Technical Handbook

Transponder Reader Family T4/T5

Document Release 2.03
06.04.2009

Content

1. Introduction

This document is the reference guide for the transponder reader families T4 and T5. It contains an functional overview of the readers, the protocol description of the serial communication and the appropriate DLL functions.

2. Serial Communication

2.1 Hardware Level

The serial communication is bidirectional with no handshaking at 9600 baud, 8 data bits, even parity bit and 1 stop bit (9600, 8E1). The host sends request packets to the reader, and the reader answers to each request with a reply packet. The reader sends a reply packet only after receiving a request packet. This rule is violated in autoanswer mode when the reader may send unrequested replies.

2.2 Packet Level

The request packets (sent by the host) have the following format:

1 lenght (=n) total byte count of the request packet

2 address reader address (255 matches any reader)

3 command the request type identifier

4..n-1 data supplementary request specific data

n checksum XOR sum of all bytes from 1 to n-1

The reply packets (sent by the reader after a request is received) have the following format:

1 lenght (=n) total byte count of the reply packet

2 address actual reader address

3 command the request type identifier (see note below)

4 status command completition status byte

5..n-1 data supplemetary data replied by the reader

n checksum XOR sum of all bytes from 1 to n-1

Note: The request type identifier in the reply packet has the same value as the request type identifier in the request packet. The only exception (due to historical reasons) is the 0x1C request. The corresponding reply packet contains then the 0x11 request type identifier. This behaviour has been changed more up-to-date versions of the firmware of the reader.

Unrequested reply packets (in autoanswer mode) have always a 0x11 request type identifier.

Example:

Reading the version information is built up as follows:

Request from the host:

```
0x04 0xFF 0x65 0x9E
```

Reply from the reader:

```
0x0A 0x00 0x65 0x00 0x02 0x3E 0x00 0x81 0x3F 0xED
```

3. Basic Reader Behaviour

3.1 Searching Transponders

The basic and most frequently used request is the read serial number request. The reader reply contains the tag type and serial number of a tag, if any, or the No transponder status code. The tag is hereby selected. Only a selected tag can be read from or written to. The tag remains selected as long as it stays in the reading range of the reader, no read/write errors have occured, and the RF field was not interrupted as a result of a specific request or an automatic shutdown.

If an IDRW-B or IDRW-C tag is removed from the RF field, then the tag loses its selection state without the reader noticing it, and the next read/write request will yield an error.

Tag selection is only meaningful for read/write tags.

3.2 Automatic RF Field Shutdown

For power saving purposes, the RF field is activated only during command execution. If the RF field is off before the execution of a received command or an internally generated read serial number command (in autoanswer mode), the RF field is first activated, and the actual command execution is started after about 200 ms. If the RF field is already on, command execution is commenced immediately.

Two seconds after command completition, the RF field is deactivated. Of course, if another command is executed in the mean time, RF deactivation is delayed for another two seconds. After RF deactivation, any selected tag (with the read serial number command) is deselected.

This behaviour must be considered when executing read and write commands. They imply a previous successfully completed read serial number command. If the subsequent read/write commands follow too late, or the delay between the read/write commands themselves is too long, then a Select error is issued.

4. Windows API

The provided DLLs do contain functions for simplifying the development of application programs under Windows.

Platform	Import Library	DLL	Header File
Windows 95, 98, ME, NT, 2000, XP, Vista	T4W.LIB	T4W.DLL	T4W.H
Windows CE	T4CE.LIB	T4CE.DLL	T4W.H

5. Compatibility and Differences

Here is an overview of transponder types and functions, which can be used in conjunction with different types of transponder readers:

	Reader family T4 **AHL810, AHL330, AHL410, AHL170, AHL570, AHL575, AXA050, AXA200, AXA250, AXA500, AXA550,** **TWN3 Multi125**	**Reader family T5** **AHL821, AXA033**
Transponder Types		
Transponder type IDRO-A (4102)	Supported	Supported
Transponder type IDRW-B (Hitag 1, Hitag S)	Supported (W/O crypto)	Supported
Transponder type IDRW-C (Hitag 2)	Supported (W/O crypto)	Supported
Transponder type IDRW-D (4150)	Supported (f/64)	Supported (f/64)
Transponder type IDRW-E (Q5, 5555, 5557)	Supported (Manchester f/64)	Not supported
Transponder type IDRO-G (ISO FDX-B)	Supported	Supported by firmware V1.11 or later
Transponder type IDRO-H (4026)	On request	Supported (f/32)
DLL-Specific Functions		
T4FindCOMPort, T4FindCOMPortNr T4FindFirstCOMPortNr T4FindNextCOMPortNr	Only USB and CF card readers can be detected by these functions	
Reader Requests		
I/O Requests	Available, but do only show an effect for AXA050, AXA200, AXA250, AXA500, AXA550, AHL575	Available, but do only show an effect for AXA033
Beep Request	Available, but do only show an effect, if a beeper is connected to the appropriate output	Not supported
Crypto Functions for Hitag 1, Hitag 2 and Hitag S	Not supported	Supported by AHL821 and AXA033 with crypto processor

6. Functions

This is the list of functions which are available via the API-DLL. Please note, that many functions are handled by the DLL internally without any communication to the reader. Where applicable, the appropriate command code and stucture of data of the serial communication is shown.

