR, it means no tag was found.

The errors starting with MT\_HARDWARE\_ALERT\_ERR\_BY are serious errors which cannot be ignored. There are some improper operations which can cause these exception and hardware damage. These operations include:

- Transmit power through antenna ports without antenna connection,
- use of unqualified antennas,
- high ambient temperature and
- high return loss caused by metal plate in front of the antennas.

It is best to turn off the reader and check the working condition and working environment for these errors.

There is another way to do error troubleshooting. You can call the GetLastDetailError function immediately after an error occurs to get a more detailed error code and an error message represented by a string.

All the functions, except for InitReader\_Notype function, of the current version of SDK are not thread-safe for the same handle of reader. Users should make certain that there is no race condition for the API functions calling or use some synchronization ways of multithread. There is no restriction above for different handles of readers.

## 10.1 GetLastDetailError Function

Get more detailed error information.

```
1. READER_ERR GetLastDetailError(int hReader, int* derrcode, char** errstring)
```

| Parameter | Description                        |
|-----------|------------------------------------|
| hReader   | Handle of the reader               |
| derrcode  | The internal error code of the SDK |
| errstring | Error string                       |

Example:

```
1. int dtcode;
2. char* errstr;
3. if (GetLastDetailError(hreader, &dtcode, &errstr) == MT_OK_ERR))
4.     printf("dtcode:%d, errstr:%s\n", dtcode, errstr);
```

## 10.2 Error Codes

The following is a table of error codes returned by calling the GetLastDetailError function.

| Status Code (Hex) | Status Code (Dec) | Description  |
|-------------------|-------------------|--|
| 0x0000            | 0                 | Success.   |
| 0x0100            | 256               | Sending data length error. The reader firmware version may not match the API version.  |
| 0x0101            | 257               | Command is not available. Make sure the reader supports the command and the power supply of the reader, as it may cause soft resets if the power supply is unstable. |
| 0x0105            | 261               | Parameter value is not available.  |
| 0x010A            | 266               | Baud rate is not available.  |
| 0x010B            | 267               | The current operating frequency band is unavailable.   |
| 0x0200            | 512               | The CRC of the firmware layer is incorrect. Try upgrading the reader firmware again.   |
| 0x0302            | 770               | Flash undefined error, flash write failed.   |
| 0x0400            | 1024              | Tag not found, usually due to insufficient power or filter mismatch.   |
| 0x0402            | 1026              | Protocol is not available. Check the reader firmware and if the API supports the protocol.   |
| 0x0404            | 1028              | When the read after write function is enabled, the write succeeded but the read failed.  |
| 0x040A            | 1034              | General tag error (read/write lock, kill command), can be re-executed.   |
| 0x040B            | 1035              | The read memory area exceeds the limit (only 96 blocks can be read at a time).   |
| 0x040C            | 1036              | Unusable kill password, usually the kill password is not written.  |
| 0x0420            | 1056              | Gen2 protocol error.   |
| 0x0423            | 1059              | The storage area is out of bounds.   |
| 0x0424            | 1060              | The storage area being operated on is locked.  |
| 0x042B            | 1067              | Insufficient energy.   |
| 0x042F            | 1071              | Non-specific error.  |
| 0x0430            | 1072              | Unknown error.   |
| 0x0500            | 1280              | Frequency value not available.   |
| 0x0504            | 1284              | Temperature is too high.   |
| 0x0505            | 1285              | Excessive return loss. Check for metal in the near environment.  |
| 0x500F            | 20495             | The detected temperature exceeds the measurement range.  |
| 0x50FF            | 20735             | Unstable temperature detected.   |
| 0x7F00            | 32512             | The RF hardware circuit failed to start.   |
| 0xAA02            | 43522             | Failed to write OEM register.  |
| 0xAA03            | 43523             | Failed to read OEM register.   |

|        |       |   |
|--------|-------|---|
| 0xAA04 | 43524 | Command execution failed.   |
| 0xAA2A | 43562 | OEM format failed.  |
| 0xAA31 | 43569 | Carrier start or stop failed (no antenna connected).                |
| 0xAA40 | 43584 | Save configuration command failed to write the OEM register.        |
| 0xAA4A | 43594 | Standing wave detection failed.                                     |
| 0xAA4B | 43595 | Reset or power on/off operation failed.                             |
| 0xAA4C | 43596 | No valid setting for multi-tag matching filter data.                |
| 0xAA4D | 43597 | Setting SESSION2/SESSION3 target to A failed.                       |
| 0xAA55 | 43605 | User-defined storage area command, read and write execution failed. |
| 0xAA56 | 43606 | User-defined command configuration or read execution failed.        |
| 0xEE01 | 60929 | The chip did not reach the application layer.                       |
| 0xEE02 | 60930 | Chip firmware startup upgrade failed.                               |
| 0xEE03 | 60931 | Chip firmware upgrade continues to fail.                            |
| 0xEE04 | 60932 | Chip firmware upgrade failed.                                       |
| 0xFF11 | 65297 | Flash initialization failed.  |
| 0xFF12 | 65298 | GPIO initialization failed.   |
| 0xFF13 | 65299 | Timer initialization failed.  |
| 0xFF14 | 65300 | SPI initialization failed.  |
| 0xFF15 | 65301 | Antenna control initialization failed.                              |
| 0xFF16 | 65302 | Frequency band initialization failed.                               |
| 0xFF17 | 65303 | EX10 initialization failed.   |
| 0xFF1F | 65311 | Firmware abnormality.   |
| 0xFFFF | 65535 | Hardware version number error.                                      |
| Non 0  | Non 0 | Command execution failed.   |