Programming manual for the PHL-2700 with the 13,56MHz RFID-module

Introduction

This is a preliminary manual giving information about the functions in rfidlib.lib that give access to the RFID capability of the PHL-2700 with the 13,56MHz RFID module.

How to build an application using the library

To build an application, see the programming manual for the PHL-2700. The functions of the RFID library can be linked by adding the following line to the link file, e.g. in front of the line “LOAD c:\mccm77\CM77ISLC.LIB”

```
LOAD RFIDLIB.LIB
```

In all the source files that make use of the functions of the RF library, include the file “`rfidlib.h`”.

Note: use operating system CBWV0126 or up. This operating system implements a virtual COM port, COM6, that features special support for the RFID library.

Note: use firmware version HAAV006 or up.

The functions in the RFID library are described in the following pages.
They are grouped in three sections:

- General functions
- Tag-it command functions
- I-Code command functions
- ISO15693 command functions

In the appendixes general information about these RFID tag families is given.

General functions

rfidopen

Description Switches the RFID module on and establishes communication with it.

Syntax
```
int rfidopen(void)
```

Arguments -

Returns

-	on success	0
-	on error (no communication)	-1
-	on error (cannot open port)	-2

Remarks
- If the RFID device is already open, the function does nothing and returns 0.
- A return value of -2 means that COM6 cannot be opened. Upgrade the OS to a version that does support COM6.
- To indicate “RFID reader active” it is suggested to call `goodreadled()` setting the red or the green LED on, as long as the RFID device is open.
- Do not leave the RFID device open longer than necessary, because it increases power consumption.

rfidclose

Description Switches the RFID module off.

Syntax
```
void rfidclose (void)
```

Arguments -

Returns -

Remarks
- Call this function whenever done reading an RFID label, to reduce power consumption.
- If the RFID device was already closed, the function does nothing.
- If a good-read LED is used to indicate “RFID reader active”, set it to off on closing the RFID device.

rfidversion

Description returns the firmware version string of the RFID module

Syntax
```
int rfidversion (char *buf)
```

Arguments `buf` pointer to a buffer into which the string is put, i.e. “HAAV0007”

Returns

- on success		the number of characters written to the buffer (=8)
- on error (e.g. RFID device not open)		-1

Remarks
The buffer size must be at least 8 characters.
The string is not zero-terminated.

rfid_lib_version

Description returns the library version string of the 13,56 MHz RFID module

Syntax
```
char *rfid_lib_version(void)
```

Arguments -

Returns - pointer to version string of 8 characters i.e. “CPWV0104”

Remarks The string is not zero-terminated.

rfidsetoptions

Description sends 4 bytes of options values to the module

Syntax
```
int rfidsetoptions (unsigned char *buf)
```

Arguments

`buf`	pointer to the buffer that contains the options bytes

Returns

- on success	0
- on error	-1

Remarks Option 0 is defined as the Family Code for I-Code, and option 1 is defined as the Application Identifier for I-Code.

In firmware version HAAV0007 and higher, bit 0 of Option 0 selects the value of the Option_flag that is used in writing operations to ISO15693 tags. If this bit is 1, the operation will be executed with Option_flag=0, if this bit is 0, it will be executed with Option_flag=1. See the appendix on ISO15693.

In the current versions of the RFID reader module, options 2 and 3 are reserved.

The default option values are all zero. The option values are reset when the RFID device is closed.

rfid_testmode

Description Sets the antenna power off, or on, with or without modulation.

Syntax `int rfid_testmode (unsigned int mode)`

Arguments

`mode`	0: antenna power off 1: antenna power on 2: antenna power on, modulation on

Returns

no error	0
general error	-1

Remarks this function is intended for diagnostics and adjustment purposes.

Tag-it read/write functions

rfid_ti_version

Description Get version data of a Tag-It label within range, and places it in the specified buffer (8 bytes).

Syntax `int rfid_ti_version (unsigned char *buf)`

Arguments

`buf`	pointer to the buffer that the result string is placed in

Returns

- on success 0
- on general error (e.g. RFID device not open) -1
- if tag not in range -2

Remarks

- If the application uses a mix of Tag-It versions with different block sizes and/or number of blocks, then use this function to find out what they are.
- see below description of the version data format.

The version data format corresponds to the below table.