6.1 Retrieving Port Numbers

For USB and CF card readers, special functions do exists, which are able to find the connected transponder readers without any communication. This saves time during the initialization process of an application. Following functions are available.

	Windows 95/98/ME	Windows 2000/XP	Windows CE/Mobile
T4FindCOMPort[1)]	Not supported	Supported	Supported (CF only)
T4FindFirstCOMPortNr	Not supported	Supported	N/A
T4FindNextCOMPortNr	Not supported	Supported	N/A
T4FindCOMPortNr	Not supported	Supported	Supported (CF only)
T4GetCOMPortString	Supported	Supported	Supported

[1)] Preferred function for single-reader applications.

`LPCTSTR T4FindCOMPort(void)`

This function is used for retrieving the port name of a CF-card- or USB-reader.

Parameter: None.

Return: If a transponder reader has been found, a pointer is returned which points to the name of the COM port. If no reader has been found the function returns a null pointer.

Attention:
Also see table “Error Codes of the Library”, especially error code `T4_NOTINIT`.

`int T4FindFirstCOMPortNr(int PortTypes)`

This function is used in conjunction with `T4FindNextCOMPortNr` in order to retrieve the port numbers of all installed CF-card- and USB readers.

Parameter:

PortTypes: Logical or of the definitions `T4_MSK_USB` and `T4_MSK_CF`, which represent the types of readers (and the type of their communication interface) to be retrieved.

Return: If a transponder reader has found, an integer value is returned which reflects the number of the COM port. If no transponder reader has been found the function returns zero.

Example:

```
// Find all installed USB readers
int PortNr;
int Count = 0;
PortNr = T4FindFirstCOMPortNr(T4_MSK_USB);
while (PortNr != 0)
{
    Count++;
    printf("USB reader found at COM%d\n",PortNr);
    PortNr = T4FindNextCOMPortNr();
}
printf("%d USB readers have been found.\n",Count);
```

int T4FindNextCOMPortNr(void)

This function is used in conjunction with `T4FindFirstCOMPortNr` in order to retrieve the port numbers of all installed CF-card- and USB readers. If T4FindFirstCOMPortNr already returned none-zero, further calls to this function do return all remaining COM ports.

Parameter: None.

Return: If a transponder reader has found, an integer value is returned which reflects the number of the COM port. If no transponder reader has been found the function returns zero.

int T4FindCOMPortNr(void)

This function is used for retrieving the port number of a CF-card- or USB-reader.

Parameter: None.

Return: If a transponder reader has found, an integer value is returned which reflects the number of the COM port. If no transponder reader has been found the function returns zero.

Attention:

Also see table "Error Codes of the Library", especially error code `T4_NOTINIT`.

LPCTSTR T4GetCOMPortString(int PortNr)

This function calculates the string of a port name out of a given port number.

Parameter:

`PortNr` The port number (e.g. 1 for COM1:)

Return: A pointer to a null terminated string which reflects the port name of the given port number. E.g. a given 1 will be converted into "COM1:".

6.2 Initialize/Deinitialize Communication

int T4Init(LPCTSTR Port)

Initializes the given serial port for communication with the transponder reader and performs a short test communication to the transponder reader.

Windows CE please note:

See

Parameter:

`Port`	Points to the null terminated string (wide characters under Windows CE!) of the desired communication port, i.e. "`COM2:` "

Return: Error code. See table error codes.

int T4DeInit(void)

Deinitialize the communication with the transponder reader.

Parameter: None.

Return: Always delivers `T4_NOERROR`.

HANDLE T4GetCurrentReader(void)

In order to communicate with more then one transponder reader at a time, it is possible to retrieve the handle of the currently selected reader. If more then one reader should be operated within one application, this function should be called. A new handle is created with the function `T4Init`. It can be retrieved, if a preceding call to `T4Init` returned an error code different to `T4_V24_INITFAIL`.

Parameter: None.

Return: The handle, which represents the communication environment for the currently selected transponder reader.

void T4SetCurrentReader(HANDLE Handle)

This function selects the communication environment of an already initialized transponder reader.

Parameter:

`Handle`	The handle, which represents the communication environment for the transponder reader to be selected.

Return: None.

