



## **Programming Interface (SDK) of TS-HRW Series (13.56 MHz)**

**Version 2.19**



### **GiS Gesellschaft für Informatik und Steuerungstechnik mbH**

Höllochstrasse 1  
D-73252 Lenningen  
Tel. +49 (0)7026 606 0  
Fax +49 (0)7026 606 66  
Email [rfid@gis-net.de](mailto:rfid@gis-net.de)  
Homepage <http://www.gis-net.de/rfid>

**Programming interface (SDK) of the TS-HRW Series****Modification state:**

Docu	SDK Version	Date	Chapter	Name	Info
2.02	2.02	03.12.2014	2.1	Blank	Setup of Gateway address
2.03	2.03	12.10.2015	5.5	Blank	ISO14443B commands added
2.03	2.03	12.10.2015	2.3	Blank	Extended error message reporting
2.03	2.03	12.10.2015	5.6	Blank	Mifare DESFire commands added
2.03	2.03	12.10.2015	6	Blank	Error messages completed
2.03	2.03	25.02.2016		Blank	Device types renewed
2.05	2.05	22.07.2016	5.6	Scherzinger	Mifare DESFire additional explanations
2.06	2.06	04.01.2017	6	Scherzinger	NFC commands added
2.06	2.06	05.01.2017	1.1, 3.8	Blank	Extensions for simplified usage in Reader mode
2.07	2.07	02.05.2017	6	Blank	NFC command for Text added
2.08	2.08	19.06.2017	3.8	Blank	Amendment at errors in Reader mode
2.09	2.09	27.07.2017	6.4	Scherzinger	NFC commands extended or renamed
2.10	2.11	04.12.2018	3.2	Blank	Reader mode parameters extended
2.12	2.12	08.07.2019	2.	Blank	additional functions added
2.13	2.13	06.12.2019		Blank	Use of several devices now different handles
2.13	2.14	08.01.2020		Blank	the commands TSHRW_Beep, TSHRW_SetLED, TSHRW_RawWrite are now executed correctly
2.14	2.14	08.06.2020	2.3	Blank	Command TSHRW_KeepAppActive added to documentation
2.17	2.17	21.09.2022	2.1 2.3	Blank	Commands for LAN Client mode added. Commands to read device name and port added.
2.18	2.18	11.09.2023	2.3, 5.3	Blank	Some commands added.
2.19	2.19	25.10.2023		Blank	Commands for Transparent mode added Naming conventions at all commands refined

**Ownership conditions:**

This document and the software (SDK) are in absolute ownership of GiS, Gesellschaft für Informatik und Steuerungstechnik mbH and all items are to be used confidentially. It is only allowed to use this information and also the SDK together with RFID-Systems of GiS. Without allowance of GiS it is strictly prohibited to make any copies or to give it to third parties neither complete nor in parts.



## Programming interface (SDK) of the TS-HRW Series

# Table of contents

<b>1. Introduction.....</b>	<b>5</b>
1.1. Operation modes .....	6
1.1.1. Reader mode.....	6
1.1.2. Programmer mode.....	6
1.2. Error treatment.....	7
1.3. Definitions .....	7
<b>2. General commands.....</b>	<b>8</b>
2.1. Get attached devices .....	8
2.2. Functions for Ethernet Devices.....	10
2.3. Functions for devices with Bluetooth Interface.....	14
2.4. Common functions for device access .....	16
2.5. Set operation mode.....	20
2.6. Set device parameter.....	21
2.7. Issue general command.....	23
<b>3. Parameter setup for Reader Mode.....</b>	<b>24</b>
3.1. Write Key value.....	24
3.2. Parameter read and write .....	24
3.2.1. Parameter structure.....	25
3.3. Prefix.....	26
3.4. Suffix.....	26
3.5. Termix.....	27
3.6. Postcode.....	27
3.7. Reader Mode Parameter (not for devices with PS2 Interface).....	28
3.7.1. Reader mode parameter structure.....	28
3.8. Data reception in Reader mode .....	29
3.8.1. Cyclic query .....	29
3.8.2. Set up callback function.....	29
3.8.3. Set LED and buzzer .....	30
<b>4. Commands for Transponder Type ISO 15693.....</b>	<b>31</b>
4.1. Get Inventory of tags in antenna field .....	32
4.2. Select transponder.....	33
4.3. Get Transponder info .....	34
4.4. Stay Quiet.....	35
4.5. Reset to ready state.....	35
4.6. Read single block.....	36
4.7. Read multiple blocks.....	36
4.8. Read security state .....	37
4.9. Write single block.....	37
4.10. Lock block.....	38
4.11. Write AFI.....	38
4.12. Lock AFI.....	38
4.13. Write DSFID.....	39
4.14. Lock DSFID .....	39
4.15. Raw request.....	40



<b>5. Commands for Transponder Type MIFARE® (ISO14443A) ..</b>	<b>41</b>
5.1. Application hints.....	41
5.1.1. MIFARE® Ultralight (NFC Type 2).....	41
5.1.2. MIFARE® Classic.....	42
5.1.3. MIFARE® DESFire.....	42
5.2. Function calls MIFARE® common .....	43
5.3. Function calls MIFARE® Classic and Ultralight .....	45
5.4. Function calls ISO14443A-4 .....	48
5.4.1. ISO14443-4 activation .....	48
5.5. Function calls ISO14443B .....	50
5.6. Function calls Mifare DESFire.....	52
5.6.1. Security commands .....	53
5.6.2. Card level commands .....	57
5.6.3. Card Level Commands for Application management .....	60
5.6.4. File level commands .....	63
5.6.5. Commands according to ISO 7816-4 .....	78
<b>6. Commands for NFC (Near Field Communication) .....</b>	<b>80</b>
6.1. General remarks .....	80
6.2. Supported transponder types.....	81
6.2.1. NFC Type 2 .....	81
6.2.2. NFC Type 4 .....	81
6.2.3. NFC Type 6 .....	81
6.2.4. NFC Type 7 .....	81
6.3. Supported types of user data.....	82
6.3.1. Text .....	82
6.3.2. WWW Address .....	82
6.3.3. Telephone.....	83
6.3.4. SMS.....	83
6.3.5. Email .....	84
6.3.6. Contact .....	85
6.4. Function calls NFC.....	86
<b>7. Commands for transparent mode.....</b>	<b>90</b>
<b>8. Error list .....</b>	<b>93</b>
8.1. Common errors .....	93
8.2. Error accessing ISO15693 Transponder.....	94
8.3. Error accessing MIFARE® transponder.....	94
8.4. SDK specific DESFire error codes .....	94
8.5. By DESFire card created native errors .....	95
8.6. DESFire error codes according to ISO7816-4, generated by card.....	95
8.7. Error codes NFC commands .....	96



## **1. Introduction**

The programming interface is built for simple integration of TS-HRWxx devices in arbitrary applications.

The interface to the devices is in GiS Standard Protocol G200 or S002.

The recognition of the device protocol is done automatically when the connection is established. This is done by making the version query with the different protocol types.

The dynamic link library (DLL) provides all commands and maps the entire functionality of the module. You can use the DLL in several program languages.

Two variants of the DLL are given for 32 Bit or 64 Bit applications.

Filename	Import library	Type	Import for C / C++	Import for C#
TS_HRW_SDK.dll	TS_HRW_SDK.lib	32Bit	ts_hrw_import.h	ts_hrw_import.cs
TS_HRW_SDK64.dll	TS_HRW_SDK64.lib	64 Bit	ts_hrw_import.h	ts_hrw_import64.cs

Depending on the device, not all commands are supported. So for example the commands for Mifare® are only supported by TS-HRW38 and TS-HRW32.

In particular, care must be taken that only TS-HW and TS-HRW devices support the programmer mode, and thus the commands from chapters 4 - 6. TS-HR devices are restricted to use in reader mode and respond in principle to commands of the programmer mode with the error code 24.



## **Programming interface (SDK) of the TS-HRW Series**

### **1.1. Operation modes**

Depending on the device, different operation modes are available. It is to be distinguished between the operation modes "Reader mode" and "Programmer mode".

Devices of "R" series work only in "Reader mode", Devices of "W" series only in "Programmer mode", while devices of "RW" series support both modes.

#### **1.1.1. Reader mode**

Automatic reading is called "reader mode". The reader works autonomous and sends the read transponder data through its interface as ASCII characters. Which data in which format is to be sent is adjusted using the "TS-HRW ReaderSetup" application or the functions described in chapter 3. This mode is also often used, if the device has to be used without usage of the SDK, especially if used with serial interface in a third party application designed for serial interface connection or as HID (Human Interface device) in keyboard mode.

To be used with the SDK special access functions are available which do not work with GiS Standard protocol. (Chapter 3.8.)

Take care using other commands in reader mode, the reader will send data automatically if a tag is presented to the device, so this may cause collisions at the reply to standard commands.

This was the reason why special direct access commands had been created to access the buzzer and the LED which do not send any response. (Chapter 3.8.)

#### **1.1.2. Programmer mode**

In programmer mode all access to the transponder is done through commands.

The commands from Chapter 4, 5 and 6 as well as the common commands from chapter 2 might be used.

In this mode it is possible to read and write all the blocks at different transponder types.



## Programming interface (SDK) of the TS-HRW Series

### 1.2. Error treatment

All function give a return value of -1 if an error occurred. To get the error code, call **TSHRW\_GetLastError()**.

You can also get an error message text using **TSHRW\_GetLastErrorMessage()**

Error list is at the end of the document.

### 1.3. Definitions

The following data types are used in the function declarations:

int	32 Bit Integer
BYTE	8 Bit unsigned integer
BYTE *	pointer to array with 8 Bit unsigned integer values
char *	pointer to array with 8 Bit signed integer values, (ANSI Strings in C definition) mostly 0-terminated
WORD	16 Bit unsigned integer
WORD *	pointer to array with 16 Bit unsigned integer values.

The order of data in the fields is always LSB first.



## **2. General commands**

### **int TSHRW\_LibVersion()**

Return: -1 error, >0 Version number with factor 100. E.g. 100 is 1.00

Provides DLL version.

This function does not need TSHRW\_OpenPort.

### **int TSHRW\_IsVirtualComInstalled()**

Return: 1 Virtual Com Driver for TS-HRW38 is installed  
0 Virtual Com Driver for TS-HRW38 is not installed.

### **int TSHRW\_IsDeviceDriverMissing()**

Return: 0: There are no GiS Devices with missing drivers attached  
1: driver for attached HRW34 USB device is missing  
2: driver for attached HRW38 VCOM device is missing  
3: driver for attached HRW34 USB and HRW38 VCOM device is missing

## **2.1. Get attached devices**

### **int TSHRW\_CountAllDevices()**

Return: Number of USB devices

The return value is the number of actually connected USB devices.

Only GiS devices are factored in. A maximum number of 1000 devices will be listed.

### **int TSHRW\_ListAllDeviceNames(char\* NamenListe, int BufferSize)**

NamenListe Name list is a number of null terminated strings.  
End of the list is an empty string.

BufferSize Size of the buffer the user allocates.

Return: <0 error, otherwise the value is the size of actually used buffer

The function **ListAllDeviceNames** provides the USB names (serial numbers) of the USB devices.

Only GiS TS-HR3x and TS-HW3x devices are factored in. Each name consist of a NULL terminated string with at least 8 and am maximum of 18 ASCII characters.

Example: "11360001" or "1460-0001 HID".

Because of the addition "HID" you can distinguish between normal and HID devices which support the USB-HID device interface.





## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_GetAvailablePorts(char\* NamenListe, int BufferSize)**

Name list            Name list is a number of null terminated strings.  
                         End of the list is an empty string.  
Buffer size        Size of the buffer the user allocates.  
Return:            <0 error, or the value is the size of actually used buffer

The function **GetAvailablePorts** provides the names of all available COM-interfaces.

Each name consists of a NULL terminated string. Example: "COM1".

The maximum number of serial interfaces is 255.

The COM Interfaces can exist as real or virtual (through USB realized) Interfaces.

At virtual ports the Name of the virtual port is also given.

Examples:

"\\.\COM1"	Real COM Port 1
"\\.\COM4 [GiS Virtual COM]"	Virtual COM Port 4 created by the "GiS Virtual COM" driver.
"\\.\COM14 [GiS/FTDI Virtual COM]"	Virtual COM Port 14 created by the "GiS/FTDI Virtual COM" driver.

The returned Interface names can be transferred to TSHRW\_OpenPort directly.



## Programming interface (SDK) of the TS-HRW Series

### 2.2. Functions for Ethernet Devices

**int TSHRW\_LanListAllDeviceNames**(char\* NamenListe, int nBufferSize)

NameList      Name list is a number of null terminated strings.

End of the list is an empty string.

BufferSize      Size of the buffer the user allocates.

Return:          <0 error, or the value is the size of actually used buffer

The function **LanListAllDeviceNames** provides the LAN names of the LAN devices.

Only GiS TS-HR3x and TS-HW3x devices are factored in. Each name consists of a NULL terminated string. The buffer space must be big enough to fit for all devices in the network.

Examples:

192.168.0.100-10001

[TS-LAN01]

The device name can be the IP-Address of the device followed by the port number or at DHCP devices, the DHCP name. The DHCP Name is written in [ ]. With this function only devices are found, which are reachable with the actual network settings.

**int TSHRW\_LanConfigListDevices**(char\* NamenListe, int nBufferSize)

NameList      Name list is a number of null terminated strings.

End of the list is an empty string.

BufferSize      Size of the buffer the user allocates.

Return:          <0 error, or the value is the size of actually used buffer

The function **LanConfigListDevices** provides the LAN names of all the LAN devices.

Each name consists of a NULL terminated string. The buffer space must be big enough to fit for all devices in the network.

Examples:

192.168.0.100-10001

[TS-LAN01]169.194.245.01-10001

The device name can be the IP-Address of the device followed by the port number or at DHCP devices, the DHCP name followed by IP-Address and Port number. The DHCP Name is written in [ ]. With this function all devices are found, also if they are not reachable with the actual network settings.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_LanChangeIPAddress(** LPCTSTR OldAddress,  
LPCTSTR NewAddress,  
LPCTSTR IPMask)

OldAddress	Old IP-Address
NewAddress	New IP-Address
IPMask	New IP-Mask

**int TSHRW\_LanChangeIPAddressEx(** LPCSTR OldAddress,  
LPCSTR NewAddress,  
LPCSTR IPMask  
LPCSTR Gateway)

OldAddress	Old IP-Address
NewAddress	New IP-Address
IPMask	New IP-Mask
Gateway	New gateway address

With this function the IP-Address and port number of a device can be changed.

To do this, the device must be reachable with the actual Network settings.

The parameters are all 0 terminated ASCII Strings.

The IP-Addresses are either the IP-Address followed by the port number or the DHCP Name embedded in [ ].

Example:

192.168.0.100-10001  
[TS-LAN01]

In the IP Mask the significant bits are set as bit mask.

Example:

255.255.255.0

At Gateway address the IP Address of the gateway is given. If no gateway shall be set, please use 0.0.0.0 . At gateway no DHCP Name can be used.

With changing the IP-Address the device is restarted. It can last up to 25 Seconds until the device is available in the network.



## Programming interface (SDK) of the TS-HRW Series

```
int TSHRW_LanGetIPAddressEx( LPCSTR Name,  
                             LPSTR Address,  
                             LPSTR IPMask  
                             LPSTR Gateway)
```

Name	Name of device
Address	IP-Address
IPMask	IP-Mask
Gateway	Gateway address

With this function the IP-Address and the port number of a device is read.  
To do this, the device has to be reachable with the actual Network settings.  
The parameters are all 0 terminated ASCII Strings.  
In Address either the DHCP Name or the IP Address is delivered.  
In each passed string enough space to store the address has to be available (at least 30 Byte)

```
int TSHRW_LanGetInfo( LPCSTR strName, char * DHCPName, char * IPAddress,  
                     char * IPMask, char * GatewayAddress);
```

strName	Name of device
DHCPName	DHCPName
Address	IP-Address
IPMask	IP-Mask
GatewayAddress	Gateway address

Using this function the DHCP Name as well as the other IP Address settings are loaded from the device. The parameters are all 0 terminated ASCII Strings.  
In contrast to GetIPAddressEx, both the DHCP name and the IP address are returned here.