# of bits	Example value	Description
32	$00018B1A_{16}$	Transponder serial number is 00018B1A
7	0000001_2	Chip manufacturer code (1=Texas Instruments)
9	000000101_2	Chip version 5
3	000_2	Reserved
5	00011_2	Block size (in bytes) minus one (3+1=4 bytes=32 bits)
8	07_{16}	Number of blocks minus one (7+1=8 blocks)

rfid_ti_getblocks

Description Get data from Tag-It label if within range.

Syntax

```
int rfid_ti_getblocks (unsigned char *buf, unsigned int
block_start, unsigned int nblocks, unsigned int nbytes,
unsigned char *error)
```

Arguments

`buf`	pointer to where the data must be put
`block_start`	first block requested
`nblocks`	number of blocks requested
`nbytes`	number of bytes in a block (must correspond to the block size of the tag)
`error`	pointer to specific error code

Returns

- on success 0
- on general error -1
- tag not in range -2
- on specific error -3 (see specific error codes)

Specific error codes 10 (hex) block not available

Remarks

- If the application uses a mix of Tag-It versions with different block sizes and/or number of blocks, then use `rfid_ti_version()` to find out what they are.
- the buffer size must be at least `nblocks*nbytes`.
- If the return value is not -3, then the value of `*error` is undefined.

rfid_ti_lockstat

Description Get lock status for a block in a Tag-It label.

Syntax
```
int rfid_ti_lockstat (unsigned char *buf, unsigned int
block, unsigned int nbytes, unsigned char *error)
```

Arguments

`buf`	pointer to a `char` where the status data must be put
`block`	block number
`nbytes`	number of bytes in a block (must correspond to the block size of the tag)
`error`	pointer to specific error code

Returns

- on success	0
- on general error	-1
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

10 (hex)	block not available

Lock status codes

0	not locked
1	user locked
2	factory locked
3	reserved

Remarks
- If the application uses a mix of Tag-It versions with different block sizes and/or number of blocks, then use `rfid_ti_version()` to find out what they are.
- If the return value is not -3, then the value of `*error` is undefined.

rfid_ti_putblock
rfid_ti_putandlockblock

Description write a block of data to Tag-it label in range.

Syntax
```
int rfid_ti_putblock (unsigned char *buf, unsigned int
block,  unsigned int nbytes, unsigned char *error)

int rfid_ti_putandlockblock (unsigned char *buf, unsigned
int block,  unsigned int nbytes, unsigned char *error)
```

Arguments

`buf`	pointer to the data
`block_start`	block to write
`nbytes`	number of bytes in a block (must correspond to the block size of the tag)
`error`	pointer to where an error code must be put

Returns

- on success	0
- on general error	-1
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

10 (hex)	block not available
12 (hex)	block already locked; not written
16 (hex)	block not successfully put

Remarks
- If the application uses a mix of Tag-It versions with different block sizes and/or number of blocks, then use `rfid_ti_version()` to find out what they are.
- In case of return value -2 or -3, the data may have been incorrectly written, so it may be a good idea to retry the operation until successful.
- `rfid_ti_putandlockblock()` is the same as `rfid_ti_putblock()` except that the block is locked at the same time.

rfid_ti_lockblock

Description Lock block (set block to read-only) in a Tag-It label.

Syntax

```
int rfid_ti_lockblock (unsigned int block, unsigned char
*error)
```

Arguments

`block`	number of the block to be locked
`error`	pointer to where an error code must be put

Returns

- on success	0
- on general error	-1
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

10 (hex)	block not available
14 (hex)	block secured
18 (hex)	block not successfully locked

Remarks In case of return value -2 or -3, the data may have been incorrectly written, so it may be a good idea to retry the operation until successful.

I-Code read/write functions

rfid_ic_getblocks

Description Get data from I-code within range

Syntax
```
int rfid_ic_getblocks(unsigned char *buf, unsigned int
block_start, unsigned int nblocks)
```

Arguments

`buf`	buffer in which the data must be placed
`block_start`	first data block to read
`nblocks`	number of blocks to read

Returns

- on success 0
- on general error -1
- tag not in range -2

Remarks

- the size of `buf` must be at least `nblocks` times 4.
- `block_start` must be at most 15
- nblocks must be at least 1 and at most 16-block_start.

rfid_ic_putblock

Description Write a block of data to an I-code label

Syntax
```
int rfid_ic_putblock (unsigned char *buf, unsigned int
block)
```

Arguments

`Buf`	buffer with the data to be written
`block`	number of data block to be written