Example:

```
int Error;
HANDLE hCOM1;
HANDLE hCOM2;

Error = T4Init("COM1:");
if (Error == T4_V24_INITFAIL)
{
    // Unable to open COM port
    return;
}
if (Error != T4_NOERROR)
{
    // Some other error
    T4DeInit();
    return;
}
hCOM1 = T4GetCurrentReader();

Error = T4Init("COM2:");
if (Error == T4_V24_INITFAIL)
{
    // Unable to open COM port
    return;
}
if (Error != T4_NOERROR)
{
    // Some other error
    T4DeInit();
    return;
}
hCOM2 = T4GetCurrentReader();

// Init reader at COM1
T4SetCurrentReader(hCOM1);
Error = SetTagDriver(T4_MSK_IDRO_A | T4_MSK_IDRW_D);

// Init reader at COM2
T4SetCurrentReader(hCOM1);
Error = SetTagDriver(T4_MSK_IDRO_A | T4_MSK_IDRW_D);
```

HANDLE T4GetHandle(void)

Retrieve the handle of the communication stream which has been opened via the Windows system call `CreateFile` by the DLL.

Parameter: None.

Return: The handle for the communication stream which has been opened via the Windows system call `CreateFile` by the DLL. If the communication is not initialized `INVALID_HANDLE_VALUE` is returned.

6.3 Functions for Selecting a Transponder

These functions are performing three actions:

1. Searching for a transponder
2. Logging into the transponder if necessary
3. Retrieving the unique ID of a transponder

The length of the unique ID depends on the type of transponder:

Transponder Type	Length of ID [Bytes]	Number of bytes returned	Index of first valid byte
IDRO-A	5	5	0
IDRW-B	4	5	1
IDRW-C	4	5	1
IDRW-D	4	5	1
IDRW-E	4[1)]	5	1
IDRO-G	8	8	0
IDRO-H	8	8	0

[1)] this is not a unique ID, but a data word configured by the user.

This can also be shown as:

Transponder Type	0	1	2	3	4	5	6	7
IDRO-A								
IDRW-B	0x00							
IDRW-C	0x00							
IDRW-D	0x00							
IDRW-E	0x00							
IDRO-G								
IDRO-H								

6.3.1 Blocking Functions for Selecting a Transponder

These functions are performing a complete search for a transponder and are returning, after the operation has been completed.

`int T4SelectTag(BYTE *TagType,BYTE *SerialNumber)`

This functions performs three tasks:

1. If a transponder is in reach of the transponder reader, the type of the transponder is determined.
2. If the transponder can be reached, it will be selected. That means it is prepared for reading and writing. If the transponder needs a password, the transponder reader will try to log into the transponder by using the password that is stored in the internal memory of the transponder reader (by default this is 00 00 00 00).
3. If the selection of the transponder was successful, the serial number will be delivered.

Parameter:

`TagType`	Delivers the type of the selected tag.
`SerialNumber`	Points to a memory space (5-8 bytes, depending on the activated transponders) where the serial number will be stored.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x11	None

Reply
Data
N Bytes Byte 0 = TagType, Byte 1..N-1 = SerialNumber (5-8 Bytes)

int T4SelectTagPW(BYTE *PassWord,BYTE *TagType, BYTE *SerialNumber)

This functions performs three tasks:

1. If a transponder is in reach of the transponder reader, the type of the transponder is determined.
2. If the transponder can be reached, it will be selected. That means it is prepared for reading and writing. If the transponder needs a password, the transponder reader will try to log into the transponder by using the given password.
3. If the selection of the transponder was successful, the serial number will be delivered.

Parameter:

`PassWord`	Pointer to the 4-byte password. If the pointer is set to NULL, the internal password of the reader (by default this is 00 00 00 00) will be used.
`TagType`	Delivers the type of the selected tag.
`SerialNumber`	Points to a memory space (5-8 bytes, depending on the activated transponders) where the serial number will be stored.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x1C	4 Bytes Password

Reply
Data
N Bytes Byte 0 = TagType, Byte 1..N-1 = SerialNumber (5-8 Bytes)

int T4SelectTagType(BYTE ReqTagType, BYTE *TagType, BYTE *SerialNumber)

This function is valid only for tag type IDRW-C. It is required because a IDRW-C may emulate a transponder of type IDRO-A. This function provides a functionality for selecting the IDRW-C in it's native mode.

Parameter:

`ReqTagType`	This parameter must be set to `T4_TYPE_IDRW_C`.
`TagType`	Delivers the type of the selected tag.
`SerialNumber`	Points to a memory space (5-8 bytes, depending on the activated transponders) where the serial number will be stored.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x1B	1 Byte = 0x02

Reply
Data
N Bytes Byte 0 = TagType, Byte 1..N-1 = SerialNumber (5-8 Bytes)

`int T4Halt(void)`

This function is used in multi-tag environments with IDRW-B. In order to handle more then one tag, following typical sequence is necessary:

1. Select a transponder via one of the select functions (see above).
2. Do all necessary actions with the selected transponder, i.e. read or write operations.
3. Stop communication with this transponder by calling `T4Halt`. This will prevent the transponder from participating from the further arbitration process.
4. Continue with 1. till no further transponder is found.

Parameter: None.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x1A	None

Reply
Data
None

`int T4RFReset(void)`

This function will turn off the RF field for 50ms. This will cause any transponder in the field to restart operation.