## Programming interface (SDK) of the TS-HRW Series

```
int TSHRW_LanGetRemoteIPAndPort( LPCSTR strName, char * pIPAddress,  
                                int MaxIPLen, int * pPortNr);
```

strName	Name of device
pIPAddress	IP-Address
MaxIPLen	Maximum length for pIPAddress
pPortNr	Port number
Rückgabewert	-1: Error, $\geq 0$ Length of IP Address string.

Use this function to read the remote IP Address as well as the remote port number of a device. The remote IP Address is read as 0 terminated string. The remote IP-Address/Port is used to connect automatically to a host which acts as server. Using this the device runs in client mode.

```
int TSHRW_LanWriteRemoteIPAndPort( LPCSTR strName,  
                                   const char * pIPAddress, int PortNr)
```

strName	Name of device
pIPAddress	IP-Address
PortNr	Port number
Rückgabewert	-1: Error, $\geq 0$ Ok

Using this the remote IP is set. The IP-Address is given as 0 terminated string. If an empty string is given, the client mode is deactivated.

```
int TSHRW_LanIsDeviceAvailable(LPCTSTR Address)
```

Address	IP-Address
---------	------------

Return value:	< 0 Error, 0 Device not available > 0 Device available
---------------	--

The IP-Address is either the IP-Address followed by the port number or the DHCP Name embedded in [ ].

Example:

```
192.168.0.100-10001  
[TS-LAN01]
```

With this function can be tested if the device is available in the network.



## Programming interface (SDK) of the TS-HRW Series

### 2.3. Functions for devices with Bluetooth Interface

At devices with Bluetooth interface a two steps connection establishment is needed.  
 First of all the local Bluetooth device has to be found (GIS Bluetooth USB Adapter)  
 The connection to this device is made through TSHRW\_OpenPort(...).  
 After this the external Bluetooth devices can be searched and connected by using the opened port.

**int TSHRW\_BT\_ListAllDeviceNames(char\* Buffer, int BufferSize)**

Buffer                      Name list is a number of null terminated strings.  
                                  End of the list is an empty string.  
 BufferSize                Size of the buffer the user allocates.  
 Return:                    <0 error, or the value is the size of actually used buffer

The function **BT\_ListAllDeviceNames** is used to get the USB-Names (Serial numbers) of all existing GIS USB Bluetooth Adapters. Every name consists of a NULL terminated string.

**int TSHRW\_BT\_Search(int hPort, BYTE \* DeviceList, int BufferSize, int Timeout)**

hPort                      logical device number  
 DeviceList                Description of DeviceList follows below.  
 BufferSize                Length of Device list  
 Timeout                  Duration for search in milliseconds usually used 120000.  
 Return:                    <0 error, or the value is the size of actually used buffer

Only as many devices are listed as space in device list is available. If more devices are found, these are ignored.

An entry in the device list is as follows:

Start	length	meaning
0	6	Bluetooth address of the remote device
6	32	Local Name of the remote device for ex.: "PN1450#0001" The Name is fills with Null bytes. The name has up to 31 Bytes, the 32nd Byte is the terminator (Null byte)

The device list is a list of such entries.  
 The returned length is always multiple of 38.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_BT\_Connect**(int hPort, BYTE \* Device,int Timeout)

hPort                      logical device number

Device                     Bluetooth Address of the remote device

Timeout                    max. Duration for the connection setup in milliseconds, normally 3000.

Return:                     < 0 Error,  
                              0 connection established.

Using this function the connection to the remote device is established.

Once the connection is established the normal functions for the communication are used as if the device would be locally connected.

**int TSHRW\_BT\_CheckConnection**(int hPort)

hPort                      logical device number

Return:                     < 0 Error,  
                              0 Connection is released  
                              1 Connection exists

Checks the connection state and returns if the connection still exists or if the link is released

**int TSHRW\_BT\_Disconnect**(int hPort)

hPort                      logical device number

Return:                     < 0 Error,  
                              0 Connection disconnected.

Disconnect the remote device. Because the local Bluetooth stick is already open a connection to another remote device can be established now also.



## Programming interface (SDK) of the TS-HRW Series

### 2.4. Common functions for device access

**Attention:** The function `TSHRW_OpenPort()` has to be called before using any other command. The returned handle is necessary for all read and write commands. If there is an interface open error, the return value is a negative error number. See error list.

**int TSHRW\_OpenPort(LPCTSTR strInterfaceName, int Baudrate, int Timeout)**

Opens the interface for RS232 and USB devices.

strInterfaceName	Name of the interface. e.g. "COM1" or serial number of USB devices. For <b>USB</b> devices you have to put in the device name. At <b>USB HID</b> devices you have to put in the device name followed by "HID" You can get the device name with the function <b>TSHRW_ListAllDeviceNames.</b> E.g. "10610011" or "1317-0001 HID" At <b>LAN</b> devices you have to give the address of the device. The available LAN devices can be found with function <b>TSHRW_LanListAllDeviceNames.</b>
Baudrate	Set the baud rate. (2400, 4800, 9600, 19200, 38400, 57600, 115200 and 230400 supported). Attention: Not all devices support every baud rate. Please read device specification for valid baud rates. <b>Default setting for RS232 readers is 19200.</b> <b>USB and LAN devices do ignore the baud rate.</b>
Timeout	Time span after which communication shall break off. (in milliseconds)
Return:	<0: error, >0: <b>PortHandle</b> logical device number

Optionally the device address can be given at the interface name. For example COM1:4 opens at COM 1 the device with address 4. This is necessary if the device address is not 1, because TSHRW\_OpenPort makes a connection to the device and this is only accepted if the device answers.

The function automatically recognizes the under laying protocol of the device (normally GiS LowLevel Protocol G200, at MultiX devices MultiX Modbus Protocol, at S002 devices GiS LowLevel Protocol S002, Medio P200u uses S004 Protocol)  
Medio P200u is always using virtual com mode and has to be opened with Baud rate 115200.

**Attention:** You generally have to call `TSHRW_ClosePort()` at the end of the application, because this function closes the interface and frees all resources.

**int TSHRW\_ClosePort(int hPort)**

hPort	logical device number
Return:	-1 error, 0 OK





## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_GetPortName**(int hPort, char \* pName, int MaxLen)

hPort                      logical device number  
pName                      pointer to memory for device name  
MaxLen                    Length of memory area for device name  
Return value:            -1 Fehler, > 0 Länge des Schnittstellennamens

Returns the name of the port in pName as given at OpenPort.

**int TSHRW\_KeepAppActive** (int hPort, int bActive)

hPort                      Access handle  
bActive                    0: Application is inactive while a SDK function is running  
                              1: Application keeps active while a SDK function is running

This is used to control the behavior of a calling application. It is helpful if functions are called in very short interval. After TSHRW\_OpenPort it is set to bActive = 0. So while a function is called the application is blocked. After calling KeepAppActive with bActive = 1 the application stays active also while inside a function call. It is to be ensured by the user that no more functions for this port handle are called while a function is running.

**int TSHRW\_IsUSB**(int hPort)

hPort                      logical device number  
Return:                    -1 error, 0 = **NO** USB Port, 1 = USB Port

**int TSHRW\_IsProgrammer**(int hPort)

hPort                      logical device number  
Return:                    -1 error  
                              0 = Programmer Mode is **not** supported  
                              1 = Programmer Mode is supported

This function returns if the programmer mode is supported by the device and the corresponding commands are allowed, not if the programmer mode is activated.

**int TSHRW\_IsReader**(int hPort)

hPort                      logical device number  
Return:                    -1 error  
                              0 = Reader Mode is **not** supported  
                              1 = Reader Mode is supported

This function returns if the reader mode is supported by the device, not if the reader mode is activated.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_SetReaderAdresse**(int hPort, int Address)

hPort                      logical device number

Address                    Address of module, default value after OpenPort is 1 or the address given at OpenPort. The address is only used in RS485 connections if more than one device is attached.

Return:                    < 0 error, 0 OK

**int TSHRW\_DeviceVersion**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number

pBuffer                    Pointer to version data

BufLen                    Length of version data (4 Byte)

Return:                    -1 Error

Device type and Firmware version in ASCII Format.

Byte 1 – 3                device type

Byte 4                    Firmware version (0 – 9, A – Z, a – z)

### The Dll supports the following device versions:

Device numbers of TS-HR3x and TS-HW3x RFID devices

The device number serves to classify the device type.

- 121 = (TS-HW34)            only programmer
- 122 = (TS-HR34 PS2)      only reader for PS2 Interface (configurable using RS232)
- 123 = (TS-HRW34 PS2)    reader for PS2 Interface and Programmer with RS232 Interface
- 124 = (TS-HR34)            only reader
- 125 = (TS-HRW34)          programmer and reader
- 135 = (TS-HRW90)          programmer and reader with RS485 Interface
- 141 = (TS-HW36)            only programmer with HID or LAN Interface
- 144 = (TS-HR36)            only reader with HID or LAN Interface
- 145 = (TS-HRW36)          programmer and reader with HID or LAN Interface
- 161 = (TS-HW38)            only programmer with HID or LAN Interface
- 164 = (TS-HR38)            only reader with HID or LAN Interface
- 165 = (TS-HRW38)          programmer and reader with HID or LAN Interface
- 171 = (TS-HW34 S002)      programmer with S002 protocol
- 195 = (TS-HRW32)          programmer and reader
- 117 = (TS Medio P200u)    programmer with S004 Protocol
- 405 = (TS-HRW390)          programmer and reader
- 411 = (TS-HW421)            only programmer with HID or LAN Interface
- 414 = (TS-HR421)            only reader with HID or LAN Interface
- 415 = (TS-HRW421)          programmer and reader with HID or LAN Interface
- 421 = (TS-HW420)            programmer with S002 protocol
- 431 = (TS-HW480)            programmer with S002 protocol
- 451 = (TS-HW380)            only programmer with HID or LAN Interface
- 454 = (TS-HR380)            only reader with HID or LAN Interface
- 455 = (TS-HRW380)          programmer and reader with HID or LAN Interface

The devices TS-HW34, HR34 and HRW34 are available in RS232, USB or Bluetooth version.

The devices TS-HW36, HR36 and HRW36 are available in USB-HID or LAN version.

The devices TS-HW38, HR38 and HRW38 are available in RS232, USB-HID, USB-VCOM or LAN version.

The devices TS-HRW32 are available in RS232, USB-HID or USB-VCOM version.

The device series TS-HRW38, TS-HRW32, TS-HRW390 and TS-HRW380 supports additional to the transponders in ISO15693 Standard also MIFARE® classic, Ultralight® and DESFire®

```
int TSHRW_GetDeviceName(int hPort, char * pDevName, int NameLen)
```

Returns the device name of the opened device.

Set baud rate (only for RS232 devices; USB devices provide an error).

**int TSHRW\_GetLastError(int hPort)**[illegible]

Provides an error message string after execution of a command.



## Programming interface (SDK) of the TS-HRW Series

### 2.5. Set operation mode

**int TSHRW\_SetReaderMode(int hPort, int mode)**

hPort	logical device number
Mode	1 sets the device to reader mode. 0 sets the device to programmer mode. 2 set device to power up mode 80H set power up mode as programmer mode 81H set power up mode as read mode
Return:	-1 error, 0 OK

For devices which can operate in reader or programmer mode, this command is used to activate the reader resp. programmer mode.

In reader mode the device send the data of the transponder without protocol header as set up in the parameter settings for the reader mode. This interferes communication in programmer mode and so the reader mode has to be deactivated during programmer usage.

Also especially at HID (Keyboard emulation) devices it can be very uncomfortable if the keyboard emulation is activated at the time a transponder is attached.

**int TSHRW\_SetRF(int hPort,int OnOff)**

hPort	logical device number
OnOff	0 = turn antenna field off, 1 = turn antenna field on
Return:	-1 error, 0 OK

Turn antenna field on or off



## Programming interface (SDK) of the TS-HRW Series

### 2.6. Set device parameter

**int TSHRW\_SetIO**(int hPort, int Maske, int Daten)

hPort                      logical device number

Maske                      Mask values for outputs to set

Daten                      Values for outputs

all bits are used, which are set in the mask.

Return:                    -1 error, 0 OK

With this outputs of the device are set. Depending on the device, different outputs are available. After power on at the device the LED's are set automatically. After using this command only those outputs are set automatically which are not included in the mask.

Definition of bits:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
led yellow	Led green	Led red	Buzzer	Out 3	Out 2	Out 1	Out 0

If the device is equipped with a relay output, this is connected through Out 0.

Example: Mask: C0H Data: 40H sets the green led and turns the yellow led off, the red led and all other outputs are kept unattended and are set through the reader regarding to the operation mode.

**int TSHRW\_ReadIO**(int hPort)

hPort                      logical device number

Return:                    -1 error, Value of the inputs read

It depends on the device version which inputs are available.

If the device is equipped with a sensor input, this is available in the lowest bit.

**int TSHRW\_SetDefault**(int hPort)

hPort                      logical device number

Return:                    -1 Error, 0 OK

Activate Factory default settings. This command is not supported at all devices.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_SetConfig**(int hPort, BYTE \* pConfig, int ConfigLen)

hPort                      logical device number  
pConfig                   pointer to configuration  
ConfigLen                Length of buffer  
Return:                    -1 Error, 0 OK

Write the configuration to the device. The configuration is dependent of the device.  
This command is not supported at all devices.

**int TSHRW\_GetConfig**(int hPort, BYTE \* pConfig, int ConfigLen)

**int TSHRW\_GetConfigEx**(int hPort, int Select, BYTE \* pConfig, int ConfigLen)

hPort                      logical device number  
Select                     Bitcoded Parts to be selected for Answer  
pConfig                   pointer to read buffer  
ConfigLen                Length of buffer  
Return:                    -1 Error, length of actually read data.

Read the configuration from the device. The configuration is dependent of the device.  
This command is not supported at all devices.

**int TSHRW\_ReadSerialNumber**(int hPort, BYTE \* pSerial, int Buflen)

hPort                      logical device number  
pSerial                    pointer to read buffer  
Buflen                     Length of buffer  
Return:                    -1 Error, length of actually read data.

Read serial number of the device. This command is only supported at new devices.  
Serial number is reported as 4 Byte binary Number with LSB first.

**int TSHRW\_SetCommunicationMode**(int hPort, int Mode)

hPort                      logical device number  
Mode                       Communication mode  
                              1: USB HID Interface  
                              2: USB Virtual COM Interface  
Return:                    -1 Error, 0 OK

Switch a HID device to Virtual COM mode or vice versa. After this the interface has to be closed and reopened in the new mode.



## Programming interface (SDK) of the TS-HRW Series

### 2.7. Issue general command

Some devices accept additional commandos, which can be accessed by this general command function. The access is always done with the given data protocol.

**int TSHRW\_Transfer**(int hPort, BYTE \* pSendBuf, int SendBufLen,  
BYTE \* pRecvBuf, int RecvBufLen)

hPort	logical device number
pSendBuf	Pointer to data to send
SendBufLen	length of data to send
pRecvBuf	pointer to data to receive
RecvBufLen	max. length of data to receive
Rückgabewert:	-1 error, >= 0 length of received data in pRecvBuf

If direct communication to the device is needed, for example in reader mode without any data protocol, this can be done with the following functions:

**int TSHRW\_RawWrite**(int hPort, BYTE \* pBuffer, int BufLen)

hPort	logical device number
pBuffer	Pointer to write buffer
BufLen	Length of buffer
Return:	-1 Error, 0 OK

This function writes arbitrary data to the interface.  
No protocol implied.