Returns

- on success 0
- on general error -1
- tag not in range -2
- verify error -3
- unable to verify -4

Remarks If result code 0 is returned, the tag was successfully written.
If result code -2 is returned, no tags were initially in range and no tag is written.
If result code -3 is returned, the data read back from the tag is different than that supposed to be written (it may be the old data or something else).
If result code -1 or -4 is returned, it is undetermined whether the tag has been (correctly) written: communication with the module was broken, or the tag moved out of range, before the module was able to read the data back for verification.
If you use this command to set the QUIET bits then you should expect result code -4 on success.

rfid_ic_resetquietbit(void)

Description reset the QUIET bit of all I-code labels in range

Syntax
```
int rfid_ic_resetquietbit(void)
```

Arguments -

Returns

no error	0
general error	-1

Remarks Result code 0 just means “no error” and not “success” since the I-code label does not respond to this command. You can check if the operation was successful by trying to read some data from it.

ISO15693 read/write functions

rfid_iso_systeminfo

Description Get system information of a ISO15693 label within range, and places it in the specified buffer (12 databytes).

Syntax
```
int rfid_iso_systeminfo (unsigned char *buf)
```

Arguments `Buf` Pointer to the buffer that the result string is placed in

Returns
- on success 0
- on general error (e.g. RFID device not open) -1
- if tag not in range -2

Remarks
- If the application uses a mix of ISO15693 versions with different block sizes and/or number of blocks, then use this function to find this out.
- see below description of the version data format.

The version data format corresponds to the below table.

# of bytes	Example value	Description
8	$E007000000000082_{16}$	Transponder serial number is E007000000000082
1	00_{16}	Data Storage Format Identifier (DSFID)
1	00_{16}	Application Family Identifier (AFI)
1	03_{16}	Block size (in bytes) minus one (3+1=4 bytes)
1	$3F_{16}$	Number of blocks minus one (3F+1=64 blocks)

rfid_iso_getblocks
rfid_iso_getblocks_fast

Description Get data from ISO15693 label if within range.

Syntax
```
int rfid_iso_getblocks (unsigned char *buf, unsigned int
block_start, unsigned int nblocks, unsigned int nbytes,
unsigned char *error)

int rfid_iso_getblocks_fast (unsigned char *buf, unsigned
int block_start, unsigned int nblocks, unsigned int nbytes,
unsigned char *error)
```

Arguments

`buf`	pointer to where the data must be put
`block_start`	first block requested
`nblocks`	number of blocks requested
`nbytes`	number of bytes in a block (must correspond to the block size of the tag)
`error`	pointer to specific error code

Returns
- on success 0
- on general error -1 (e.g. RFID device not open or communication lost)
- tag not in range -2
- on specific error -3 (see specific error codes)

specific error codes 10 (hex) block not available

Remarks
- If the application uses a mix of ISO15693 versions with different block sizes and/or number of blocks, then use `rfid_iso_systeminfo()` to find out what they are.
- The buffer size must be at least `nblocks*nbytes`.
- If the return value is not -3, then the value of `*error` is undefined.

The only difference between the functions: 'rfid_iso_getblocks' and 'rfid_iso_getblocks_fast' is that they send different commands to the ISO15693 label. The 'rfid_iso_getblocks'-function uses the 'read single block'-command to read every single requested block. The 'rfid_iso_getblocks_fast'-function uses the faster 'get multiple blocks'-command to read a maximum of 4 blocks at once. For this reason the 'rfid_iso_getblocks_fast'-function reads multiple blocks in a much shorter period of time. If an apllication needs to read more than one block of data at a time, it's recommended to use the 'rfid_iso_getblocks_fast'-function

rfid_iso_lockstats

Description Gets lock statuses of multiple blocks in an ISO15693 label.

Syntax
```
int rfid_iso_lockstats (unsigned char *buf, unsigned int
block, unsigned int nblocks, unsigned char *error)
```

Arguments

`buf`	pointer to a `char` where the status data must be put
`block`	block number
`nblocks`	number of blocks
`error`	pointer to specific error code

Returns

- on success	0
- on general error	-1 (e.g. RFID device not open or communication lost)
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

10 (hex)	block not available

Lock status codes

0 (hex)	not locked
1 (hex)	locked

Remarks

- If the return value is not -3, then the value of `*error` is undefined.
- The lock status of a block of data is encoded in one byte. Only the least significant bit from this byte is used. (The rest of the bits are reserved for future use). If this bit is set the block is permanently locked.

rfid_iso_putblock

Description Writes a block of data to an ISO15693 label in range.