Parameter: None.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x69	None

Reply
Data
None

6.3.2 None-Blocking Functions for Selecting a Transponder

This set of functions lets a application search for transponders in the background without consuming CPU time for a longer period.

`void T4InitTagPoll(BYTE *PassWord)`

This function prepares the usage of the functions `T4DoTagPoll` (see below).

Parameter:

`PassWord`	Pointer to the 4-byte password. If the pointer is set to NULL, the internal password of the reader (by default this is 00 00 00 00) will be used.

Return: None.

`int T4DoTagPoll(BYTE *TagType,BYTE *SerialNumber)`

This function is the none blocking version of `T4SelectTag` and `T4SelectTagPW`. Before calling this function the function `T4InitTagPoll` must be called. This function is the best choice for cyclic searching for a transponder e.g. in the timer event of a Windows program. A typical value for the repeat time is 50 milliseconds.

Parameter:

`TagType`	Delivers the type of the selected tag.
`SerialNumber`	Points to a memory space (5-8 bytes, depending on the activated transponders) where the serial number will be stored.

Return: Error code. See table error codes.

`int T4DeInitTagPoll(void)`

This function ends any pending poll cycle of the poll command. It is not necessary to call this function before calls to other functions, because it is called automatically internal by the DLL if necessary.

Parameter: None.

Return: Error code. See table error codes.

int T4GetTagPollState(void)

This function retrievs the current state of a poll cycle.

Parameter: None.

Return:

T4_POLLNOTINIT	Tag poll is not initialized.
T4_POLLIDLE	Currently, a tag poll is active
T4_POLLBUSY	Currently, a command is executed

6.3.3 None-Blocking Functions for Auto-Answer Mode

This set of functions lets a application search for transponders in the background without consuming CPU time for a longer period. Furthermore, these functions do require the auto-answer mode of the transponder reader to be turned on.

int T4InitAAMPoll(void)

This function starts cyclic polling for an auto-answer packet from the reader.

Parameter: None.

Return: Error code. See table error codes.

int T4DoAAMPoll(BYTE *TagType,BYTE *SerialNumber)

This function returns the received transponder type and serial number if a transponder has been presented to the RF field.

Parameter:

TagType	Delivers the type of the selected tag.
SerialNumber	Points to a memory space (5-8 bytes, depending on the activated transponders) where the serial number will be stored.

Return: Error code. See table error codes.

int T4DeInitAAMPoll(void)

Parameter: None.

Return: Error code. See table error codes.

6.4 Functions for Reading and Writing of Transponders

Reading and writing from/to the transponder is always performed in multiples of blocks that consist of four bytes.

T4WriteData(BYTE BlockAddress,BYTE BlockCount,BYTE *Data)

Writes data blocks (= 4 bytes = 32 bit) to the transponder. Before writing the transponder must be selected.

Parameter:

BlockAddress	Address of the first written block.
BlockCount	Count of blocks to be written. Valid values are from 1 to 4.
Data	Pointer to the written data.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x14	1 + (4 x BlockCount) Bytes, Byte 0 = BlockAddress, Remaining Bytes = Data The count of blocks is determined by the length of the telegram

Reply
Data
None

int T4ReadData(BYTE BlockAddress,BYTE BlockCount,BYTE *Data)

Reads data blocks (= 4 bytes = 32 bit) from the transponder. Before reading the transponder must be selected.

Parameter:

BlockAddress	Address of the first read block.
BlockCount	Count of blocks to be read. Valid values are from 1 to 4.
Data	Pointer to the read data.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x15	2 Bytes Byte 0 = BlockAddress, Byte 1 = BlockCount

Reply
Data
(4 x BlockCount) Bytes of Data

6.5 Functions for Configuring Transponders

`int T4WriteConfigData(BYTE ConfigAddress,BYTE *ConfigData)`

Writes a configuration block to the transponder. The transponder must be selected first.

Note:

This functions should only be called, if you have studied the corresponding datasheet of the transponder.

Parameter:

`ConfigAddress`	Configuration address of the transponder.
`ConfigData`	Pointer to the new configuration data (4 bytes).

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x16	5 Bytes, Byte 0 = ConfigAddress Byte 1..4 = ConfigData

Reply
Data
5 Bytes

`int T4ReadConfigData(BYTE ConfigAddress,BYTE *ConfigData)`

Reads a configuration block from the transponder. The transponder must be first selected.

Parameter:

`ConfigAddress`	Configuration address of the transponder.
`ConfigData`	Pointer to the data space where the configuration block will be stored. Please note: five bytes are returned. The first byte contains the transponder type and the remaining 4 bytes contain the data word itself.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x17	1 Byte = ConfigAddress

Reply
Data
5 Bytes, Byte 0 = TagType, Byte 1..4 = ConfigData

6.6 Configuring Support for Transponders

int T4SetTagDriver(BYTE Driver)

This function configures the support for the different transponder types.

Parameter:

`Driver`	Or combination of the desired transponder types. Please note the `T4_MSK_IDRW_E` can only be used alone.

Return: Error code. See table error codes.