**int TSHRW\_RawRead**(int hPort, BYTE \* pBuffer, int BufLen)

hPort	logical device number
pBuffer	Pointer to read buffer
BufLen	Length of buffer
Return:	-1 Error, length of actually read data.

This function reads arbitrary data to the interface.  
No protocol implied.



### **3. Parameter setup for Reader Mode**

These commands are used to set up the device in reader mode

#### **3.1. Write Key value**

This function is only used at MIFARE® transponders and is only supported at readers which support MIFARE®.

**int TSHRW\_WriteMifareKey**(int hPort, int KeyType, BYTE \* pKey, int KeyLen)

hPort                      logical device number  
KeyType                    Key type  
pKey                        pointer to key  
                              The key has to have 6 Bytes.  
KeyLen                     length of key (6 Byte)  
Return:                    -1 Error, 0 OK

#### **3.2. Parameter read and write**

If multiple parameter data structures are given, the transponder types are polled alternating.

If only one data structure is given, this does not have to be filled completely, if multiple data structures are given, these have to be set completely with 20 Byte per data structure.

Attention: only device type TS-HR38 and TS-HR32 supports multiple data structures. All former devices only one data structure with maybe less parameters is supported.

**int TSHRW\_ReadParam**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number  
pBuffer                    Pointer to read buffer.  
                              See: **3.2.1 Parameter structure**.  
BufLen                     Length of read buffer (20 Byte per data structure)  
Return:                    -1 error, length of actually read data

**int TSHRW\_WriteParam**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number  
pBuffer                    Pointer to write buffer.  
                              See: **3.2.1 Parameter structure**.  
BufLen                     Length of write buffer (20 Byte per data structure)  
Return:                    -1 error, 0 OK





## Programming interface (SDK) of the TS-HRW Series

### 3.2.1. Parameter structure

The parameter for reading and writing are passed always in the same order.

Pos.	length	Name	Description
0	1	Wait	Wait time between 2 characters: (0 – 127) Only PS/2. Wait time = $N/2$ in milliseconds. Default = 10
1	1	TTyp	Transponder type: Bit 0 – 5    0 = UID ISO15693    1 = Register ISO15693 2 = UID Mifare        3 = Register Mifare 4 = UID ICode1       5 = Register ICode1 6 = UID DESFire      7 = File content DESFire Bit 6:        0 = normal            1 = Bitwise mirrored Bit 7         0 = LSB first        1 = MSB first (Byte wise turned)
2	5	Register	The entry consists always of 4 Register data bytes and the end identification 0xffH (5th Byte). This entry is only valid if a "Register" type is selected in TTyp (1 or 3). Byte 5 is always the end identification 0xffH. e.g. 0x00,0x01,0x0f,0x05,0xff. If TTyp 7 File content DESFire is used, the first 3 byte define the Application ID and the 4 <sup>th</sup> Byte defines the FileID.
7	1	DTyp	Data type: 0 = Hexadecimal,                    3 = Decimal without leading zero, 1 = decimal,                         4 = Hexadecimal with lower case letters 2 = ASCII,                            5 = ASCII, 00H will be suppressed, do not fill with SPACE
8	1	Character	Number of characters. (1–32). Defines how much characters are transferred per block.
9	1	Frequenz	Frequency. Only for PS/2. Here we transfer a special interval time span. You can calculate the corresponding frequency with the formula: $f[\text{MHz}] = \frac{1}{(64[\mu\text{s}] + 8[\mu\text{s}] * N)}$ The value 5 is correlating with frequency 10 kHz. (Default) The value 17 is correlating with frequency 5 kHz. (Values are rounded).
10	1	Timeout	You can calculate the time out with the following formula: (Only for PS/2): $T[\text{ms}] = 30[\text{ms}] + 30[\text{ms}] * \text{Timeout}$ . Default value is 5, which means 180 ms. After this time is gone, the same transponder can be read again.
11	1	ValidBytes	(1-16) Number of bytes used in UID or data blocks. With this, the upper bits of the UID can be masked out. Default value is 5
12	1	ValidFrom	(1-16) Start byte from which the data is transferred. This is only available at TS-HR38 with Version 1.07 or higher and TS-HR32.
13	1	TypKenn1	Detection characters for the transponder type are transmitted before the data, 1 ASCII character, at 0 nothing is transferred.
14	1	TypKenn2	Detection characters for the transponder type is transmitted according to the data, 1 ASCII character, at 0 nothing is transferred.
15	1	DesfireMode	0:plain, 1:Authenticated, 2:with MAC, 3:fully encrypted
16	4	-	Reserve, default value 0



## Programming interface (SDK) of the TS-HRW Series

### 3.3. Prefix

Reading and writing of the Prefix setting

The prefix consists of 31 characters maximum. End marker is 0xFF Hex.

**int TSHRW\_WritePrefix**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number

pBuffer                   Pointer to write buffer.

BufLen                    Length of write buffer

Return:                    -1 error, 0 OK

**int TSHRW\_ReadPrefix**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number

pBuffer                   Pointer to read buffer.

BufLen                    Length of read buffer

Return:                    -1 error, length of actually read data

### 3.4. Suffix

Reading and writing of the Suffix setting

The suffix consists of 31 characters maximum. End marker is 0xFF Hex.

**int TSHRW\_WriteSuffix**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number

pBuffer                   Pointer to write buffer.

BufLen                    Length of write buffer

Return:                    -1 error, 0 OK

**int TSHRW\_ReadSuffix**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number

pBuffer                   Pointer to read buffer.

BufLen                    Length of read buffer

Return:                    -1 error, length of actually read data



## Programming interface (SDK) of the TS-HRW Series

### 3.5. Termix

Reading and writing of the Termix setting

The termix consists of 31 characters maximum. End marker is 0xFF Hex.

**int TSHRW\_WriteTermix**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number  
pBuffer                    Pointer to write buffer.  
BufLen                     Length of write buffer  
Return:                    -1 error, 0 OK

**int TSHRW\_ReadTermix**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number  
pBuffer                    Pointer to read buffer.  
BufLen                     Length of read buffer  
Return:                    -1 error, length of actually read data

### 3.6. Postcode

Reading and writing of the Postcode setting

The postcode consists of 31 characters maximum. End marker is 0xFF Hex.

**int TSHRW\_WritePostcode**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number  
pBuffer                    Pointer to write buffer.  
BufLen                     Length of write buffer  
Return:                    -1 error, 0 OK

**int TSHRW\_ReadPostcode**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number  
pBuffer                    Pointer to read buffer.  
BufLen                     Length of read buffer  
Return:                    -1 error, length of actually read data



## Programming interface (SDK) of the TS-HRW Series

### 3.7. Reader Mode Parameter (not for devices with PS2 Interface)

Read and write the reader mode parameters.

**int TSHRW\_WriteReadModeParam**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number

pBuffer                    Pointer to write buffer.

See: **3.7.1 Read mode parameter structure**

BufLen                    Length of write buffer

Return:                    -1 error, 0 OK

**int TSHRW\_ReadReadModeParam**(int hPort, BYTE \* pBuffer, int BufLen)

hPort                      logical device number

pBuffer                    Pointer to read buffer.

See: **3.7.1 Read mode parameter structure**

BufLen                    Length of read buffer

Return:                    -1 error, length of actually read data

#### 3.7.1. Reader mode parameter structure

The parameter for reading and writing are passed always in the same order.

Data 1	Data 2	Data 3	Data 4
Mode	Cycle time	Prompt	Timeout

**Mode:**                    Operation mode

0: Send data when transponder goes in field and when transponder leaves field.

1: Send data when prompt is sent.

The prompt is defined in **Prompt**.

The device sends data if the prompt is sent to the device.

3: Send data cyclic

**Cycle time:**            The time is given in 1/10 seconds. The default value is 10 which means 1 second.

The cycle time is valid in mode 3 and 5.

**Prompt:**                    Character for prompting which is needed only in Mode 1. Default value is '?' (3fH)

**Timeout**                    The time is given in 1/10 seconds. The default value is 20 which means 2 seconds.

The Timeout is valid in mode 0 and 5. It defines how long a transponder has to be out of field to be recognized again as new transponder.



## Programming interface (SDK) of the TS-HRW Series

### 3.8. Data reception in Reader mode

The data received in reader mode can be accepted in different ways.

#### 3.8.1. Cyclic query

The received items can be loaded using multiple calls of the TSHRW\_RawRead command.

Then the data set has to be connected out of all the received parts. The check for end of data has to be done by the calling instance.

#### 3.8.2. Set up callback function

Using an application specific call back function, the data set can be received and evaluated by the SDK. The complete data set is given to the callback function as one string of data. It is essentially needed, that transmission ends with a unique character. Normally CR = 0DH is used for this.

**int TSHRW\_StartAutoRead**(int hPort, int TermChar, AutoReadCallback pAutoReadProc)  
hPort                      logical device number  
TermChar                  Terminating character for the transmission. This character triggers the call of the callback function.  
pAutoReadProc            pointer to callback function  
Return value:             -1 Error, 0: OK

Definition of callback function:

'C'

```
typedef int(__stdcall* AutoReadCallback)(char * pData, int Len);
```

'C#'

```
delegate int AutoReadCallback( [MarshalAsAttribute(UnmanagedType.LPStr)] string pData, int Len);
```

'VB'

```
Delegate Function AutoReadCallback ( <MarshalAs(UnmanagedType.LPStr)> Arr As String, ByVal Len As Integer) As Integer
```

The usage of the callback function is described in the "Readermodus" sample application.

The callback function is called with an empty string and Len=0, if the used device is no longer available, for example, if the USB Device is removed during work.

**int TSHRW\_StopAutoRead**(int hPort)  
hPort                      logical device number  
Return value:             -1 Error, 0: OK  
Terminates usage of the callback function.



## Programming interface (SDK) of the TS-HRW Series

### 3.8.3. Set LED and buzzer

These functions are only supported at TS-HR38/HRW38 Version 1.23 as well as TS\_HR32/HRW32 Version 1.02 and TS-HR39/TS-HRW39 Version 1.02 or higher version at these devices.

**int TSHRW\_SetLED(int hPort, int red, int green, int yellow)**

hPort                      logical device number

red                        -1: do not change red LED

                             0: turn off red LED

                             1: turn on red LED

                             2: let the device control the red LED

green                    -1: do not change green LED

                             0: turn off green LED

                             1: turn on green LED

                             2: let the device control the green LED

yellow                   -1: do not change yellow LED

                             0: turn off yellow LED

                             1: turn on yellow LED

                             2: let the device control the yellow LED

Return value:            -1 Error, 0: OK

**int TSHRW\_Beep(int hPort)**

hPort                    logical device number

Return value:            -1 Error, 0: OK

Activate the buzzer for 200 msec. This of course works only if the optional buzzer is available in the connected device.



## Programming interface (SDK) of the TS-HRW Series

### ***4. Commands for Transponder Type ISO 15693***

Basically at all commands for ISO15693 the RequestFlag is given. Please take care of the ISO15693 spec for definition of this flag.

At the following functions the needed bits of the RequestFlag are automatically set dependent to the transponder type. If the command is called with UID, the transponder type is included in the UID. If used without UID, the transponder type can be added to the request flag.

Transponder type flag are defined as follows:

MYD1	=	0x0100
ICODE2	=	0x0200
TAGIT2	=	0x0400
ELECTRO MARIN	=	0x0800
STM	=	0x1000
FUJITSU	=	0x2000
KSW	=	0x4000
AVOID Automatic	=	0x8000

The setting of request flag can be avoided by adding IGNORE\_AUTOMATIC\_FLAGS = 0x8000 to the given value. But with this it is up to the user to set the RequestFlag correctly.



## Programming interface (SDK) of the TS-HRW Series

### 4.1. Get Inventory of tags in antenna field

**int TSHRW\_ISO\_Inventory**(int hPort, BYTE \* pBuffer, int BufLen, int RequestFlag)

**int TSHRW\_ISO\_InventoryAFI**(int hPort, BYTE \* pBuffer, int BufLen,  
int RequestFlag, int AFI)

Read inventory information as described in ISO 15693.

This command has to be supported by all ISO transponders.

Inventory provides the unified identifier (UID)

hPort                      logical device number

pBuffer                    Pointer to inventory data

BufLen                    Length of data (x times 11 Bytes) (x = Number of recognized RFID's)  
Maximum RFID number = 16.

RequestFlag              Request Flag, see ISO 15693

AFI                         AFI Value, see ISO 15693

Return                    < 0 error, length of read data

The TSHRW\_ISO\_InventoryAFI function is used to get inventory information of transponders with specific AFI setting.

Each transponder provides 11Byte with the structure:

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
<b>Byte0</b>	Timeslot number				ISO 15693 - Error code			

Error code = 0, no error

Byte1: Response Flags see ISO 15693.

Byte2: DSFID see ISO 15693.

Byte3 - Byte10: UID





## Programming interface (SDK) of the TS-HRW Series

### 4.2. Select transponder

**int TSHRW\_ISO\_Select**(int hPort, BYTE \* pUID,int UIDLen,int nRequestFlag)

hPort                logical device number

pUID                pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored.

UIDLen              length of blocks (must be 8). If Length != 8 parameter pUID is ignored.

RequestFlag        Request Flag see ISO 15693

Return               < 0 error, 0 = OK

Select calls the transponder with this UID. If the UID matches the transponder switches in selected state and you get an answer. If UID doesn't match there is no answer.



## Programming interface (SDK) of the TS-HRW Series

### 4.3. Get Transponder info

**int TSHRW\_ISO\_GetSystemInfo**(int hPort, BYTE \* pBuffer, int BufLen, BYTE \* pUID, int UIDLen, int nRequestFlag)

hPort	logical device number
pBuffer	pointer to system information see ISO norm 15693
BufLen	length of system information (variable)
pUID	pointer to UID. Request flag is set automatically. If pointer is NULL, the UID will be ignored.
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
RequestFlag	Request Flag see ISO 15693. At some transponders it might be necessary to set the additional Protocol extension flag (ISO15693_PROTOKOLL_EXTENSION = 08H).
Return	< 0 error, length of read data

**int TSHRW\_ISO\_GetTagInfo**(int hPort, BYTE \* pUID, int UIDLen, int\* pBlockAnzahl, int\* pBytesProBlock, int\* pMfgCode, int nRequestFlag)

hPort	logical device number
pUID	pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored.
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
pBlockAnzahl	pointer to block number. If pointer = NULL, Info is ignored.
pBytesProBlock	pointer to Bytes Pro Block If pointer = NULL, Info is ignored.
pMfgCode	pointer to Manufacturer code. If pointer = NULL, Info is ignored.
RequestFlag	Request Flag see ISO Norm 15693
Result	-1 Error, 0 = OK

GetTagInfo() can provide specific information about the RFID. This is no direct ISO command. It is made by GiS to help the clients that are not familiar with ISO norm 15693. You only have to know the UID (you get UID from TSHRW\_ISO\_Inventory()) and you get block number, bytes per block and manufacturer code. It uses the GetSystemInfo function internally if needed.



## Programming interface (SDK) of the TS-HRW Series

### 4.4. Stay Quiet

**int TSHRW\_ISO\_StayQuiet**(int hPort, BYTE \* pUID, int UIDLen, int nRequestFlag)

hPort                logical device number

pUID                pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored.

UIDLen             length of blocks (must be 8). If Length != 8 parameter pUID is ignored.

RequestFlag        Request Flag see ISO Norm 15693

Return              < 0 error, 0 = OK

“Stay quiet” causes the transponder to stay in low energy level. “Stay quiet” only makes sense with UID. Because exactly this transponder is meant. After “Stay quiet” “Inventory” ignores the transponder. The state will be suspended with “**Reset to Ready**”. Or with a command that uses UID, like “**ReadSingleBlock**” with UID. The UID parameter resets the stay quiet state.