Syntax
```
int rfid_iso_putblock(unsigned char *buf, unsigned int
block, unsigned int nbytes, unsigned char *error)
```

Arguments

`buf`	Pointer to the data
`block_start`	Block to write
`nbytes`	Number of bytes in a block (must correspond to the block size of the tag)
`error`	Pointer to where an error code must be put

Returns

- on success	0
- on general error	-1 (e.g. RFID device not open or communication lost)
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

10 (hex)	Block not available
12 (hex)	Block locked; contents cannot be changed
13 (hex)	Block not successfully programmed

Remarks

- If the application uses a mix of ISO15693 versions with different block sizes and/or number of blocks, then use `rfid_iso_systeminfo()` to find out what they are.
- In case of return value -2 or -3, the data may have been incorrectly written, so it may be a good idea to retry the operation until successful.
- Depending on the tag IC type, you have to set option byte 0 bit 0. See the appendix on ISO15693.

rfid_iso_putAFI

Description Writes AFI to an ISO15693 label in range.

Syntax
```
int rfid_iso_putAFI(unsigned char *buf, unsigned char
*error)
```

Arguments

`buf`	Pointer to the data
`error`	Pointer to where an error code must be put

Returns

- on success	0
- on general error	-1 (e.g. RFID device not open or communication lost)
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

12 (hex)	AFI locked; contents cannot be changed
13 (hex)	AFI not successfully programmed

Remarks

- In case of return value -2 or -3, the AFI may have been incorrectly written, so it may be a good idea to retry the operation until successful.
- Depending on the tag IC type, you have to set option byte 0 bit 0. See the appendix on ISO15693.

rfid_iso_putDSFID

Description Writes DSFID to an ISO15693 label in range.

Syntax
```
int rfid_iso_putDSFID(unsigned char *buf, unsigned char
*error)
```

Arguments

`buf`	Pointer to the data
`error`	Pointer to where an error code must be put

Returns

- on success	0
- on general error	-1 (e.g. RFID device not open or communication lost)
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

12 (hex)	DSFID locked; contents cannot be changed
13 (hex)	DSFID not successfully put

Remarks

- In case of return value -2 or -3, the DSFID may have been incorrectly written, so it may be a good idea to retry the operation until successful.
- Depending on the tag IC type, you have to set option byte 0 bit 0. See the appendix on ISO15693.

rfid_iso_lockblock

Description Locks block (set block to read-only) of an ISO15693 label.

Syntax
```
int rfid_iso_lockblock (unsigned int block, unsigned char
*error)
```

Arguments

`block`	Number of the block to be locked
`error`	Pointer to where an error code must be put

Returns

- on success	0
- on general error	-1 (e.g. RFID device not open or communication lost)
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

10 (hex)	Block not available
11 (hex)	Block already locked
14 (hex)	Block not successfully locked

Remarks In case of return value -2 or -3, the data may have been incorrectly written, so it may be a good idea to retry the operation until successful.
Depending on the tag IC type, you have to set option byte 0 bit 0. See the appendix on ISO15693.

rfid_iso_lockAFI

Description Lock AFI (set AFI to read-only) of an ISO15693 label.

Syntax
```
int rfid_iso_lockAFI(unsigned char *error)
```

Arguments

`Error`	Pointer to where an error code must be put

Returns

- on success	0
- on general error	-1 (e.g. RFID device not open or communication lost)
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

10 (hex)	AFI not available
11 (hex)	AFI already locked
14 (hex)	AFI not successfully locked

Remarks In case of return value -2 or -3, the AFI may have been incorrectly locked, so it may be a good idea to retry the operation until successful.
Depending on the tag IC type, you have to set option byte 0 bit 0. See the appendix on ISO15693.

rfid_iso_lockDSFID

Description Lock DSFID (set DSFID to read-only) of an ISO15693 label.