Example:

```
SetTagDriver(T4_MSK_IDRO_A | T4_MSK_IDRW_D);
```

Serial Communication:

Request	
Command	**Data**
0x80	2 Bytes, Byte 0 = 0x00, Byte 1 = Driver

Reply
Data
None

int T4SetTagOptions(BYTE TagType,BYTE TagOptions)

With this function the transponder reader behavior for a given tag is configured.

Note:

Only behavior for transponder type IDRW-E can be configured.

Parameter:

`TagType`	Transponder type
`TagOptions`	Bit combination which defines the new behavior.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0xA5	2 Bytes, Byte 0 = TagType, Byte 1 = TagOptions

Reply
Data
None

6.6.1 Options for Transponder Type IDRW-B

For the transponder type IDRW-B there is one possible value for parameter `TagType`.

6.6.1.1 TagType = 0x01 (=T4_TYPE_IDRW_B)

The configuration is stored in the internal EEPROM of the reader and is defined as follows:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved	reserved	reserved	reserved	reserved	reserved	HF-Reset	Anticol.

Bit 0:

0: Standard mode. Select an IDRW-B without arbitration. If more then one transponder is in the HF field, the transponder with the highest signal will be selected.

1: Anti-collision mode. A transponder is selected via the arbitration function of IDRW-B.

Bit 1:

0: Standard mode, 1: Reset HF before selecting IDRW-B

This option is necessary to gain access to IDRW-B, which are running in IDRO-A emulation mode. It is not possible to use anti-collision together with this function.

Note:

In order to use multi-tag function, also see function `T4Halt`.

Bit 2-7:

Reserved for future use. Please keep this bits reset to 0.

6.6.2 Options for Transponder Type IDRW-E

For the transponder type IDRW-E there are two possible values for parameter `TagType`.

6.6.2.1 TagType = 0x04 (=T4_TYPE_IDRW_E)

The configuration is stored in the internal EEPROM of the reader and is defined as follows:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RepCnt3	RepCnt2	RepCnt1	RepCnt0	Conf/Dat	SelAdr2	SelAdr1	SelAdr0

Bit 0-2:

Address that will be read by selecting the transponder.

Bit 3:

0: a data word is read, 1: a config word is read

Bit 4-7:

Repeat count. Therefore that IDRW-E as no serial number by default, the reader can be configured to read a given word many times. If the word is always the same, the transponder reader will return a successful read.

Notes:

If repeat count is set to 0, the select command will always return a successful read even if there is no transponder in the field. The serial number will always be 00 00 00 00. This value could be used if you are determining if a transponder is in the field in a second step by reading a memory address of the transponder.

If repeat count is set to 1, the select command will always return a successful read even if there is no transponder in the field. The serial number will be from noise to a good read depending on the distance of the transponder. So this value should not be used.

So a good choice for RepeatCount is 3 or above to get more safety in detecting a transponder.

6.6.2.2 TagType = 0x84 (=T4_TYPE_IDRW_E | 0x80)

For safety reasons this configuration byte is stored only in the internal RAM of the reader and will be reset to its default value at next startup of the reader:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Res.	res.	res.	res.	res.	res.	Lock	Use PW

Bit 0:

0: password mode off, 1: password mode on.

Bit 1:

0: written data won't be locked, 1: written data will be locked

6.6.3 Configuring Reader Internal Password

`int T4ChangeReaderPW(BYTE *OldPW,BYTE *NewPW)`

With this function the reader internal password can be changed. By default, the internal password of the reader is 0x00 0x00 0x00 0x00. This password is used for selecting tags of type IDRW-C, IDRW-D and IDRW-E.

Parameter:

`OldPW`	Pointer to four bytes of data, which contain the old password.
`NewPW`	Pointer to four bytes of data, which contain the new password.

Return: Error code. See table error codes.

Serial Communication:

Changing the reader internal password is a two-step process. First, the reader has to prepared for the password change with the old/current password.

Request	
Command	**Data**
0xA3	4 Bytes, Old/Current Password

Reply
Data
None

In the second step the new password has to be transferred:

Request	
Command	**Data**
0xA4	4 Bytes, New Password

Reply
Data
None

6.7 Cryptographic Functions

Versions of the reader family T5 do contain an additional crypto processor, which enable the support for the Hitag specific authentication and crypto-functions.

6.7.1 Setting up Crypto Keys

Note:

Please keep track of the crypto keys that have been configured. In order to change a crypto key, both the current configured, and the new key must be specified!

`int T4Personalisation12(const BYTE *Data)`

This function is used to write the keys and passwords for Hitag 1 and Hitag 2 into the built-in crypto processor of an appropriate transponder reader.

Note:

There is no way to reset the personalization information!

Parameter:

`Data` — Points to an array of 72 bytes which do contain both the old and the new personalization information. See below.

Return: — If the function succeeds, the return code is `T4_NOERROR`. If the old information has been wrong, the return code is `T4_READERCHPWFAIL`.