### 4.5. Reset to ready state

**int TSHRW\_ISO\_ResetToReady**(int hPort, BYTE \* pUID, int UIDLen, int nRequestFlag)

hPort                logical device number

pUID                pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored.

UIDLen             length of blocks (must be 8). If Length != 8 parameter pUID is ignored.

RequestFlag        Request Flag see ISO 15693

Return              < 0 error, 0 = OK

Suspends the „Quiet“- State.



## Programming interface (SDK) of the TS-HRW Series

### 4.6. Read single block

**int TSHRW\_ISO\_ReadSingleBlock**(int hPort, int BlockNr, BYTE \* pBuffer, int BufLen, BYTE \* pUID, int UIDLen, int nRequestFlag)

hPort	logical device number
BlockNr	Block number
pBuffer	pointer to data
BufLen	length of data (variable)
pUID	pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored.
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
RequestFlag	Request Flag see ISO Norm 15693 Using transponders of STM with more than 256 blocks, you have to give the protocol extension flag (ISO15693_PROTOKOLL_EXTENSION = 08H) additionally. This is needed at access to all block of such a transponder.
Return	< 0 error, length of read data

### 4.7. Read multiple blocks

**int TSHRW\_ISO\_ReadMultipleBlocks**(int hPort, int nFirstBlock, int nNumberOfBlocks, BYTE \* pBuffer, int BufLen, BYTE \* pUID, int UIDLen, int nRequestFlag)

hPort	logical device number
nFirstBlock	Number of first block (0-254)
nNumberOfBlocks	Number of blocks to be read.
pBuffer	pointer to data
BufLen	length of data (variable)
pUID	pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored. You can get all UID's with TSHRW_Inventory().
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
RequestFlag	Request Flag see ISO Norm 15693 Using transponders of STM with more than 256 blocks, you have to give the protocol extension flag (ISO15693_PROTOKOLL_EXTENSION = 08H) additionally. This is needed at access to all block of such a transponder.
Return	< 0 error, length of read data



## Programming interface (SDK) of the TS-HRW Series

### 4.8. Read security state

```
int TSHRW_ISO_GetMultipleBlockSecurityStatus(int hPort, int FirstBlock,
                                             int NumberOfBlocks,
                                             BYTE * pReceiveBuffer, int BufLen,
                                             BYTE * pUID, int UIDLen,
                                             int nRequestFlag)
```

hPort	logical device number
FirstBlock	Number of first block
NumberOfBlocks	Number of blocks to read
pReceiveBuffer	pointer to receive buffer
BufLen	length of receive buffer
pUID	pointer to UID. Request flag is set automatically if UID is given. If pointer is NULL, the UID is ignored. UIDs are found using TSHRW_ISO_Inventory().
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
RequestFlag	Request Flag see ISO 15693 Using transponders of STM with more than 256 blocks, you have to give the protocol extension flag (ISO15693_PROTOKOLL_EXTENSION = 08H) additionally. This is needed at access to all block of such a transponder.
Return	< 0 error, length of read data

Provides block security of multiple blocks

### 4.9. Write single block

```
int TSHRW_ISO_WriteSingleBlock(int hPort, int BlockNr, BYTE * pBuffer, int BufLen,
                                BYTE * pUID, int UIDLen, int nRequestFlag)
```

hPort	logical device number
BlockNr	Block number
pBuffer	pointer to transponder data
BufLen	length of transponder data (variable)
pUID	pointer to UID. Request flag is set automatically if UID is given. If pointer is NULL, the UID is ignored.
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
RequestFlag	Request Flag see ISO Norm 15693 Using transponders of STM with more than 256 blocks, you have to give the protocol extension flag (ISO15693_PROTOKOLL_EXTENSION = 08H) additionally. This is needed at access to all block of such a transponder.
Return	< 0 error, 0 = OK



## Programming interface (SDK) of the TS-HRW Series

### 4.10. Lock block

**int TSHRW\_ISO\_LockBlock**(int hPort, int nBlockNr, BYTE \* pUID, int UIDLen,  
int nRequestFlag)

hPort	logical device number
BlockNr	Block number (0-254)
pUID	pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored. You can get all UID's with TSHRW_Inventory().
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
RequestFlag	Request Flag see ISO Norm 15693
Return	< 0 error, 0 = OK

Locks the block with this number. **Attention: Process is irreversible.**

### 4.11. Write AFI

**int TSHRW\_ISO\_WriteAFI**(int hPort, int AFI, BYTE \* pUID, int UIDLen,  
int nRequestFlag)

hPort	logical device number
int AFI	AFI (see ISO 15693)
pUID	pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored.
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID will be ignored.
RequestFlag	Request Flag see ISO 15693
Return	< 0 error, 0 = OK

Writes „Application family identifier“ into the transponder.  
See table: AFI coding in sheet **ISO/IEC 15693 – 3**. E.g.: Transport or financial chip.

### 4.12. Lock AFI

**int TSHRW\_ISO\_LockAFI**(int hPort, BYTE \* pUID, int UIDLen, int nRequestFlag)

hPort	logical device number
pUID	pointer to UID. Request flag is set automatically. If pointer is NULL, the UID will be ignored.
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
RequestFlag	Request Flag see ISO 15693
Return	< 0 error, 0 = OK

Locks AFI (Application family identifier)



## Programming interface (SDK) of the TS-HRW Series

### 4.13. Write DSFID

**int TSHRW\_ISO\_WriteDSFID**(int hPort, int DSFID, BYTE \* pUID, int UIDLen,  
int nRequestFlag)

hPort	logical device number
int DSFID	DSFID
pUID	pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored.
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
RequestFlag	Request Flag see ISO 15693
Return	< 0 error, 0 = OK

Writes self defined “data storage format identifier” DSFID

### 4.14. Lock DSFID

**int TSHRW\_ISO\_LockDSFID**(int hPort, BYTE \* pUID, int UIDLen, int nRequestFlag)

hPort	logical device number
pUID	pointer to UID. Request flag is set automatically. If pointer is NULL, the UID is ignored.
UIDLen	length of blocks (must be 8). If Length != 8 parameter pUID is ignored.
RequestFlag	Request Flag see ISO 15693
Return	< 0 error, 0 = OK

Locks “data storage format identifier” DSFID



## Programming interface (SDK) of the TS-HRW Series

### 4.15. Raw request

```
int TSHRW_ISO_RawRequest(int hPort, int RequestType, int RequestFlag,  
                        int Command, BYTE * pParam, int ParamLen, BYTE * pData,  
                        int DataLen, BYTE * pReceive, int ReceiveLen)
```

hPort	logical device number
RequestType	Request type = specific reader flag
RequestFlag	Request Flag see ISO Norm 15693
Command	ISO Command
pParam	pointer to parameter
ParamLen	number of parameter
pData	pointer to data
DataLen	number of data
pReceive	pointer to receive buffer
ReceiveLen	length of receive buffer
Return	< 0 error, Length of read data

With raw request you can send an arbitrary command to the transponder. The answer is pure and must be evaluated by the user. There are no specific assumptions inside. You find the general structure in data sheet **ISO/IEC 15693 – 3** chapters 7.3.





## ***5. Commands for Transponder Type MIFARE® (ISO14443A)***

Die Commands for MIFARE® transponders are supported only at some devices (only TS-HRW38 and TS-HRW32).

### **5.1. Application hints**

There are different types of MIFARE® transponders available.

Using this programming interface MIFARE® Classic, MIFARE® Ultralight (NFC Type 2) and MIFARE® DESFire transponders are supported.

#### **5.1.1. MIFARE® Ultralight (NFC Type 2)**

Today there are different MIFARE® Ultralight compatible transponders available from different manufacturers. So you have to check the datasheet of the transponder to get the memory configuration.

Original MIFARE® Ultralight Transponders have no crypto engine and so the access is easier to use. The original MIFARE® Ultralight Transponder do have a block structure with 4 Bytes per Block and totally 16 blocks. Only blocks 4 – 15 are user defined.

The MIFARE® Ultralight C Transponder does have 44 Blocks while the NTAG216F has up to 230 blocks. All these and more Tags are supported.

To access any MIFARE® Ultralight compatible transponder the transponder has to be selected first.

This is done using the functions

TSHRW\_Mifare\_Request()

TSHRW\_Mifare\_Select()

After this, the function TSHRW\_MifareRead() and TSHRW\_MifareWrite() can be used to access the blocks.

The functions TSHRW\_Mifare\_Authenticate(), TSHRW\_MifareGetValue(),

TSHRW\_MifareSetValue() and TSHRW\_MifareChangeValue() are not supported by this chip type.



## **Programming interface (SDK) of the TS-HRW Series**

### **5.1.2. MIFARE® Classic**

The MIFARE® Classic Chip is available in 2 Variants:

MIFARE® 1K (1KByte EEPROM) and

MIFARE® 4K (4KByte EEPROM)

MIFARE® Classic transponders do have a crypto engine and the access is only possible after authentication.

First of all also

`TSHRW_Mifare_Request()`

`TSHRW_Mifare_Select()`

has to be used to select the Chip.

After this the memory range which shall be accessed is enabled using

`TSHRW_MifareAuthenticate()` or `TSHRW_MifareAuthenticateDirect()`.

These calls differ in the mode how the access to the key is done.

At `AuthenticateDirect` the key is given directly, at `Authenticate` a previously stored key (using `SetKey`) is used.

The TS-HW38 stores up to 15 keys of key type A and B separately.

With this an effective use of the keys without the necessity of having the keys available all the time is possible.

Is also possible to store the key in the device previously, so the end user does not need to get the key in plain text.

The authentication is only valid for the given sector.

For each sector, choose the access rights separately.

The data sheet of the transponder describes this.

### **5.1.3. MIFARE® DESFire**

The MIFARE® DESFire transponders are microcontroller based. The security mechanism might be selected per usage

MIFARE® DESFire supports the different cryptography methods DES, 3DES und AES.

The communication can be done in plain or enciphered. This is set up in the configuration of the MIFARE® DESFire chip

The MIFARE® DESFire transponder includes a file system, multiple applications with several files can be created. This is totally different to MIFARE® Classic and Ultralight transponder where access is done using blocks.

The access to MIFARE® DESFire transponders uses ISO14443A-4 specification.



## Programming interface (SDK) of the TS-HRW Series

### 5.2. Function calls MIFARE® common

These functions are used to access a MIFARE® (ISO14443A) card. This is the same for all variants of MIFARE®. After selection of the MIFARE® card then the commands differ depending on the type of card.

**int TSHRW\_Mifare\_Request(int hPort, WORD \*pATQA)**

hPort                      logical device number  
pATQA                     pointer to Answer To Request A  
Return                     < 0 error, 0 OK

Only the lowest byte in ATQA is of interest.

Here you have:

0x0044:                  MIFARE® ultralight or MIFARE® 1K (7BUID) found  
0x0004:                  MIFARE® 1K (4BUID) found  
0x0002:                  MIFARE® 4K (4BUID) found  
0x0042:                  MIFARE® 4K (7BUID) found  
0x0344:                  MIFARE® DESFire found

**int TSHRW\_Mifare\_RequestAll(int hPort, WORD \*pATQA)**

hPort                      logical device number  
pATQA                     pointer to Answer To Request A  
Return                     < 0 error, 0 OK

Only the lowest byte in ATQA is of interest.

Here you have:

0x0044:                  MIFARE® ultralight or MIFARE® 1K (7BUID) found  
0x0004:                  MIFARE® 1K (4BUID) found  
0x0002:                  MIFARE® 4K (4BUID) found  
0x0042:                  MIFARE® 4K (7BUID) found  
0x0344:                  MIFARE® DESFire found



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Mifare\_Select**(int hPort, BYTE \* pUID, int nUIDLen, BYTE \* pSAK)

hPort	logical device number
pUID	pointer to UID of the transponder
nUIDLen	length of UID
pSAK	point to Select Acknowledge
Return	< 0 error, Length of read UID

This command selects the first found chip and gives the UID of this chip and also the Select Acknowledge value which gives information's about the chip type.

The SAK Value has the following information:

0x00:	MIFARE Ultralight® with 7 Byte UID
0x08	MIFARE® 1K
0x18	MIFARE® 4K
0x20	MIFARE® DESFire 7 Byte UID

**int TSHRW\_Mifare\_SelectUID**(int hPort, BYTE \* pUID, int nUIDLen, BYTE \* pSAK)

hPort	logical device number
pUID	pointer to UID of the transponder
nUIDLen	length of UID
pSAK	point to Select Acknowledge
Return	< 0 error, Length of read data

This command selects the chip with the given UID if the chip is in field and returns the Select Acknowledge value which gives information's about the chip type.

The SAK Value has the following information:

0x00:	MIFARE Ultralight® with 7 Byte UID
0x08	MIFARE® 1K
0x18	MIFARE® 4K
0x20	MIFARE® DESFire 7 Byte UID

**int TSHRW\_Mifare\_Halt**(int hPort)

hPort	logical device number
Return	< 0 error, 0 OK

The actual selected chip is stopped.

After this another known chip can be activated using TSHRW\_Mifare\_SelectUID().



## Programming interface (SDK) of the TS-HRW Series

### 5.3. Function calls MIFARE® Classic and Ultralight

**int TSHRW\_MifareAuthenticate**(int hPort, BYTE \* pUID, int nUIDLen,  
int KeyType, int nAdresse, int BlockNr)

hPort	logical device number
pUID	pointer to UID of transponder or NULL if the selected transponder has to be used.
nUIDLen	length of UID or 0 if the selected transponder has to be used.
KeyType	Type of Key 0: KeyA 1: KeyB
nAdresse	Index of Keys in Key-Memory
BlockNr	Number of block which is authenticated.
Return	< 0 error, 0 OK

Authenticates the sector which contains the given block.

A stored key is used. Keys can be stored permanently using TSHRW\_MifareSetKey().

MifareAuthenticate is only supported by MIFARE® Classic.

**int TSHRW\_MifareAuthenticateDirect**(int hPort, BYTE \* pUID, int nUIDLen,  
int KeyType, BYTE \* pKey, int nKeyLen, int BlockNr)

hPort	logical device number
pUID	pointer to UID of transponder or NULL if the selected transponder has to be used.
nUIDLen	length of UID or 0 if the selected transponder has to be used.
KeyType	Type of Key 0: KeyA 1: KeyB
pKey	Pointer to key data
nKeyLen	length of Key-Data, always 6
BlockNr	Number of block which is authenticated.
Return	< 0 error, 0 OK

Authenticates the sector which contains the given block.

The given key is used.

MifareAuthenticateDirect is only supported by MIFARE® Classic.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_MifareSetKey**(int hPort, int KeyType,  
int nAdresse, BYTE \* pKey, int nKeyLen)

hPort	logical device number
KeyType	Type of key 0: KeyA 1: KeyB
nAdresse	Index of key in Key-Memory
pKey	Pointer to key data
nKeyLen	length of Key-Data, always 6
Return	< 0 error, 0 OK

The given key is stored in the device at the given key address.  
Up to 15 Keys for KeyA and KeyB can be stored.

**int TSHRW\_MifareRead**(int hPort, int BlockNr, BYTE \* pData, int nDataLen)

hPort	logical device number
BlockNr	Number of the block to be read.
pData	Data of the block
nDataLen	Length of data
Return	< 0 error, Length of read data

Read a block from the transponder. To do this the selection and eventually the authentication has to be done before.

Always 16 Bytes are read. At MIFARE Ultralight® the block length is 4 Bytes, so always 4 blocks are read beginning with the given block number.

**int TSHRW\_MifareWrite** (int hPort, int BlockNr, BYTE \* pData, int nDataLen)

hPort	logical device number
BlockNr	Number of the block to be written.
pData	Data of the block
nDataLen	Length of data
Return	< 0 error, 0 OK

Write a block to the transponder. To do this the selection and eventually the authentication has to be done before.