Syntax
```
int rfid_iso_lockDSFID(unsigned char *error)
```

Arguments

`Error`	Pointer to where an error code must be put

Returns

- on success	0
- on general error	-1 (e.g. RFID device not open or communication lost)
- tag not in range	-2
- on specific error	-3 (see specific error codes)

Specific error codes

10 (hex)	DSFID not available
11 (hex)	DSFID already locked
14 (hex)	DSFID not successfully locked

Remarks In case of return value -2 or -3, the DSFID may have been incorrectly locked, so it may be a good idea to retry the operation until successful.
Depending on the tag IC type, you have to set option byte 0 bit 0. See the appendix on ISO15693.

Appendix: Information about Tag-it labels

The following information is based on documentation provided by Texas Instruments.

Address (serial number)

Each Tag-it transponder has a unique address that is factory-programmed and is 32 bits long, allowing an address range of more than 4 billion individual addresses. The possibility cannot be excluded that at some future time, a previously used address is assigned to a new transponder. Considering the expected lifetime of transponders however, and based on statistical calculations, the probability that 2 transponders with the same address are present simultaneously in the same reader location is well below 10^{-10}.

Non-addressed operation

The reader module in the PHL-2700 supports non-addressed operation only.
It is key to successful implementation of the non-addressed operation that only one transponder is within the reader's range.
Otherwise, the following may occur:

- if the reader is performing a read function, 2 or more answers will be sent by transponders. These answers will collide, resulting in an unintelligible message at the reader's side.
- if the reader is performing a write function, all reachable transponders will perform this function, and for the reasons stated above, such that the reader will not receive a clear confirmation that the operation has been performed correctly. This can result in data corruption or render the transponder unusable if the Lock command has been used.

Transponder version

Tag-it products have been designed as a family of products based on a common technology. This means that transponders with different IC's, having different features, may be presented to the reader.
In order to know what the characteristics of each transponder IC are, and thus be able to correctly request execution of commands, each Tag-it IC is programmed during manufacturing with its version number, its manufacturer code and additional information about memory size and structure.

Memory organization

The Tag-it IC user memory is typically organized in blocks (or pages) with each block individually addressable. All blocks have the same size, which can be found in the relevant IC datasheet. The block size and the number of blocks can also be retrieved using the rfid_ti_version() function. In addition to user memory, service memory and information memory are implemented. Service memory is provided to contain information about the memory, e.g. the locking status of the blocks. Information memory contains information about the IC, for instance the SID address, the IC version and some elements of its characteristics. Service memory and information memory can be accessed using the specific functions provided for the purpose.

The currently available transponders have chip version 5, and contain 8 blocks of data, each 32 bits (4 bytes) in size.

User memory

	byte 0	byte 1	byte 2	byte 3	
block 0					user data
block 1					:
block 2					:
block 3					:
block 4					:
block 5					:
block 6					:
block 7					user data

Block locking

A block can only be locked once. A locked block cannot be modified by any subsequent command. It is permanently locked and the data contained in the block cannot be changed.
Blocks can be locked using the library functions provided for this purpose.
It is also possible to have blocks factory-locked; contact a Texas Instruments sales office for more information about this.

Appendix: I-Code labels

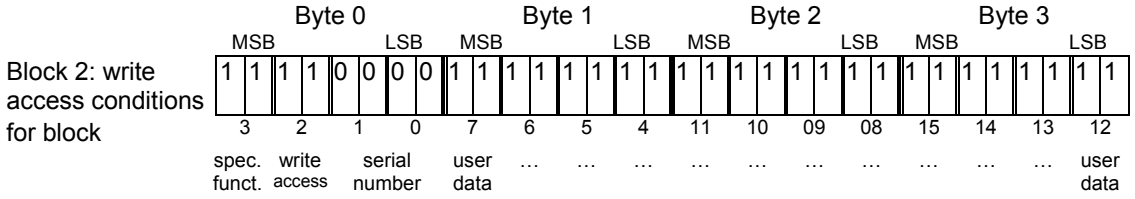
The following information is based on documentation provided by Philips.
The EEPROM in the I-Code1 IC has a memory capacity of 512 bit and is organised in 16 blocks consisting of 4 bytes each (1 block = 32 bits). The higher 12 blocks contain user data and the lowest 4 blocks contain the serial number, the write access conditions and some configuration bits. Block 4 may contain a family code and application identifier.