Serial Communication:

Request	
Command	**Data**
0xA6	72 Bytes of Data

Reply
Data
None

Here is an example of a structure of the personalization information for Hitag 1 and Hitag 2. It sets the crypto processor from the initial state to factory default personalization information of Hitag 1 / Hitag 2. This is also the default delivery status of the transponder reader.

```
const BYTE PersData[72] =
{
    // Old Key A             New Key A
    0xFF,0xFF,0xFF,0xFF,     0x00,0x00,0x00,0x00,
    // Old Key B             New Key B
    0xFF,0xFF,0xFF,0xFF,     0x00,0x00,0x00,0x00,
    // Old Logdata 0A        New Logdata 0A
    0xFF,0xFF,0xFF,0xFF,     0x00,0x00,0x00,0x00,
    // Old Logdata 0B        New Logdata 0B
    0xFF,0xFF,0xFF,0xFF,     0x00,0x00,0x00,0x00,
    // Old Logdata 1A        New Logdata 1A
    0xFF,0xFF,0xFF,0xFF,     0x00,0x00,0x00,0x00,
    // Old Logdata 1B        New Logdata 1B
    0xFF,0xFF,0xFF,0xFF,     0x00,0x00,0x00,0x00,
    // Old Hitag 2 Key 16    New Hitag 2 Key 16
    0xFF,0xFF,0xFF,0xFF,     0x00,0x00,0x4F,0x4E,
    // Old Hitag 2 Key 32    New Hitag 2 Key 32
    0xFF,0xFF,0xFF,0xFF,     0x4D,0x49,0x4B,0x52,
    // Old Hitag 2 Password  New Hitag 2 Password
    0xFF,0xFF,0xFF,0xFF,     0x00,0xAA,0x48,0x54
};
```

int T4PersonalisationS(const BYTE *Data)

This function is used to write the Keys and Passwords for Hitag S into the built-in crypto processor of the transponder reader. The default personalization information is entire 0xFF.

Note:

There is no way to reset the personalization information!

Parameter:

`Data`	Points to an array of 24 bytes which do contain both the old and the new personalization information. See below.
Return:	If the function succeeds, the return code is `T4_NOERROR`. If the old information has been wrong, the return code is `T4_READERCHPWFAIL`.

Serial Communication:

Request	
Command	**Data**
0xA7	24 Bytes of Data

Reply
Data
None

Here is an example of a structure of the personalization information for Hitag S. This is also the default delivery status of the transponder reader.

```
const BYTE PersData[24] =
{
    // Old Key 16            New Key 16
    0xFF,0xFF,0xFF,0xFF,     0x00,0x00,0x4F,0x4E,
    // Old Key 32            New Key 32
    0xFF,0xFF,0xFF,0xFF,     0x4D,0x49,0x4B,0x52,
    // Old Password          New Password
    0xFF,0xFF,0xFF,0xFF,     0x00,0xAA,0x48,0x54
};
```

6.7.2 Configuring Crypto Functions for Hitag 1 and Hitag S

Due to the fact that Hitag 1 and Hitag S are very similar in the selection process, they are treated both as IDRW-B in the T4 API. During the selection of Hitag 1 or Hitag S the transponder reader is distinguishing between the two types by analyzing the serial number of the transponder. In the further process the appropriate crypto sequence for the specific transponder is used. Following behavior can be configured:

Hitag 1:	**Definition**
Do not use authentication	`HTG1_AUTH_NONE`
Authentication with KEY A	`HTG1_AUTH_KEYA`
Authentication with KEY B	`HTG1_AUTH_KEYB`

Hitag S:	**Definition**
Do not use authentication	`HTGS_AUTH_NONE`
Authentication without password	`HTGS_AUTH_STD`
Authentication with password	`HTGS_AUTH_PW`
Automatic selection with or without authentication	`HTGS_AUTH_NONESTD`
Automatic selection with or without authentication and password	`HTGS_AUTH_NONEPW`

The behavior for both transponders is configured with a single function call. Example:

Let the transponder reader select a Hitag 1 via KEY A and Hitag S without authentication:

```
Error = T4SetTagOptions(0x80 | T4_TYPE_IDRW_B,
                        HTG1_AUTH_KEYA | HTGS_AUTH_NONE);
```

The 0x80 tells the reader that we are configuring temporary options which are not stored in the internal EEPROM. T4_TYPE_IDRW_B selects the options for Hitag 1 and Hitag S.