The length of data depends on the Transponder type and is 4 Bytes at MIFARE Ultralight® and 16 Bytes at MIFARE® 1K and MIFARE® 4K.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_MifareGetValue**(int hPort, int BlockNr, int \*pValue)

hPort	logical device number
BlockNr	Number of the block to be read.
pValue	pointer to value
Return	< 0 error, 0 OK

Read a block in value format. This function can be used only if the block is written in value format. The value formation is only supported by MIFARE® 1K and MIFARE® 4K Chips. To do this the selection and the authentication has to be done before.

**int TSHRW\_MifareSetValue**(int hPort, int BlockNr, int Value)

hPort	logical device number
BlockNr	Number of the block to be written.
Value	Value to be written
Return	< 0 error, 0 OK

Write a block in value format. This function formats the block in value format also. The value formation is only supported by MIFARE® 1K and MIFARE® 4K Chips. To do this the selection and the authentication has to be done before.

**int TSHRW\_MifareChangeValue**(int hPort, int BlockNr, int Richtung, int Differenz)

hPort	logical device number
BlockNr	Number of the block to be written.
Richtung	Direction of value change 0 = Decrement, 1 = Increment
Differenz	absolute amount to be changed
Return	< 0 error, 0 OK

Change a block in value format. This function can be used only if the block is written in value format. The value formation is only supported by MIFARE® 1K and MIFARE® 4K Chips. To do this the selection and the authentication has to be done before. Depending on the given Direction the block is incremented or decremented by the given different. Depending on the access rights the actual authentication may only allow increment or decrement!



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_ISO14443A\_Transmit** (int handle, BYTE \*pSendData, int nSendLen,  
BYTE \* pRecvData, int RecvLen);

handle	logical device number
pSendData	data to be sent
nSendLen	length of data to be sent
pRecvData	data received
RecvLen	max Length of data to be received
Return	< 0 error, Length of received data

With this any command can be sent to an activated ISO14443A card. (Only available at TS-HRW380)

### 5.4. Function calls ISO14443A-4

After selection of cards using ISO1443A-3 from chapter 5.2. the access to the ISO14443A-4 commands is possible after activation of the ISO14443A-4 command level. For this the activation commands in chapter 5.4.1. are used.

#### 5.4.1. ISO14443-4 activation

**int TSHRW\_ISO14443SetCID** (int hPort, int CID)

hPort	logical device number
CID	CardIdentifier, Identification number of the card, if always only one card is activated, this can be kept as 1.
Return	< 0 error, 0 OK

With this function the CardIdentifier is set, which is used at all following commands. Multiple cards can be selected, each of them needs a separate card identifier.

**int TSHRW\_ISO14443Rats** (int hPort, BYTE \* pData, int RecvBufLen)

hPort	logical device number
pData	Answer To Select Data
RecvBufLen	Max. Length of data buffer
Return	< 0 error, Length of read data

This function is used to get the selection response (Requ<sup>st</sup> for Ans<sup>wer</sup> to Select)  
The meaning of the bytes in the selection response is described in the MIFARE® DESFire data sheet.





## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_ISO14443PPS** (int hPort, int Speed)

hPort                      logical device number

Speed                      transmission speed

Return                      < 0 error, 0 OK

With this the parameter of the transmission is defined.

The transmission speed PCD → PICC and PICC → PCD can be defined using the following table:

Speed	PCD → PICC	PICC → PCD	Speed	PCD → PICC	PICC → PCD
0 (0000)	106 kBit	106 kBit	8 (1000)	106 kBit	424 kBit
1 (0001)	212 kBit	106 kBit	9 (1001)	212 kBit	424 kBit
2 (0010)	424 kBit	106 kBit	10 (1010)	424 kBit	424 kBit
3 (0011)	848 kBit	106 kBit	11 (1011)	848 kBit	424 kBit
4 (0100)	106 kBit	212 kBit	12 (1100)	106 kBit	848 kBit
5 (0101)	212 kBit	212 kBit	13 (1101)	212 kBit	848 kBit
6 (0110)	424 kBit	212 kBit	14 (1110)	424 kBit	848 kBit
7 (0111)	848 kBit	212 kBit	15 (1111)	848 kBit	848 kBit

All other values are invalid!

**int TSHRW\_ISO14443Transmit** (int hPort, BYTE \*pSendData, int nSendLen,  
BYTE \* pRecvData, int RecvLen)

hPort                      logical device number

pSendData                data to be sent

nSendLen                length of data to be sent

pRecvData                data received

RecvLen                max Length of data to be received

Return                      < 0 error, Length of received data

With this any command can be sent to an activated ISO14443-4 card.

So all commands for MIFARE® DESFire cards can be mapped to this function.



## Programming interface (SDK) of the TS-HRW Series

### 5.5. Function calls ISO14443B

These functions are only supported at TS-HRW32 in the moment.

**int TSHRW\_ISO14443B\_Request** (int handle, BYTE \* pUID, int UIDLen,  
BYTE \* pAppData, int AppDataLen,  
BYTE \* pProtocolInfo, int ProtocolInfoLen);

handle	logical device number
pUID	Pointer to PUPI of transponders
UIDLen	max. Length of PUPI
pAppData	Pointer to application data
AppDataLen	max. Length of application data
pProtocolInfo	Pointer to protocol information
ProtocolInfoLen	max. Length of protocol information
Return value:	< 0 Fehler, 0 OK

With this command the REQB Request is sent to the tag and the ATQB response is received.

pUID receives the 4 byte PUPI (Pseudo Unique PICC Identifier)

pAppData receives the 4 byte application data with AFI, CRC\_B and Application count

pProtocolInfo receives 3 byte of protocol information.

Please refer to the ISO14443-3 documentation for further description.

**int TSHRW\_ISO14443B\_Wakeup** (int handle, BYTE \* pUID, int UIDLen,  
BYTE \* pAppData, int AppDataLen,  
BYTE \* pProtocolInfo, int ProtocolInfoLen);

handle	logical device number
pUID	Pointer to PUPI of transponders
UIDLen	max. Length of PUPI
pAppData	Pointer to application data
AppDataLen	max. Length of application data
pProtocolInfo	Pointer to protocol information
ProtocolInfoLen	max. Length of protocol information
Return value:	< 0 Fehler, 0 OK

With this command the WUPB Request is sent to the tag and the ATQB response is received.

pUID receives the 4 byte PUPI (Pseudo Unique PICC Identifier)

pAppData receives the 4 byte application data with AFI, CRC\_B and Application count

pProtocolInfo receives 3 byte of protocol information.

Please refer to the ISO14443-3 documentation for further description.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_ISO14443B\_Select** (int handle, int Speed, BYTE \* pUID, int UIDLen);  
 handle                      logical device number  
 Speed                        transmission speed  
 pUID                         Pointer to PUPI of transponders  
 UIDLen                      max. Length of PUPI

With this the ATTRIB command is sent to the transponder.

Using Speed the transmission speed PCD → PICC and PICC → PCD can be defined using the following table:

Speed	PCD → PICC	PICC → PCD	Speed	PCD → PICC	PICC → PCD
0 (0000)	106 kBit	106 kBit	8 (1000)	106 kBit	424 kBit
1 (0001)	212 kBit	106 kBit	9 (1001)	212 kBit	424 kBit
2 (0010)	424 kBit	106 kBit	10 (1010)	424 kBit	424 kBit
3 (0011)	848 kBit	106 kBit	11 (1011)	848 kBit	424 kBit
4 (0100)	106 kBit	212 kBit	12 (1100)	106 kBit	848 kBit
5 (0101)	212 kBit	212 kBit	13 (1101)	212 kBit	848 kBit
6 (0110)	424 kBit	212 kBit	14 (1110)	424 kBit	848 kBit
7 (0111)	848 kBit	212 kBit	15 (1111)	848 kBit	848 kBit

All other values are invalid!

In pUID the PUPI is given, which was received by TSHRW-ISO14443B\_Request.

**int TSHRW\_ISO14443B\_Transmit** (int handle, BYTE \*pSendData, int nSendLen, BYTE \* pRecvData, int RecvLen);  
 handle                      logical device number  
 pSendData                  data to be sent  
 nSendLen                  length of data to be sent  
 pRecvData                  data received  
 RecvLen                    max Length of data to be received  
 Return                      < 0 error, Length of received data

With this any command can be sent to an activated ISO14443B card.



## **Programming interface (SDK) of the TS-HRW Series**

### **5.6. Function calls Mifare DESFire**

To use DESFire functions the card has to be activated as described in chapter 5.2. and 5.4.

The DESFire card uses a flexible file system. This file system allows a maximum of 28 applications on a card. Each application provides up to 32 files. There are five different types of files.

An application is not an executable program. An application is a directory with files for data used by a program which runs on an external device which can communicate with the DESFire card.

For changing settings of the card or creating an application, select card level (Application ID 0x00).

For changing settings of an application or creating and accessing files, select the associated application.

After selection of the card according to ISO ISO14443-4 the DESFire card level is selected.

Prior to data transmission a mutual three pass authentication can be done between card and reader device. After authentication, both sides, the card as well as the reader device, know that they can trust each other because both had to know the same secret key.

The authentication generates also a session key which is automatically stored both in card and in Desfire SDK, which can be used to keep the further communication path secure.

All further communication can be done in three different ways:

- Data is sent plain
- Data is sent plain with cryptographic checksum (CMAC), to avoid unnoticed replacement of card or reader device or man-in-the-middle-attacks
- Data is sent encrypted to avoid unnoticed replacement of card or reader device or man-in-the-middle-attacks and additionally to avoid unwanted listening.

The type of encryption can be either:

- DES with 8 bytes, or 16 bytes with two equal halves. Least significant bit of each byte is only used for key version
- 3DES with 16 bytes, the least significant bit of each byte is only used for key version and will be ignored for encryption
- AES with 16 bytes

It is possible to set configuration of card and applications in such a way that all changes of settings, creating / deleting of applications and files and reading / writing files need a prior authentication and transmitted data is enciphered.



## Programming interface (SDK) of the TS-HRW Series

### 5.6.1. Security commands

Depending on security settings a reader device has to authenticate first to change structure, data or settings on the card. After authentication, both sides, the card as well as the reader device, know that they can trust each other, because both had to know the same secret key. The authentication generates also a session key which is automatically stored in card and in Desfire SDK, which can be used to keep the communication path secure.

The authentication is invalidated by

- Selecting an application

- Changing the key used for reaching the current authentication state

- Command failure

The security uses DES/3DES or AES enciphering.

Keys for DES or 3DES encryption have a length of 16 bytes. The least significant bit of each byte is used for key version and will be ignored for encryption. It is recommended not to use parity bits of key for setting least significant bits.

Keys for AES encryption have a length of 16 bytes, key version is stored separated.

The version of a key is stored with the key on card and can be used für own purposes, the Desfire card itself don't check key version in any respect.

The Security commands are used in card level and application level.

**int TSHRW\_Desfire\_Authenticate** (int hPort, BYTE KeyNo, BYTE\* pKey, int KeySize)

hPort                      logical device number

KeyNo                     Number of key

pKey                      Pointer to buffer with key. Key has length of 16 bytes.

KeySize                  Length of key

Return:                   -1 error, 0 OK

Authenticates with specified key.

The currently selected application or card level has to use encryption type DES or 3DES.

This type of authentication is provided for backwards compatibility with older versions of Desfire cards. It is recommended to use AuthenticateIso if possible.

If the value of pKey was wrong, TSHRW\_GetLastError delivers error 100AE, "Current authentication status does not allow the requested command".



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_AuthenticateIso** (int hPort, BYTE KeyNo, BYTE\* pKey, int KeySize)

hPort	logical device number
KeyNo	Number of key
pKey	Pointer to buffer with key. Key has length of 16 bytes.
KeySize	Length of key
Return:	-1 error, 0 OK

Authenticates with specified key. This type of authentication is used if the currently selected application or card level uses encryption type DES or 3DES.

If the value of pKey was wrong, TSHRW\_GetLastError delivers error 100AE, “Current authentication status does not allow the requested command”.

**int TSHRW\_Desfire\_AuthenticateAES** (int hPort, BYTE KeyNo, BYTE\* pKey, int KeySize)

hPort	logical device number
KeyNo	Number of key
pKey	Pointer to buffer with key. Key has length of 16 bytes.
KeySize	Length of key
Return:	-1 error, 0 OK

Authenticates with specified key. This type of authentication is used if the currently selected application or card level uses encryption type AES.

If the value of pKey was wrong, TSHRW\_GetLastError delivers error 100AE, “Current authentication status does not allow the requested command”.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_GetKeySettings** (int hPort,  
 BYTE\* pKeySettings, int KeySettingsLen)

hPort                      logical device number

pKeySettings            Byte 0: encryption type in the application ( used in Authenticate(), CMAC)  
                               returns always 0 in card level  
                               0: (3)DES  
                               1: 3K3DES  
                               2: AES

Byte 1: maximum number of keys in the application

Byte 2: Key number to be used for authentication to change a key:  
           0: with master key,  
           1 to 13: with this key number, except for master key & the key itself  
           14: with the key to be changed,  
           15: all keys except master key are frozen

Byte 3:  
       Bit0: master key changeable,  
       Bit1: get file lists and app settings without app master key authent,  
              app lists and card settings without card master key authent  
       Bit2: create/delete file without app master key authenticate  
              create app without card master key authenticate  
              delete app also with app master key authenticate

Bit3: configuration changeable

SettingsLen              Length of pKeySettings

Return:                   -1 error, 0 OK

Provides security settings and maximum number of keys of the currently selected application (or card level, if selected with AID 0x00).

**int TSHRW\_Desfire\_ChangeKeySettings** (int hPort, BYTE ChangeKeyAuthentWith,  
 BYTE OtherKeySettings)

hPort                      logical device number

ChangeKeyAuthentWith   Key number to be used for authentication to change a key:  
                               0: with master key (MK)  
                               1 to 13: with this key number, except for MK & the key itself  
                               14: with the key to be changed,  
                               15: all keys except master key are frozen

OtherKeySettings        Bit0: master key changeable  
                               Bit1: get file lists and settings without master key authenticate  
                                       respectively app lists and settings in card level  
                               Bit2: create/delete file/app without master key authenticate  
                               Bit3: configuration changeable

Return:                   -1 error, 0 OK

Changes security settings of the currently selected application (or card level, if selected with AID 0x00). The number of keys in an application cannot be changed after creation.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_ChangeKey** (int hPort, BYTE KeyNo,  
BYTE\* pNewKey, int NewKeyLen, BYTE NewKeyVersion,  
BYTE\* pCurrentKey, int CurrentKeyLen,  
BYTE NewMasterKeyCrypto)

hPort	logical device number
KeyNo	Number of the key which should be changed
pNewKey	New key value. 16 bytes. For 3DES keys with integrated new key version
NewKeyLen	Länge des Schlüssels in pNewKey
NewKeyVersion	Key version, for AES keys. For 3DES keys ignored
pCurrentKey	Current value of key with KeyNo. 16 bytes. Only relevant if key number of key to be changed and key number of key used for necessary authentication are different.
CurrentKeyLen	Länge des Schlüssels in pCurrentKey
MasterKeyCrypto	Only used if card level is selected (AID==0x00). Set encryption type of card master key. 0: DES/3DES, 2: AES.
Return:	-1 error, 0 OK

Changes value and version of a key. If the card level is selected the card master key is changed. If application is selected the application key is changed.

The encryption type of card master key can be changed from 3DES to AES, but not vice versa. On application level the specified key within the currently selected application is changed, its encryption type is not changeable.

Prior the command an authentication is needed. KeyNo for authentication is depending of the key settings of the application.

If the value of pCurrentKey was wrong, TSHRW\_GetLastError delivers error 1001E, "Integrity error, CRC or MAC does not match data".

**int TSHRW\_Desfire\_GetKeyVersion** (int hPort, BYTE KeyNo)

hPort	logical device number
KeyNo	Key number
Return:	-1 error, >=0 key version with 1 byte

Provides the version of key specified with key number. No prior authentication is needed.