	byte 0	byte 1	byte 2	byte 3	
block 0	SNR0	SNR1	SNR2	SNR3	serial number (lower bytes)
block 1	SNR4	SNR5	SNR6	SNR7	serial number (higher bytes)
block 2	F0	FF	FF	FF	write access conditions
block 3	x	x	x	x	special functions (EAS/QUIET)
block 4	x	x	x	x	family code/application identifier/user data
block 5	x	x	x	x	user data
block 6	x	x	x	x	:
block 7	x	x	x	x	:
block 8	x	x	x	x	:
block 9	x	x	x	x	:
block 10	x	x	x	x	:
block 11	x	x	x	x	:
block 12	x	x	x	x	:
block 13	x	x	x	x	:
block 14	x	x	x	x	:
block 15	x	x	x	x	user data

The values, in hexadecimal, are shown as stored in the EEPROM after the wafer production process. The contents of blocks marked with ‘x’ are not defined at delivery.

Serial number
The unique 64 bit serial number is stored in blocks 0 and 1 and is programmed during the production process.

Write access conditions
The write access conditions bits in block 2 determine the write access conditions for each of the 16 blocks. These bits can only be set to 0 (and never be changed to 1), i.e. already write protected blocks can never be written to from this moment on. This is also true for block 2. If this block is set into write protected state by clearing of bits 4 and 5 at byte 0, no further changes in write conditions are possible.



The write access conditions must be cleared in pairs. The least significant 2 bits in byte 0 of block 2 control write access to block 0, and so forth. Writing of bit pairs 1|0 or 0|1 is not allowed! For example, to write protect block 4, read block 4, clear the two least significant bits of byte 1, and write back the result. On manufacture, the 4 least significant bits of byte 0 of block 2 are already cleared, so that the serial number of the label cannot be changed.

It is extremely important to be particularly careful when clearing the write access bits in block 2, as you can lose write access to all of the blocks on the label in case of a mistake.

Special functions
The Special functions block holds the two EAS bits (Electronic Article Surveillance mode active --> the label answers at and EAS command) as well as the two QUIET bits (QUIET mode enabled --> the label is permanently disabled but can be activated again using the “reset QUIET bit” command). The state of the QUIET bit does not influence the functionality of the EAS command.
The remaining 28 bits in block 3 are reserved for future use.
Writing of bit pairs 0|1 or 1|0 to block 3 is not allowed.

Changing of the write access control or configuration must be done in a secure environment (by reading the current value of the block and masking in the new values for bit positions that may be changed). The label must not be moved out of the communication field of the antenna during writing! It is recommended to put the label close to the antenna and not to remove it during the operation.

Family Code and Application Identifier
The I-Code system offers the feature to use (independently) Family Codes and /or Application Identifiers with some commands (this ability can be used to create “label families”.
These two 8-bit values are located at the beginning of User Data (block 4) and are only evaluated if the corresponding bytes at the commands are unequal to zero (this can be set using the rfidsetoptions() function)
If the corresponding bytes at the commands are zero, then these locations can be used for user data without restriction.

Configuration of delivered IC’s
Philips delivers the I-Code1 label IC’s with the following configuration:
- Serial number is unique and read only
- Write access conditions allow to change all other blocks
- Status of EAS and QUIET mode, Family Code, Application Identifier, and user data is NOT defined.

Note: as the status of QUIET mode is not defined at delivery, the first command to be executed on the label should be the “reset QUIET bit” command!

Appendix: Information about ISO15693 labels

Unique identifier (UID)
Each ISO15693 transponder has a unique address that is factory-programmed and is 64 bits long.
The unique identifier is set permanently by the IC manufacturer in accordance with the figure below.

MSB		LSB
64 57	56 49	48 1
'E0'	IC Mfg code	IC manufacturer serial number

The UID comprises:
- The 8 MSB shall always be 'E0'
- The IC manufacturer code is according to ISO/IEC 7816-6
- The unique serial number on 48 bits assigned by the IC manufacturer. The probability that at some future time, a previously used address is assigned to a new transponder and is presented by the same reader in the same period of time is practically nihil.

Application Family Identifier (AFI)
The ISO15693 labels have one byte reserved for the Application Family Identifier (AFI). This identifier represents the type of application targeted by the reader and can be used to create "label families".
This 8-bit value can only be addressed (read/write/lock) by specific commands and not by using the same commands that can be used to address a block of data.