6.7.3 Configuring Crypto Functions for Hitag 2

For Hitag 2 the options for the cryptographic functions are as follows:

Hitag 2:	**Definition**
Do not use crypto mode	`HTG2_AUTH_PW`
Crypto mode without password check	`HTG2_AUTH_CRYPT`
Crypto mode with password check	`HTG2_AUTH_CRYPT_PW`

Example:

Let the transponder reader select Hitag 2 in crypto mode with password check:

```
Error = T4SetTagOptions(0x80 | T4_TYPE_IDRW_C,
                        HTG2_AUTH_CRYPT_PW);
```

6.8 Configuring Auto-Answer Mode

`int T4AAMSetConfig(BYTE AAMMode)`

Configure the behaviour of the auto-answer mode. If a tag is detected and reported, the search ceases for the repeat period time. Every autoanswer reply has the reply format of the 0x11 request. Replies are only sent, if a transponder have been detected.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable	Trigger	Edge/ Level	res.	res.	IO Select 2	IO Select 1	IO Select 0

Bit 0-2:

Input selection for trigger

000: line IO1

001: line IO2

010: line IO3

011: line IO4

100: line IO5

other values are invalid

Bit 5:

0: Auto-answer is active whenever the logic state of the selected input is true (high,'1') and read serial number operations are then tried until a tag is detected.

1: Auto-answer is triggered by a rising edge of the logic level of the selected input line and one single tag read serial number operation is tried.

Bit 6:

0: Auto-answer mode is permanently activated

1: Auto-answer mode is triggered by external events

Bit 7:

If '1', autoanswer mode is enabled, else auto-answer mode is disabled

Parameter:

`AAMMode` Configuration byte as specified above.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0xAC	1 Byte, AAMMode

Reply
Data
None

`int LIBFUNC T4AAMSetRepeatTime(WORD RepeatTime)`

Configure the repeat time of the auto-answer mode.

Parameter:

`RepeatTime` Repeat time which is specified in multiples of 5 milliseconds.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0xAD	2 Bytes, Byte 0 = RepeatTime MSB, Byte 1 = RepeatTime LSB

Reply
Data
None

6.9 Configuring Digital I/Os

`int T4IOSetConfig(BYTE IODirection,BYTE IOLevel)`

The data direction for the I/O lines of the reader is freely programmable. A ’1’ bit on the corresponding position of IODirection makes that signal line an output. A ’0’ bit makes that line an input. The correspondence is given in the following table:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
res.	res.	res.	IO5	IO4	IO3	IO2	IO1

The physical level of each I/O can be programmed to be the negated logical value if the corresponding bit in IOLevel is set. Both inputs and outputs are affected by this setting.

Parameter:

`IODirection` A byte which contains the input/output configuration for the signal lines. A ‘1’ makes the line an output, a ‘0’ makes the line an input.

`IOLevel` A byte which contains the logic polarity of the signal lines. A ‘1’ makes the I/O visible as a active low port, a ‘0’ will make it an active high port.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0xA8	2 Bytes, Byte 0 = IODirection, Byte 1 = IOLevel

Reply
Data
None

`int T4IOSetDefaults(BYTE IODefaultValue)`

IODefaultValue specifies the logical value of the output lines after a reset.

Parameter:

`IODefaultValue` A byte which contains the logical value of the outputs after a reset.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0xA9	1 Byte, IODefaultValue

Reply
Data
None

6.10 I/O Requests

`int T4IOReadInputs(BYTE *InputState)`

This request reads the status of the digital inputs. The logical state of the ports is reported, that is, the physical state of the inputs is XORed with the given I/O level

`InputState` Pointer to a byte which contains the state of inputs after calling the function.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0xB0	None

Reply
Data
1 Byte = InputState

int T4IOSetOutputs(BYTE SetBits)

This request forces a logic ‘1’ state to the outputs corresponding to the ’1’ bits in SetBits. The outputs corresponding to ’0’ bits in SetBits remain unchanged.

Parameter:

SetBits	A byte which contains the bits of the output ports being set (= set to active state).

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0xB1	1 Byte = SetBits

Reply
Data
None

T4IOClearOutputs(BYTE ClearBits)

This request forces a logic ‘0’ state to the outputs corresponding to the ’1’ bits in ClearBits. The outputs corresponding to ’0’ bits in ClearBits remain unchanged.

Parameter:

ClearBits	A byte which contains the bits of the output ports being cleared (= set to inactive state).

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0xB2	1 Byte = ClearBits

Reply
Data
None

int T4IOBeep(BYTE BeepBits,BYTE Duration,BYTE Frequency)

This request causes the signal on the outputs corresponding to the '1' bits in beep-bits to toggle with the specified frequency for the specified duration. The duration is given in 20 milliseconds increments. The frequency is given in approx. 30 Hz increments. The reply is sent only after request completion. Then the toggled outputs assume their previous state.

Parameter:

BeepBits	A byte which contains the output ports which will sound a beep.
Duration	The duration of the beep in multiples of 20 milliseconds.
Frequency	The frequency of the beep in approx. 30 Hz increments.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0xB3	3 Bytes: Byte 0 = BeepBits, Byte 1 = Duration Byte 2 = Frequency

Reply
Data
None

6.11 Reading Version Information

int T4VersionTagReader(BYTE *Version)

Delivers the version of the connected transponder reader.

Parameter:

Version	Points to a data space of five bytes which do contain the version information: Byte 1: Major number of the version Byte 2: Minor number of the version Byte 3: reserved Byte 4: Device type (T4 = 0x81, T5 = 0x87) Byte 5: Or combination of supported transponders

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x65	None

Reply
Data
5 Bytes

int T4VersionLibrary(void)

Delivers the version of the linked library.