## Programming interface (SDK) of the TS-HRW Series

### 5.6.2. Card level commands

To use these commands the card level has to be activated.

**int TSHRW\_Desfire\_GetVersion** (int hPort, BYTE\* pVersionData, int VersionDataLen)

hPort	logical device number
pVersionData	Pointer to version data, 28 bytes
	Byte 0 to 6 hardware related information.
	byte 0: vendor ID
	byte 1: type
	byte 2: subtype
	byte 3: major version
	byte 4: minor version
	byte 5: storage size (e.g. 0x16 == 2048 bytes, 0x18 == 4096 bytes, 0x1a == 8192 bytes)
	byte 6: communication protocol type
	Byte 7 to 13 software related information.
	byte 7: vendor ID
	byte 8: type
	byte 9: subtype
	byte 10: major version
	byte 11: minor version
	byte 12: storage size (e.g. 0x16 == 2048 bytes, 0x18 == 4096 bytes, 0x1a == 8192 bytes)
	byte 13: communication protocol type
	Bytes 14 to 20: unique serial number
	Bytes 21 to 25: production batch number
	Byte 26: calendar week of production (BCD, binary coded decimal)
	Byte 27: year of production (BCD)
VersionDataLen	Length of version data
Return:	-1 error, 0 OK

Reads manufacturing related data of the card (28 bytes).

**int TSHRW\_Desfire\_GetFreePICCMemory** (int hPort)

hPort	logical device number
Return:	-1 error, >=0 available memory size

Provides the available memory of the card.



## Programming interface (SDK) of the TS-HRW Series

**bool TSHRW\_Desfire\_FormatPICC** (int hPort)

hPort                      logical device number

Return:                    -1 error, 0 OK

Releases all allocated user memory on the card for further use. All applications and all files are deleted. This cannot be rolled back. Releases allocated memory of earlier deleted applications and files. The card master key and the card master key settings keep their current values.  
The format command requires a preceding authentication with the card master key.

**int TSHRW\_Desfire\_GetCardUID** (int hPort, BYTE\* pUID, int UIDBufferLen)

hPort                      logical device number

pUIDBuffer                Pointer to buffer for UID

UIDBufferLen             Length of buffer for UID

Return:                    -1 error, >=0 number of bytes used in pUIDBuffer

Provides the UID of the card. The UID has a length of 7 bytes.

An authentication with any key needs to be performed prior to the command.

The command is needed if card is set to random UID. It is not applicable without random UID.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_SetConfiguration** (int hPort, BYTE Option,  
BYTE\* pSettings, int SettingsLen)

hPort	logical device number
Option	0: setting configuration byte, 1: setting default-key, 2: setting ATS (Answer to Select)
pSettings	Pointer to settings. Option=0: configuration byte bit 0 = 0: Format card enabled bit 0 = 1: Format card disabled (cannot be reset), bit 1 = 0: Random ID disabled bit 1 = 1: Random ID enabled (cannot be reset) Option=1: default key with version, 17 bytes. Byte 0-15 new default key Byte 16 key version (3DES and AES) Option=2: User defined ATS (Answer to Select) parameter
SettingsLen	Length of settings
Return:	-1 error, 0 OK

This function is used to change the configuration on card level.  
The command offers the choice of three different options:

If Random ID is enabled, the card returns a random generated UID at TSHRW\_Mifare\_Select command. To read the card UID, the reader device has to read the UID using TSHRW\_Desfire\_GetCardUID, this requires a preceding authentication

If default key is set, this key is used as initial value for all keys in newly created applications.

With option = 2 the RATS answer can be set. This should be done only by experts!  
Please see the MIFARE DESFire Functional Specification for details on this function.

### 5.6.3. Card Level Commands for Application management

An application includes files, up to 32 files in one application are possible. The application provides functionality to control access to its files.

**int TSHRW\_Desfire\_GetApplicationIds** (int hPort, BYTE\* pAppIds, int AppIdsLen)

pAIDBuffer	Pointer to buffer for AIDs. An array of AIDs will be written with 4 bytes per AID.
------------	--

Return: -1 error,  $\geq 0$  number of IDs returned in pAIDBuffer, each with 4 bytes

[illegible]

**Return:** -1 error,  $\geq 0$  number of bytes used in pAppNamesArray

Applications without DF name are ignored. Command cannot be used in authentication status after authentication with AuthenticateISO or AuthenticateAES

**int TSHRW\_Desfire\_SelectApplication** (int hPort, int AppId)

Selects an application. Selection with AppId 0 selects card level.

hPort	logical device number
AppId	Application ID. 3 Bytes. Must be unique. 0 is reserved for card level.
KeyCrypto	Type of encryption (e.g. for Authenticate(), CMAC) 0: DES/3DES, 2: AES
MaxNoOfKeys	Count of keys (maximal 14)
bFilesWithIsoID	0: files with ISO7816 file ID are not possible. 1: further created files within this application have ISO7816 file ID Independent of application having ISO7816 application ID.
ChangeKeyAuthentWith	Key number to be used for authentication to change a key (Change Key): 0: with application master key 1 to 13: with this key number, except for master key & the key itself, master key and change key need authentication with master key 14: with the key to be changed 15: all keys except master key are frozen
OtherKeySettings	Bit 0: master key is changeable Bit 1: get file lists and settings without authenticate Bit 2: create/delete file without master key Bit 3: configuration is changeable
pIso7816Data	Pointer to ISO7816 application ID, NULL: ignore
Iso7816DataLen	ISO AID with 2 bytes + DF name (can be empty)
Return:	Length of ISO7816-Application-ID, 0: ignore pIso7816Data -1 error, 0 OK

Creates an application. Encryption type and number of keys cannot be changed after creation.  
All keys are initialized with the default key.  
The command requires that card level is selected.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_DeleteApplication** (int hPort, int AppId)

hPort                      logical device number

AppId                      Application ID

Return:                    -1 error, 0 OK

Deletes an application with all its associated files.

The command requires that card level is selected.

Memory of deleted application and deleted files will not be released. The only way to release memory is formatting the card.



## **Programming interface (SDK) of the TS-HRW Series**

### **5.6.4. File level commands**

All file commands refer only to the files in the currently selected application. A file is specified with its file identifier (one byte) which must be unique within its associated application.

For several commands authentication is needed, depending of key settings.

There are five different file types:

- Standard Data Files and Backup Data Files are used to store unformatted user data.
- Backup Data Files have additionally an interior backup mechanism, see remarks below.
- Value Files are used to store and manipulate a 32bit signed integer value.
- Linear Record Files and Cyclic Record Files are used to store structural data.

They consist of an array of records and one current record.

Cyclic and Linear Record Files have only one difference: if all records in file are filled and an additional record should commit, a Linear Record File will return an error, a Cyclic Record File will overwrite the oldest record.

All file types except Standard Data File work with interior backup mechanism. A writing operation changes a mirror file and not the file itself, a following reading operation will read the old data. A CommitTransaction is needed to validate changes in a file. Selection of another application without previous commit, removing card from reader without previous commit or an AbortTransaction will invalidate all changes since last CommitTransaction and conserve original data.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_GetFileIds** (int hPort, BYTE\* pFileIds, int FileIdsLen)

hPort                      logical device number

pFileIds                  Pointer to buffer for File IDs.

An array of IDs will be written with 1 byte per File ID.

FileIdsLen                Length of pFileIds

Return:                    -1 error, >=0 number of File IDs returned in pFileIDs

Provides the file identifiers (File ID) of all files in the currently selected application.

**int TSHRW\_Desfire\_GetFileIsoIds** (int hPort, BYTE\* pFileIds, int FileIdsLen)

hPort                      logical device number

pFileIds                  Pointer to buffer for ISO7816 File IDs An array of IDs will be written with  
2 bytes per ID.

FileIdsLen                Length of pFileIds

Return:                    -1 error, >=0 number of File IDs returned in pFileIDs, each with 2 bytes

Provides the file identifiers according to ISO7816-4 (ISO File ID) of all files in the currently selected application. Files without ISO7816 File ID are ignored.





## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_GetFileSettings** (int hPort, int FileId,  
BYTE\* pFileSettings, int FileSettingsBufferLen)

hPort	logical device number
FileId	File ID
pFileSettings	<p>Byte 0: File type</p> <p>0: Standard Data File, 1: Backup Data File, 2: Value File, 3: Linear Record File, 4: Cyclic Record File</p> <p>Byte 1: communication mode</p> <p>0: Plain communication 1: Plain communication secured by MACing 3: Fully enciphered communication</p> <p>Bytes 2-3: access rights</p> <p>Bits 0-3: Change AccessRights Bits 4-7: Read &amp; Write Access Bits 8-11: Write Access Bits 12-15: Read Access</p> <p>for each access right:</p> <p>0x0 - 0xd: key number used for authentication 0xe: free access, 0xf: deny access,</p> <p>For Standard and Backup Data File:</p> <p>Bytes 4-6: useable file size</p> <p>For Value File:</p> <p>Bytes 4-7: lower limit Bytes 8-11: upper limit Bytes 12-15: current maximal limited credit value Byte 16: 0x00: Limited credit command disabled 0x01: Limited credit command allowed</p> <p>For Linear and Cyclic Record File:</p> <p>Bytes 4-7: record size Bytes 8-11: maximal number of records Bytes 12-15: current number of records</p>
FileSettingsBufferLen	Length of pFileSettings
Return:	-1 error, >=0 Length of data written to pFileSettings

Provides information on the properties of a specified file. The information is file type, communication mode, access rights and other properties which depend on the file type.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_ChangeFileSettings** (int hPort, int FileId,  
int bChangeAccessRightsFree,  
BYTE\* pFileSettings, int FileSettingsLen)

hPort	logical device number
FileId	File ID
bChangeAccessRightsFree	1: Free access without authentication. 0: Authentication needed.
pFileSettings	Byte 0: communication mode 0: Plain communication 1: Plain communication secured by MACing 3: Fully enciphered communication Bytes 1-2: access rights Bits 0-3: Change AccessRights Bits 4-7: Read & Write Access Bits 8-11: Write Access Bits 12-15 Read Access for each access right: 0x0 - 0xd: key number used for authentication 0xe: free access, 0xf: deny access
FileSettingsLen	Length of data in pFileSettings
Return:	-1 error, 0 OK

Changes communication mode and access rights of a specified file.  
Type of file and file size could not be changed after file creation.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_CreateStandardDataFile** (int hPort, BYTE FileId,  
int IsoFileId, int FileSize,  
BYTE CommunicationMode,  
int AccessRights)

<b>hPort</b>	logical device number
<b>FileId</b>	File ID. Within the range 0x00 to 0x1f. Must be unique within the currently selected application. It is not necessary to create the files within the application in a special order,
<b>IsoFileId</b>	File identifier according to ISO7816-4 (ISO File ID). 2 bytes. Within the range 0x0001 und 0xffff. Must be unique within the currently selected application. For commands according to ISO7816-4 like TSHRW_Desfire_IsoSelectFile. If this value is 0 no ISO File ID is set.
<b>FileSize</b>	Need and possibility dependent on application setting "files have ISO ID" Usable file size. Cannot be changed after creation.
<b>CommunicationMode</b>	Mode for file access: 0: plain, 1: plain secured by MAC, 3: fully enciphered
<b>AccessRights</b>	Bits 0-3: Change AccessRights Bits 4-7: Read & Write Access Bits 8-11: Write Access Bits 12-15 Read Access for each access right: 0 - D <sup>hex</sup> : key number used for authentication E <sup>hex</sup> : free access, F <sup>hex</sup> : deny access
<b>Return:</b>	-1 error, 0 OK

Creates a Standard Data File within the selected application.

This file type is used for the storage of unformatted user data. It works without backup mechanism.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_CreateBackupDataFile** (int hPort, BYTE FileId,  
int IsoFileId, int FileSize,  
BYTE CommunicationMode,  
int AccessRights)

**hPort** logical device number

**FileId** File ID. Within the range 0x00 to 0x1f. Must be unique within the currently selected application.

**IsoFileId** File identifier according to ISO7816-4 (ISO File ID).  
2 bytes. Within the range 0x0001 und 0xffff.  
Must be unique within the currently selected application.  
For commands according to ISO7816-4 like  
TSHRW\_Desfire\_IsoSelectFile.  
If this value is 0 no ISO File ID is set.

**FileSize** Need and possibility dependent on application setting “files have ISO ID”  
Usable file size. Cannot be changed after creation.

**CommunicationMode** Mode for file access:  
0: plain,  
1: plain secured by MAC,  
3: fully enciphered

**AccessRights** Bits 0-3: Change AccessRights  
Bits 4-7: Read & Write Access  
Bits 8-11: Write Access  
Bits 12-15 Read Access  
for each access right:  
0- D<sup>hex</sup>: key number used for authentication  
E<sup>hex</sup>: free access,  
F<sup>hex</sup>: deny access

**Return:** -1 error, 0 OK

Creates a Backup Data File within the selected application.

This file type is used for the storage of unformatted user data. It works with interior backup mechanism.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_CreateValueDataFile** (int hPort, BYTE FileId, BYTE CommunicationMode, int AccessRights, int LowerLimit, int UpperLimit, int InitialValue, BYTE Config)

**hPort** logical device number

**FileId** File ID. Within the range 0x00 to 0x1f. Must be unique within the currently selected application.

**CommunicationMode** Mode for file access:  
 0: plain,  
 1: plain secured by MAC,  
 3: fully enciphered

**AccessRights** Bits 0-3: Change AccessRights  
 Bits 4-7: Read & Write Access  
 Bits 8-11: Write Access  
 Bits 12-15: Read Access  
 for each access right:  
 0 - D<sup>hex</sup>: key number used for authentication  
 E<sup>hex</sup>: free access,  
 F<sup>hex</sup>: deny access

**LowerLimit** Minimal value possible

**UpperLimit** Maximal value possible

**InitialValue** Initial value of value file

**Config** Bit 0: „LimitedCredit“ feature, 1: enabled, 0: disabled  
 Bit 1: „Free GetValue“ feature, 1: enabled, 0: disabled

**Return:** -1 error, 0 OK

Creates a Value File within the selected application.

This file type is used to store and manipulate a 32bit signed integer value.

It works with interior backup mechanism.



## Programming interface (SDK) of the TS-HRW Series

<b>int TSHRW_Desfire_CreateLinearRecordFile</b>	(int hPort, BYTE FileId, int IsoFileId, BYTE CommunicationMode, int AccessRights, int RecordSize, int MaxCountRecords)
<b>hPort</b>	logical device number
<b>FileId</b>	File ID. Within the range 0x00 to 0x1f. Must be unique within the currently selected application.
<b>IsoFileId</b>	File identifier according to ISO7816-4 (ISO File ID). 2 bytes. Within the range 0x0001 und 0xffff. Must be unique within the currently selected application. For commands according to ISO7816-4 like TSHRW_Desfire_IsoSelectFile. If this value is 0 no ISO File ID is set. Need and possibility dependent on application setting “files have ISO ID”
<b>CommunicationMode</b>	Mode for file access: 0: plain, 1: plain secured by MAC, 3: fully enciphered
<b>AccessRights</b>	Bits 0-3: Change AccessRights Bits 4-7: Read & Write Access Bits 8-11: Write Access Bits 12-15 Read Access for each access right: 0 - D <sup>hex</sup> : key number used for authentication E <sup>hex</sup> : free access, F <sup>hex</sup> : deny access
<b>RecordSize</b>	Size of a record, within range from 1 to 0xfffff (16,777,215)
<b>MaxCountRecords</b>	Maximal number of records, range from 1 to 0xfffff (16,777,215)
<b>Return:</b>	-1 error, 0 OK

Creates a Linear Record File within the selected application.