The AFI is 1 byte long. The most significant nibble of the AFI selects the application families, if non-zero, as defined in the table below.
The least significant nibble of the AFI is used to code one specific or all application sub-families. Sub-family codes different from 0 are proprietary. (See ISO15693-part 3)

AFI most significant nibble	AFI least significant nibble	Meaning ISO15693-labels respond from	Examples/ note
'0'	'0'	All families and subfamilies	No applicative preselection
X	'0'	All sub-families of family X	Wide applicative preselection
X	Y	Only the Yth sub-family of family X	
'0'	Y	Proprietary sub-family Y only	
'1'	'0', Y	Transport	Mass transit, Bus, Airline
'2'	'0', Y	Financial	IEP, Banking, Retail
'3'	'0', Y	Identification	Access control
'4'	'0', Y	Telecommunication	Public telephony, GSM
'5'	'0', Y	Medical	
'6'	'0', Y	Multimedia	Internet services
'7'	'0', Y	Gaming	
'8'	'0', Y	Data storage	Portable files
'9'	'0', Y	Item management	
'A'	'0', Y	Express parcels	
'B'	'0', Y	Postal services	
'C'	'0', Y	Airline bags	
'D'	'0', Y	RFU (Reserved for future use)	
'E'	'0', Y	RFU	
'F'	'0', Y	RFU	

Notes: X = '1' to 'F', Y = '1' to 'F'
The support of AFI by the ISO15693 labels is optional

Data storage Format Identifier (DSFID)
The ISO15693 labels have one byte reserved for the Data Storage Format Identifier (DSFID). This identifier indicates how the data is structured in the IC's memory. This 8-bit value can only be addressed (read/write/lock) by specific commands and can not be addressed by using the commands to address a block of data.
It is intended to provide a description of the logical organisation of the data. ISO15693 does not specify the formats. The default value is '00'

Memory organisation
The physical memory is organised in blocks of fixed size. The block size can be up to 256 bits and there can be up to 256 blocks.
The actual block size and the number of blocks depend on the version of the ISO15693 labels, if a mix of ISO15693 versions us used, it's necessary to check the block sizes and number of blocks.

Write access conditions
The write access conditions of a block of data are encoded in one additional byte per block. Currently only the least significant bit is used to encode the lock status. If this bit is '0' than the block is marked as 'Not locked' and if this bit is set the block is permanently locked. The remaining bits are reserved for future use.

Optional commands
Even though the commands of this library are supported by almost all versions of ISO15693 labels. It might be possible that a version of ISO15693-labels is used that doesn't support one or more of the commands.
This is possible because many of the used commands implemented in this library are optional as far as ISO15693 part 3 concerned. This means that the manufacturer can decide whether to implement al the commands or not.
For example, not all versions of ISO15693 labels support the AFI and DSFID commands. The other commands are very likely to be implemented in all versions of ISO15693 labels.

Block locking
Locking a block, the AFI or the DSFID is a one-way operation. (You can't unlock them once they are locked). For this reason locked data cannot be modified by any subsequent command. It is permanently locked and the data of this block or identifier cannot be changed.

Write and lock commands
The ISO15693 labels need a short period of time (<20ms) to perform a write (or lock) action. After this period the reader receives confirmation of this writing action from the label. If the label is moved out of the field of the reader before this confirmation could be asked, the write and locks commands of this library will return the error code: '-2: tag not in range' or '-3', even though the write action could have been successfully performed by the label.
For this reason it's recommended that the results of the write and lock commands will be confirmed by a second command if the command returns one of those error codes. (This can be done be reading the data of the specific block or by checking the lock status).

Writing and locking ISO15693 labels from different manufacturers
According to ISO15693, two alternative writing and locking operation sequences are allowed, and these must be selected using the Option_flag that is sent as part of the command to the tag. Unfortunately, some types of ISO15693 labels support only Option_flag=1, and others only Option_flag=0. If the correct value is not specified, the tag will not execute the write operation.

In firmware version HAAV0007 and higher, you can select the Option_flag value using option byte 0, bit 0, using the library function rfid_setoptions(). For compatibility reasons, Option_flag is set to the reverse of this bit value.

You can use the function rfid_system_info() to find out the manufacturer code and IC type. We have found that tags from Texas Instruments (IC manufacturer code 07) can be written using the default setting (option byte 0=0). To write the SL2 IC S20 from Philips (IC manufacturer code 04) you have to set option byte 0 to 1 before writing to the tag. Please refer to the data sheets of the tag IC's.