Parameter: None.

Return: Integer which contains the version number, i.e. 102 for version 1.02.

6.12 Miscellaneous

int T4ConfigurationReset(void)

Resets the configuration of the transponder reader to the original factory settings.

Parameter: None.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x64	None

Reply
Data
None

int T4CPUReset(void)

This function will reset the CPU of the transponder reader like a power-up.

Parameter: None.

Return: Error code. See table error codes.

Serial Communication:

Request	
Command	**Data**
0x63	None

Reply
Data
None

const char *T4GetStatusString(int Status)

Returns a clear text version of the given status code.

Parameter:

`Status` Status code to convert

Return: Pointer to the clear text status string.

7. Definitions

7.1 General

7.1.1 Version of the Library

T4_VERSIONLIBRARY	Version of the header file this version should match the version of the linked library.

7.1.2 Type of Transponders

The following definitions correspond to the returned transponder types of the select commands:

T4_TYPE_IDRO_A	Read-Only-Tag IDRO-A.
T4_TYPE_IDRW_B	Read-Write-Tag IDRW-B.
T4_TYPE_IDRW_C	Read-Write-Tag IDRW-C.
T4_TYPE_IDRW_D	Read-Write-Tag IDRW-D.
T4_TYPE_IDRW_E	Read-Write-Tag IDRW-E.
T4_TYPE_IDRO_G	Read-Only-Tag IDRO-G.
T4_TYPE_IDRO_H	Read-Only-Tag IDRO-H.

7.1.3 Codes to Support the Different Transponder Types

The following definitions are used in combination with the function T4SetTagDriver:

T4_MSK_IDRO_A	Read-Only-Tag IDRO-A.
T4_MSK_IDRW_B	Read-Write-Tag IDRW-B.
T4_MSK_IDRW_C	Read-Write-Tag IDRW-C.
T4_MSK_IDRW_D	Read-Write-Tag IDRW-D.
T4_MSK_IDRW_E	Read-Write-Tag IDRW-E.
T4_MSK_IDRO_G	Read-Only-Tag IDRO-G.
T4_MSK_IDRO_H	Read-Only-Tag IDRO-H.

7.1.4 Port Types

The following definitions are used in combination with the function T4FindFirstCOMPortNr:

T4_MSK_USB	Find USB readers.
T4_MSK_CF	Find CF card readers.

7.2 Error Codes

The following error codes are returned by functions of T4xx.LIB:

7.2.1 General

T4_NOERROR	The function has been executed successful.

7.2.2 Error Codes from Transponder Operations

T4_NOTAG	There is no transponder in the reading range of the transponder reader.
T4_DATAERROR	Erroneous data. Maybe the distance between reader and transponder is to large or the communication has been disturbed in another way.
T4_WRITEERROR	Error during write to a transponder. Maybe the transponder is write-protected.
T4_ADRESSERROR	The specified address is out of the valid range.
T4_WRONGTAG	It has been tried to start a operation that is not supported by the transponder.
T4_READERROR	Error during a read operation of the transponder.
T4_NOTSELECTED	There was read or write access without a preceding select of the transponder.
T4_WRONGTAGPW	Transponder reader failed to login to the transponder. The password is wrong.

7.2.3 Error Codes from Reader Operations

T4_PARAERRWRITE	Configuration parameters could not be saved into EEPROM.
T4_PARANOTVALID	Configuration parameter is invalid.
T4_PARAERRPWCYCLE	During change of internal reader password an error occurred.
T4_PARANOLOGIN	It has been tried to execute a reader command that needs a login to the reader.
T4_PARAWRONGPW	It has been tried to execute a reader command that needs a login to the reader but the password was wrong.
T4_READERCHPWFAIL	Reader password could not be changed.

7.2.4 Error Codes from the Serial Connection

T4_RX_TIMEOUT	During receiving of a telegram a timeout occurred.
T4_RX_WRONGANSWER	During receiving of a telegram there was received a wrong character.
T4_RX_WRONGCSUM	The checksum of a received telegram was wrong.
T4_RX_PARITY	A parity error occurred.
T4_RX_OVERRUN	An overrun error occurred.
T4_RX_FRAME	A frame error occurred.

7.2.5 Error Codes of the Library

T4_RX_WRONGLENGTH	The length of a telegram was wrong.
T4_WRONGBLOCKCNT	The count of blocks within a telegram was too small or too big.
T4_POLLNOTINIT	Try to start the function T4DoTagPoll without a preceding call to T4InitTagPoll.
T4_ALREADYINIT	Another call to T4Init, but the library was already initialized.
T4_NOTINIT	A function call to the library occurred but the library was not (or no more) initialized. **Please note:** **This error code will also be returned if a CF card reader or USB reader is removed while the DLL is initialized.**
T4_V24_INITFAIL	The initialization of the library via T4Init failed. The given COM port was not available.