This file type is used to store structural data. File consists of an array of records and one current record. If all records in file are filled and an additional record should commit, an error will returned. It works with interior backup mechanism.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_CreateCyclicRecordFile** (int hPort, BYTE FileId,  
int IsoFileId,  
BYTE CommunicationMode,  
int AccessRights,  
int RecordSize, int MaxCountRecords)

**hPort** logical device number

**FileId** File ID. Within the range 0x00 to 0x1f. Must be unique within the currently selected application.

**IsoFileId** File identifier according to ISO7816-4 (ISO File ID).  
2 bytes. Within the range 0x0001 und 0xffff.  
Must be unique within the currently selected application.  
For commands according to ISO7816-4 like  
TSHRW\_Desfire\_IsoSelectFile.  
If this value is 0 no ISO File ID is set.  
Need and possibility dependent on application setting “files have ISO ID”

**CommunicationMode** Mode for file access:  
0: plain,  
1: plain secured by MAC,  
3: fully enciphered

**AccessRights** Bits 0-3: Change AccessRights  
Bits 4-7: Read & Write Access  
Bits 8-11: Write Access  
Bits 12-15 Read Access  
for each access right:  
0 - D<sup>hex</sup>: key number used for authentication  
E<sup>hex</sup>: free access,  
F<sup>hex</sup>: deny access

**RecordSize** Size of a record, within range from 1 to 0xfffff (16,777,215)

**MaxCountRecords** Maximal number of records, range from 1 to 0xfffff (16,777,215).  
Because of 1 current record the maximal number of valid records is 1 less.

**Return:** -1 error, 0 OK

Creates a Cyclic Record File within the selected application.

This file type is used to store structural data. A file consists of an array of records and one current record. If all records in file are filled and an additional record should commit, the oldest record will be overwritten by newly committed record.

It works with interior backup mechanism.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_DeleteFile** (int hPort, BYTE FileId)

hPort                      logical device number

FileId                     File ID

Deletes a file within the selected application.

Allocated memory associated with the deleted file is not freed

The file identifier of the deleted file can be reused to create a new file within that application.

**int TSHRW\_Desfire\_ReadData** (int hPort, BYTE FileId, BYTE CommunicationMode,  
int StartIndex, int ReadAnz,  
BYTE\* pReadData, int ReadDataLen)

hPort                      logical device number

FileId                     File ID

CommunicationMode 0: data will be sent plain,  
1: plain secured by MAC,  
3: fully enciphered

StartIndex                Index of first byte to be read

ReadAnz                  Number of bytes to be read. 0: reading entire file, starting at StartIndex

pReadDataBuffer        Pointer to read buffer

ReadDataLen             Length of read buffer

Return:                   -1 error, >= 0 length of actually read data

Reads data from Standard Data File or Backup Data File.

Requires Read or Read&Write access right.

**int TSHRW\_Desfire\_WriteData** (int hPort, BYTE FileId, BYTE CommunicationMode, DWORD  
OffsetInFile, BYTE\* pWriteData, int WriteDataLen)

hPort                      logical device number

FileId                     File ID

CommunicationMode 0: data need to be sent plain,  
1: plain secured by MAC,  
3: fully enciphered

OffsetInFile              Index in file of first byte to be overwritten

pWriteData                Pointer to buffer with data for writing

WriteDataLen             Length of data in pWriteData

Return:                   -1 error, 0 OK

Writes data from Standard Data File or Backup Data File.

Requires Write or Read&Write access right.





## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_GetValue** (int hPort, BYTE FileId, BYTE CommunicationMode, unsigned int \* pValue)

hPort                      logical device number

FileId                     File ID

CommunicationMode      0: value need to be sent plain,

                              1: plain secured by MAC,

                              3: fully enciphered

                              If "FreeGetValue" is set in CreateValueDataFile, then use plain or plain with MAC depending on authentication.

pValue                     Pointer to buffer for value of file. A value has a length of 4 bytes.

Return:                    -1 error, 0 OK

Reads the currently stored value from Value File.

Requires Read, Write or Read&Write access right.

Can used without these access rights if file has attribute "FreeGetValue".

**int TSHRW\_Desfire\_Credit** (int hPort, BYTE FileId, BYTE CommunicationMode, unsigned int Value)

hPort                      logical device number

FileId                     File ID

CommunicationMode      0: value need to be sent plain,

                              1: plain secured by MAC,

                              3: fully enciphered

Value                      Amount to increase value. Must be positive.

Return:                    -1 error, 0 OK

Increases the value stored in a Value Files.

Requires Read&Write access right.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_LimitedCredit** (int hPort, BYTE FileId,  
BYTE CommunicationMode, unsigned int Value)

hPort	logical device number
FileId	File ID
CommunicationMode	0: value need to be sent plain, 1: plain secured by MAC, 3: fully enciphered
Value	Amount to increase value. Must be positive.
Return:	-1 error, 0 OK

Allows a limited increase of a value stored in a Value File without having full Read&Write permissions to the file. This feature can be enabled or disabled during value file creation. The value is limited to the debit value of the last transaction.  
Requires Write or Read&Write access right.

**int TSHRW\_Desfire\_Debit** (int hPort, BYTE FileId, BYTE CommunicationMode,  
unsigned int Value)

hPort	logical device number
FileId	File ID
CommunicationMode	0: value need to be sent plain, 1: plain secured by MAC, 3: fully enciphered
Value	Amount to decrease value with. Must be positive.
Return:	-1 error, 0 OK

Decreases the value stored in a Value Files.  
Requires Read, Write or Read&Write access right.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_ReadRecord** (int hPort, BYTE FileId, BYTE CommunicationMode,  
int StartRecord, int ReadCountRecords, int RecordSize,  
BYTE\* pReadDataBuffer, int ReadDataBufferLen)

hPort	logical device number
FileId	File ID
CommunicationMode	0: sent in plain, 1: plain secured by MAC, 3: fully enciphered
StartRecord	Index of newest record to be read, 0 is equivalent to newest created record
ReadCountRecords	Number of records to be read. 0: reading all records
RecordSize	Size of a record
pReadDataBuffer	Pointer to read buffer
ReadDataBufferLen	Length of read buffer
Return:	-1 error, >0 length in bytes of actually read data

Reads a set of complete records from a Linear Record File or Cyclic Record File.

Records are transmitted in chronological order. Starting with the oldest, which is ReadCountRecords-1 before the one addressed by StartRecord. If StartRecord is 0 then all records, from the oldest record up to and including the newest record are read. The current record in contrast is not committed yet and is not read.

Requires Read or Read&Write access right.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_WriteRecord** (int hPort, BYTE FileId,  
BYTE CommunicationMode, DWORD OffsetInRecord,  
BYTE\* pWriteData, int WriteDataLen)

hPort	logical device number
FileId	File ID
CommunicationMode	0: sent in plain, 1: plain secured by MAC, 3: fully enciphered
OffsetInRecord	Offset in current record
pWriteData	Pointer to buffer with data for writing
WriteDataLen	Length of data for writing
Return:	-1 error, 0 OK

Writes data in the current record of a Linear Record File or Cyclic Record File.

With a following CommitTransaction the current record will be validated, will become the newest validated record, can be read from now on but cannot be written any more, and a new current record will append at the end and will be filled with zeros. If all records in file are filled yet, a Linear Record File will return an error, a Cyclic Record File will overwrite the oldest record with the current record. An AbortTransaction will invalidate the writing.

Requires Write or Read&Write access right.

**int TSHRW\_Desfire\_ClearRecordFile** (int hPort, BYTE FileId)

hPort	logical device number
FileId	File ID
Return:	-1 error, 0 OK

Reset a Cyclic or Linear Record File to the empty state.

After executing this command the CommitTransaction command is needed. Before CommitTransaction command all WriteRecord commands will fail.

An AbortTransaction command will invalidate the clearance.

Requires Read&Write access right.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_CommitTransaction** (int hPort)

hPort                      logical device number

Return:                    -1 error, 0 OK

Validate all previous write access to Backup Data Files, Value Files and Record Files within an application.

**int TSHRW\_Desfire\_AbortTransaction** (int hPort)

hPort                      logical device number

Return:                    -1 error, 0 OK

Invalidate all previous write access to Backup Data Files, Value Files and Record Files within an application. The command uses integrated backup mechanism. It doesn't change authentication state.



## Programming interface (SDK) of the TS-HRW Series

### 5.6.5. Commands according to ISO 7816-4

These commands are according to ISO7816-4. It is not possible to mix their execution with execution of native DESFire commands. After selection of a card according to ISO14443 it is necessary to use either only native DESFire commands or only ISO7816 commands.

**int TSHRW\_Desfire\_IsoSelectApplication** (int hPort, BYTE\* pAppName,  
int AppNameLen)

hPort                      logical device number  
pAppName                 pointer to buffer for application name, equals DF-Name  
                             card level has 0xD2 76 00 00 85 01 00  
AppNamesArrayLen       Length of application name  
Return:                    -1 error, 0 OK

Select an application or card level.

**int TSHRW\_Desfire\_IsoSelectFile** (int hPort, int IsoFileId)

hPort                      logical device number  
IsoFileId                 ISO File ID  
Return:                    -1 error, 0 OK

Select a file within current application.

**int TSHRW\_Desfire\_IsoReadData** (int hPort, int StartIndex, int ReadCount,  
BYTE\* pReadDataBuffer, int ReadDataBufferLen)

hPort                      logical device number  
StartIndex                Index of first byte to be read  
ReadCount                Number of bytes to be read  
pReadDataBuffer         Pointer to read buffer  
ReadDataBufferLen       Length of read buffer  
Return:                    -1 error, > 0 length of actually read data

Reads data from Standard Data File or Backup Data File.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Desfire\_IsoWriteData** (int hPort, int StartIndex,  
BYTE\* WriteData, int WriteDataLen)

hPort	logical device number
StartIndex	Index in file of first byte to be overwritten
pWriteData	Pointer to buffer with data for writing
WriteDataLen	Length of data in pWriteData
Return:	-1 error, 0 OK

Writes data to Standard Data File or Backup Data File.



## Programming interface (SDK) of the TS-HRW Series

### **6. Commands for NFC (Near Field Communication)**

Using the SDK you can write data to transponders using the NFC specification in a comfortable way. These transponders can be read and used by other NFC enabled devices like many modern Smartphones.

#### **6.1. General remarks**

NFC stands for "**Near Field Communication**". This is an international transmission standard where many leading companies like Google, Microsoft, Nokia, Samsung and others are involved. Most modern smartphones with Android or Windows Phone operating system already have an NFC reader included.

If NFC is activated at the smartphone and a NFC tag comes near to the phone, the content is automatically read.

Depending on the content the system starts the corresponding app and offers the data.

For Example:

Contact information is stored in the tag in NFC Format. A contact information is like a name card and contains name, phone numbers, email address and so on.

If the tag is read, the contact information is found and the user is asked if he wants to add this contact. Then the contact information is added to the contacts (phone book).





## **Programming interface (SDK) of the TS-HRW Series**

### **6.2. Supported transponder types**

Memory sizes stated below are user data memory sizes of the transponders. To store NFC data there is some extra data space required for NFC protocol. So depending of the data type less data space is available for actual user data.

#### **6.2.1. NFC Type 2**

1. NXP MIFARE Ultralight® is able to store 48 Bytes of user data.
2. NXP MIFARE Ultralight® C is able to store 144 Bytes of user data.
3. NXP NTAG 203 is able to store 144 Bytes of user data.
4. Infineon my-d NFC is able to store 128 Bytes of user data.
5. Infineon my-d move NFC is able to store 128 Bytes of user data.
6. Further transponder compatible to NFC Type 2.

#### **6.2.2. NFC Type 4**

1. NXP MIFARE® DESFire with different memory sizes

#### **6.2.3. NFC Type 6**

1. NXP ICode SLI / SLIX with different memory sizes
2. TI TagIT HF with different memory sizes
3. Other ISO15693 compatible transponder.

Especially at NFC Type 6 not all ISO15693 compatible transponders are supported by all NFC devices. There are many NFC devices known which support only NXP transponder.

#### **6.2.4. NFC Type 7**

1. NXP MIFARE® Classic 1k is able to store 720 Bytes of user data.
2. NXP MIFARE® Classic 4k is able to store 3360 Bytes of user data.

Not supported by all NFC devices.



## Programming interface (SDK) of the TS-HRW Series

### 6.3. Supported types of user data

The JSON format is used to exchange user data with the SDK commands for reading and writing. JSON (JavaScript Object Notification) is an open-standard format that uses human-readable text to transmit data between applications.

JSON parsers exists for nearly all common programming languages.

Remarks to the following examples in JSON format:

**Spaces, line feeds and carriage returns** between and around syntactic elements are allowed and ignored by the parsers and this SDK.

The SDK supports following types of user data:

#### 6.3.1. Text

A NFC device which reads such a transponder shows the text given by the transponder immediately.

Example (JSON format):

```
{"TEXT":{"TITLE":"This is an example for a text"}}
```

or

```
{"TEXT":  
  {  
    "TITLE": "This is an ""example for a text"" with quotation marks"  
  }  
}
```

#### 6.3.2. WWW Address

A NFC device which reads such a transponder wishes to load and display the web address in the internet browser. Depending on the NFC device either the web address only or also an optional description is shown and the user has to accept opening it, or the browser is opened immediately and tries to load the address.

Example (JSON format):

```
{"BOOKMARK":  
  {  
    "URL": "https://www.exampleAddress.com",  
    "TITLE": "This is an optional description"  
  }  
}
```



## Programming interface (SDK) of the TS-HRW Series

### 6.3.3. Telephone

A NFC device which reads such a transponder opens the stored number in the phone app to call this number. Depending on the NFC device either the phone number only or also an optional description is shown and the user has to accept opening it, or the phone application is opened immediately and the phone number is showed (but not called).

Example (JSON format):

```
{ "TELEPHON":  
  {  
    "NUMBER": "12345678",  
    "TITLE": "This is an optional description"  
  }  
}
```

### 6.3.4. SMS

A NFC device which reads such a transponder shows the SMS given by the transponder so it can be edited and sent.

Depending on the NFC device either the SMS phone number only or also an optional description is shown and the user has to accept opening it, or the SMS editor is opened immediately and the SMS is shown (but not sent).

Example (JSON format):

```
{ "SMS":  
  {  
    "ADDRESS": "12345678",  
    "MESSAGE": "This is a message text",  
    "TITLE": "This is an optional description"  
  }  
}
```



## Programming interface (SDK) of the TS-HRW Series

### 6.3.5. Email

A NFC device which reads such a transponder opens the Email app and shows the mail. The user can edit or send the mail.

Example (JSON format):

```
{ "MAIL":  
  {  
    "ADDRESS": "example@exampleAddress.com",  
    "SUBJECT": "This is a subject",  
    "MESSAGE": "This is a message text",  
    "TITLE": "This is an optional description"  
  }  
}
```



## Programming interface (SDK) of the TS-HRW Series

### 6.3.6. Contact

A NFC device which reads such a transponder adds this contact to the phone book. Depending on the NFC device either the contact name only or also an optional description is shown and the user has to accept opening it, or the phone book is opened immediately and the contact is added.

A contact consists of first name, surname, company / organisation and any amount of phone numbers, email addresses and websites.

A phone number consists of number, place (0: unknown, 1: work, 2: private) and device type (0: unknown or fixed-line, 1: mobile, 2: fax).

Example (JSON format):

```
{ "VCARD":
{
  "FIRSTNAME": "This is a first name ",
  "SURNAME": "This is a surname ",
  "ORGANISATION": "This is a company or an organization ",
  "MAILS":
  {
    "MAIL0": "example@exampleAddress.de",
    "MAIL1": "secondExample@secondAddress.com ",
    "MAIL2": anotherExample@anotherAdresse.eu
  },
  "URLS":
  {
    "URL0": "https://www. ExampleAdresse.de",
    "URL1": http://www. secondAdresse.com
  },
  "PHONES":
  {
    "PHONE0":
    {
      "NUMBER": "12345678",
      "TYPE_PLACE": 2,
      "TYPE_DEVICE": 1
    },
    "PHONE1":
    {
      "NUMBER": "11223344"
    }
  }
}
```



## Programming interface (SDK) of the TS-HRW Series

### 6.4. Function calls NFC

**int TSHRW\_Nfc\_ReadTag** (int hPort, char\* pReadData, int ReadDataLen)

hPort                      logical device number

pReadData                Pointer to read buffer for JSON string with terminating 0

ReadDataLen              Length of read buffer

Return:                    -1 error, >= 0 length of actually read data (without terminating 0)

Reads from transponder user data in NFC format. The data is returned as string in JSON format.

**int TSHRW\_Nfc\_ReadUrl** (int hPort, char\* pUrl, int UriLen, char\* pText, int TextLen)

hPort                      logical device number

pUrl                        Pointer to read buffer for URL with terminating 0

UriLen                     Length of read buffer for URL

pText                        Pointer to read buffer for text (description) with terminating 0.

NULL: ignore text

TextLen                    Length of read buffer for text. 0: ignore text

Return:                    -1 error, >= 0 length of actually read URL (without terminating 0)

Reads from transponder user data in NFC format from type URL (Uniform Resource Locator, e.g. a web address) plus an optional description text. If user data is not a URL or not in NFC format, return value is -1.

**int TSHRW\_Nfc\_ReadText** (int hPort, char\* pText, int TextLen)

hPort                      logical device number

pText                        Pointer to buffer for Text with terminating 0

TextLen                    Length of pText

Return:                    -1 error, >= 0 length of actually read Text (without terminating 0)

Reads from transponder user data in NFC format of type Text.

If user data is not a Text or not in NFC format, return value is -1.

hPort	logical device number
pReadData	Pointer to read buffer
ReadDataLen	Length of read buffer
Return:	-1 error, >= 0 length of actually read data

Reads from transponder its UID (Unique Identifier). A UID has a length of 8 bytes.

```
int TSHRW_Nfc_WriteTag (int hPort, char* pWriteData, int WriteDataLen,
                        BYTE* pKey, int KeyLen, int Lock)
```

hPort	logical device number
pWriteData	Pointer to buffer with JSON string for writing, no terminating 0 necessary
WriteDataLen	Length of string for writing
pKey	Pointer to buffer with key. A key has a length of 16 bytes.
	NULL: ignore key
KeyLen	Length of key (0: ignore key)
Lock	1: Sets the format and the user data of transponder to read only after writing.
	<b>Attention: This cannot be rolled back.</b>
	0: do not set to read only
Return:	-1 error, 0 OK

Writes user data in NFC format in transponder.

If transponder is not NFC formatted yet, the function formats it as NFC.

The user data has to be in JSON format.

Transponder of type DESFire might be write protected with key. In this case the key is needed too.



## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Nfc\_WriteUrl** (int hPort, char\* pUrl, int UrlLen,  
char\* pText, int TextLen,  
BYTE\* pKey, int KeyLen, int Lock)

hPort                      logical device number  
pUrl                        Pointer to buffer with URL for writing, no terminating 0 necessary  
UrlLen                     Length of URL to be written  
pText                       Pointer to buffer with description text for writing,  
no terminating 0 necessary,  
NULL: ignore text  
TextLen                    Length of description text to be written (0: ignore text)  
pKey                        Pointer to buffer with key. A key has a length of 16 bytes.  
NULL: ignore key  
KeyLen                     Length of key (0: ignore key)  
Lock                        1: Sets the format and the user data of transponder to read only after writing.  
**Attention: This cannot be rolled back.**  
0: do not set to read only  
Return:                    -1 error, 0 OK

Writes user data of type URL in NFC format in transponder plus an optional description text (URL: Uniform Resource Locator, e.g. a web address).

If transponder is not NFC formatted yet, the function formats it as NFC.

The user data has to be in JSON format.

Transponder of type DESFire might be write protected with key. In this case the key is needed too.

**int TSHRW\_Nfc\_WriteText** (int hPort, char\* pText, int TextLen,  
BYTE\* pKey, int KeyLen, int Lock)

hPort                      logical device number  
pText                       Pointer to buffer with Text for writing, no terminating 0 necessary  
TextLen                    Length of Text to be written  
pKey                        Pointer to buffer with key. A key has a length of 16 bytes.  
NULL: ignore key  
KeyLen                     Length of key (0: ignore key)  
Lock                        1: Sets the format and the user data of transponder to read only after writing.  
**Attention: This cannot be rolled back.**  
0: do not set to read only  
Return:                    -1 error, 0 OK

Writes NFC formatted text to a transponder.

If transponder is not NFC formatted yet, the function formats it as NFC.

Transponder of type DESFire might be write protected with key. In this case the key is needed too.





## Programming interface (SDK) of the TS-HRW Series

**int TSHRW\_Nfc\_FormatTag** (int hPort, BYTE\* pKey, int KeySize)

hPort                      logical device number  
pKey                        Pointer to buffer with key. A key has a length of 16 bytes.  
                              NULL: ignore key  
KeySize                    Length of key (0: ignore key)  
Return:                    -1 error, 0 OK

Formats the transponder in NFC format.

Transponder of type DESFire can be write protected with key. In this case the key is needed too.

**int TSHRW\_Nfc\_LockTag** (int hPort, BYTE\* pKey, int KeySize)

hPort                      logical device number  
pKey                        Pointer to buffer with key. A key has a length of 16 bytes.  
                              NULL: ignore key  
KeySize                    Length of key (0: ignore key)  
Return:                    -1 error, 0 OK

Sets the format and the user data of transponder to read only.

Transponder of type DESFire can be write protected with key yet. In this case the key is needed too (to secure that all, the user data also, is read only).

**int TSHRW\_Nfc\_SetTagPasswordProtected** (int hPort, int bUserDataToo,  
    BYTE\* pOldKey, int OldKeySize,  
    BYTE\* pNewKey, int NewKeySize)

hPort                      logical device number  
bUserDataToo              != 0: protect user data too,  
                              0: protect only format and settings of transponder  
pOldKey                    Pointer to buffer with key. A key has a length of 16 bytes.  
                              NULL: ignore key  
OldKeySize                Length of key (0: ignore key)  
pNewKey                    Pointer to buffer with new key. A key has a length of 16 bytes.  
NewKeySize                Length of new key  
Return:                    -1 error, 0 OK

Protect the format and the user data of a DESFire transponder by key for unauthorized overwriting.

Transponder of type DESFire can be write protected with key yet. In this case the key is needed too (to secure that all, the user data also, is protected).



## Programming interface (SDK) of the TS-HRW Series

### 7. Commands for transparent mode

Using transparent mode it is possible to access transponders which do not follow to the full standard. You need extensive knowledge about the behavior of the transponders.

**int TSHRW\_TM\_SetProtocol** (int hPort, int Protocol)

hPort                      logical device number  
Protocol                  Used transmission protocol  
                              0: ISO15693  
                              1: ISO1443A  
                              2: ISO14443B  
Return:                    -1 Fehler, 0 OK

**int TSHRW\_TM\_GetProtocol** (int hPort)

hPort                      logical device number  
Return:                    -1 Error,  $\geq 0$  Used transmission protocol, see above

**int TSHRW\_TM\_SetFWTETU** (int hPort, int FwtETU)

hPort                      logical device number  
FwtETU                    FrameWaitTime Dauer in ETU (Elementary Time Units)  
                               $ETU = 128/f_c$ , where  $f_c=13.56\text{MHz}$ , so  $ETU = 9,44 \mu\text{S}$ .  
Return:                    -1 Error, 0 OK

**int TSHRW\_TM\_GetFWTETU** (int hPort)

hPort                      logical device number  
Return:                    -1 Error,  $\geq 0$  FrameWaitTime, see above



## Programming interface (SDK) of the TS-HRW Series

With the following functions the data rate for send and receive is set or retrieved.  
The data rate is given as index values and it depends on the chosen protocol.

ISO15693: TxRate 0: 1 out of 256 (1,65kBit/s)

1: 1 out of 4 (26,48kBit/s)

RxRate 0: Single Subcarrier 6,26 kBit/s,

1: Single Subcarrier 26,48 kBit/s

2: Double Subcarrier 6,67 kBit/s

3: Double Subcarrier, 26,69 kBit/s

ISO14443: TxRate

RxRate 0: 106 kBit/s

1: 212 kBit/s

2: 424 kBit/s

3: 828 kBit/s

**int TSHRW\_TM\_SetDatarate** (int hPort, int TxRate, int RxRate)

hPort                      logical device number

TxRate                     Data rate during send

RxRate                     Data rate during receive

Return :                    -1 Error, 0 OK

**int TSHRW\_TM\_GetDatarate** (int hPort, int \* pTxRate, int \* pRxRate)

hPort                      logical device number

pTxRate                    Pointer to Data rate during send

pRxRate                    Pointer to Data rate during receive

Return:                    -1 Error, 0 OK



## Programming interface (SDK) of the TS-HRW Series

The data transmission in transparent modus is done with the following functions:

**int TSHRW\_TM\_Transmit** (int hPort, BYTE \* pSendBuf, int SendBufLen,  
BYTE \* pRecvBuf, int RecvBufLen)

hPort	logical device number
pSendBuf	data to be sent
SendBufLen	length of data to be sent
pRecvBuf	received data
RecvBufLen	max. length of received data
Return:	-1 Error, $\geq 0$ Length of data in pRecvBuf

**int TSHRW\_TM\_Transmit4** (int hPort, BYTE \* pSendBuf, int SendBufLen,  
BYTE \* pRecvBuf, int RecvBufLen)

hPort	logical device number
pSendBuf	data to be sent
SendBufLen	length of data to be sent
pRecvBuf	received data
RecvBufLen	max. length of received data
Return:	-1 Error, $\geq 0$ Length of data in pRecvBuf

This function is used, if a 4 bit result is expected. (ACK/NAK)

Then the 4 bit result is reported in the 1<sup>st</sup> byte of the receive buffer in the lower 4 bits and return value = 1

**int TSHRW\_TM\_TransmitP** (int hPort, BYTE \* pSendBuf, int SendBufLen,  
BYTE \* pRecvBuf, int RecvBufLen)

hPort	logical device number
pSendBuf	data to be sent
SendBufLen	length of data to be sent
pRecvBuf	received data
RecvBufLen	max. length of received data
Return:	-1 Error, $\geq 0$ Length of data in pRecvBuf

Using TSHRW\_TM\_TransmitP Function, the transmitted and received data includes parity bits. Each data item is represented by two 8 bit values, where the first byte contain the data and second byte contains the parity information in the lowest bit. This is used both at send and receive buffers. Amount is given in bytes always!

The return value is always even if data is sent back because one data value consists of two transmitted bytes. If an ACK/NAK is received, the return value = 1 and only the lowest 4 bits of the first byte contain the ACK/NAK code.



## **8. Error list**

### **8.1. Common errors**

No. (dec.)	Description
1	Error opening the port
2	Timeout-value invalid
3	Port number invalid
4	Timeout
5	given buffer too small
6	buffer size invalid
7	wrong parameter
8	No data for communication
9	No data received
10	Checksum error
11	No communication to device
12	check buffer empty
13	Mode invalid
14	Command not allowed
15	Block number invalid
16	Command error
17	Antenna field is off. Turn on antenna field first using TSHRW_SetRF
19	Block has no value entry
20	Data amount wrong
21	No Transponder (NACK= 15H)
22	Checksum error from device (SYNC= 16H)
23	Collision error
24	Invalid command received, the device cannot execute this command. For example if programmer commands (Chapter 4) are used at TS-HR38 which is only capable of reader mode. (CAN= 18H)
25	Communication error (CRC) at air interface



## Programming interface (SDK) of the TS-HRW Series

### 8.2. Error accessing ISO15693 Transponder

No. (dec.)	Description
32	ISO Error Command not supported
33	ISO Error Command not recognized
34	ISO Error Option not supported
35	ISO Error No information
36	ISO Error Block not available
37	ISO Error Block already locked
38	ISO Error locked block cannot be changed
39	ISO Error Block cannot be written
40	ISO Error Block cannot be locked
41	ISO Error unknown error
42	ISO Error Length invalid
43	ISO Error Unknown transponder
44	ISO Error Framing Error

### 8.3. Error accessing MIFARE® transponder

No. (dec.)	Description
250	MIFARE® Error invalid Value
251	MIFARE® Error Parity Error
252	MIFARE® Error Authenticate Error,
254	MIFARE® Error CRC Error

### 8.4. SDK specific DESFire error codes

No. (hex)	Description
10020	Key length is not 16 bytes
10021	Old key length is not 16 bytes
10022	Crypt method 3K3DES not supported
10023	Command needs authentication
10024	MAC does not match data
10025	CRC after decryption does not match data
10026	Length of parameter buffer is too small for data
10027	Length of parameter field not valid
10028	Not allowed after authentication with AuthenticateISO or AuthenticateAES



## Programming interface (SDK) of the TS-HRW Series

### 8.5. By DESFire card created native errors

The lower 2 digits of the error number are the error number returned by DESFire card.

No. (hex)	Description
1000C	No changes done
1000E	Internal error, insufficient NV-Memory to complete operation
1001C	Command code not supported
1001E	Integrity error, CRC or MAC does not match data / Padding bytes not valid
10040	Invalid Key number specified
1007E	Internal error, length of Command string invalid
1009D	Current config/status does not allow the requested command
1009E	Internal error, value of the parameter invalid
100A0	Requested AID not present on PICC
100A1	Unrecoverable error within application, application will be disabled
100AE	Current authentication status does not allow the requested command
100AF	Internal error, additional data frame is expected to be sent
100BE	Attempt to read/write data from/to beyond the file's limits
100C1	Unrecoverable error in PICC, PICC will be disabled
100CA	Previous command was not fully completed.
100CD	PICC was disabled by an unrecoverable error
100CE	Max. number of applications reached. Number of applications limited to 28
100DE	Duplicate application/file number given
100EE	Internal error, EEPROM could not be written
100F0	File not found
100F1	Unrecoverable error in file

### 8.6. DESFire error codes according to ISO7816-4, generated by card

Lower 4 digits of error number are ISO7816-4 error number

No. (hex)	Description
16282	End of file reached before reading all wished bytes
16700	Internal error, Wrong length
16982	File access not allowed
16985	File empty
16A82	Internal error, wrong parameter P1 or P2
16A86	Internal error, wrong parameter P1 or P2
16C00	File not found
16F00	No precise diagnostics



## Programming interface (SDK) of the TS-HRW Series

### 8.7. Error codes NFC commands

Nr. (hex)	Description
20001	Transponder type is not supported in functions for NFC
20002	This Mifare transponder type is not supported in functions for NFC
20003	Transponder is not of type DESFire
20004	Transponder is not NFC formatted
20005	Authentication of MifareClassic transponder failed
20006	Authentication of DESFire card level failed
20007	Authentication of DESFire NFC application level failed
20008	Error at changing settings
20009	Error at changing key
2000A	Error at locking of blocks in ISO15693 transponder
2000B	Error at formatting transponder as NFC
2000C	NFC in transponder has newer unsupported version as NFC in SDK
2000D	NFC in transponder has older unsupported version as NFC in SDK
2000E	Transponder is write protected
2000F	Transponder is NFC read protected
20010	Transponder is NFC write protected
20011	Error at reading of blocks
20012	Error at writing of blocks
20013	Error at writing in DESFire file
20014	Transponder has not enough free memory for this action
20015	ID of DESFire NFC application not found
20016	Transponder is NFC formatted, but has no user data
20017	Error in NFC structure
20018	Error in user data of type URL
20019	Error in user data of type MIME
2001A	Error in given JSON-String
2001B	Transponder is NFC formatted and has user data of unknown type
2001C	Transponder has NFC user data, but not of type URL
2001D	Transponder has NFC user data, but not of type Text