

**MC2002 Hand-Held Smart Card
Read/Write Device (RWD)
Optional Device Library Manual:
Access ISO15693 Chips with
RC632**

Version 1.0

Sep 2003

REVISION HISTORY

Version Number	Date	Description	Author
1.0	12 Sep, 2003	Initial release	HYJ

Contents

INTRODUCTION	5
1.1 PHILIPS RC632	6
1.2 SYSTEM REQUIREMENT	6
OVERVIEW	7
2.1 TRANSACTION FLOW	8
2.2 ISO15693 CHIPS STATE TRANSITION DIAGRAM.....	9
2.3 EXTENDED ERROR CODE.....	10
2.4.1 <i>Interface initialization</i>	11
2.4.2 <i>Interface deactivation</i>	11
2.4.3 <i>Reset RF field</i>	11
2.5 ISO15693 CHIPS OPERATION.....	11
2.5.1 <i>Select One Card</i>	11
2.5.2 <i>Stay Quiet</i>	11
2.5.3 <i>Read Block</i>	12
2.5.4 <i>Write Block</i>	12
2.5.5 <i>Lock Block</i>	12
2.5.6 <i>Select with UID</i>	13
2.5.7 <i>Reseat to Ready</i>	13
2.3.8 <i>Write AFI</i>	13
2.5.9 <i>Lock AFI</i>	13
2.5.10 <i>Write DSFID</i>	14
2.5.11 <i>Lock DSFID</i>	14
2.5.12 <i>Get System Informat</i> e.....	14
2.5.13 <i>Get Multi-Block Security</i>	14
SUBROUTINES	15
3.1 INTERFACE CONTROL	16
3.1.1 <i>Interface initialization</i>	16
3.1.2 <i>Interface deactivation</i>	17
3.1.3 <i>Reset RF field</i>	18
3.2 ISO15693 CHIPS OPERATION.....	19
3.2.1 <i>Select One Card</i>	19
3.2.2 <i>Stay Quiet</i>	21
3.2.3 <i>Read Block</i>	22
3.2.4 <i>Write Block</i>	23
3.2.5 <i>Lock Block</i>	24
3.2.6 <i>Select with UID</i>	25
3.2.7 <i>Reseat to Ready</i>	26
2.3.9 <i>Write AFI</i>	27

MC2002™

2.5.9	<i>Lock AFI</i>	28
2.5.10	<i>Write DSFID</i>	29
2.5.11	<i>Lock DSFID</i>	30
2.5.12	<i>Get System Information</i>	31
2.5.13	<i>Get Multi-Block Security</i>	32
SAMPLES		33
4	WRITE INPUT DATA TO A DESIGNATED BLOCK OF A TI-TAGIT	34

Chapter 1

Introduction

Introduction

1

1.1 Philips RC632

The CL RC632, from Philips Semiconductors, is a versatile multi-standard single-chip reader IC for contactless smart cards and labels operating at 13.56 MHz. It is designed to support the ISO contactless standards in this frequency range, including ISO14443 and ISO15693, and offers system integrators the flexibility to develop interoperable RFID systems for different high-volume reader applications.

In this manual mainly introduce how to access chips measuring up ISO15693 standard (such as Philips I-Code and Ti-Tagit) with RC632.

1.2 System requirement

An optional peripheral unit must be used for MC2002 to access ISO15693 chips. This unit features an ISO15693 interface chip by Philips RC632, which is capable of accessing ISO15693 vicinity tags.

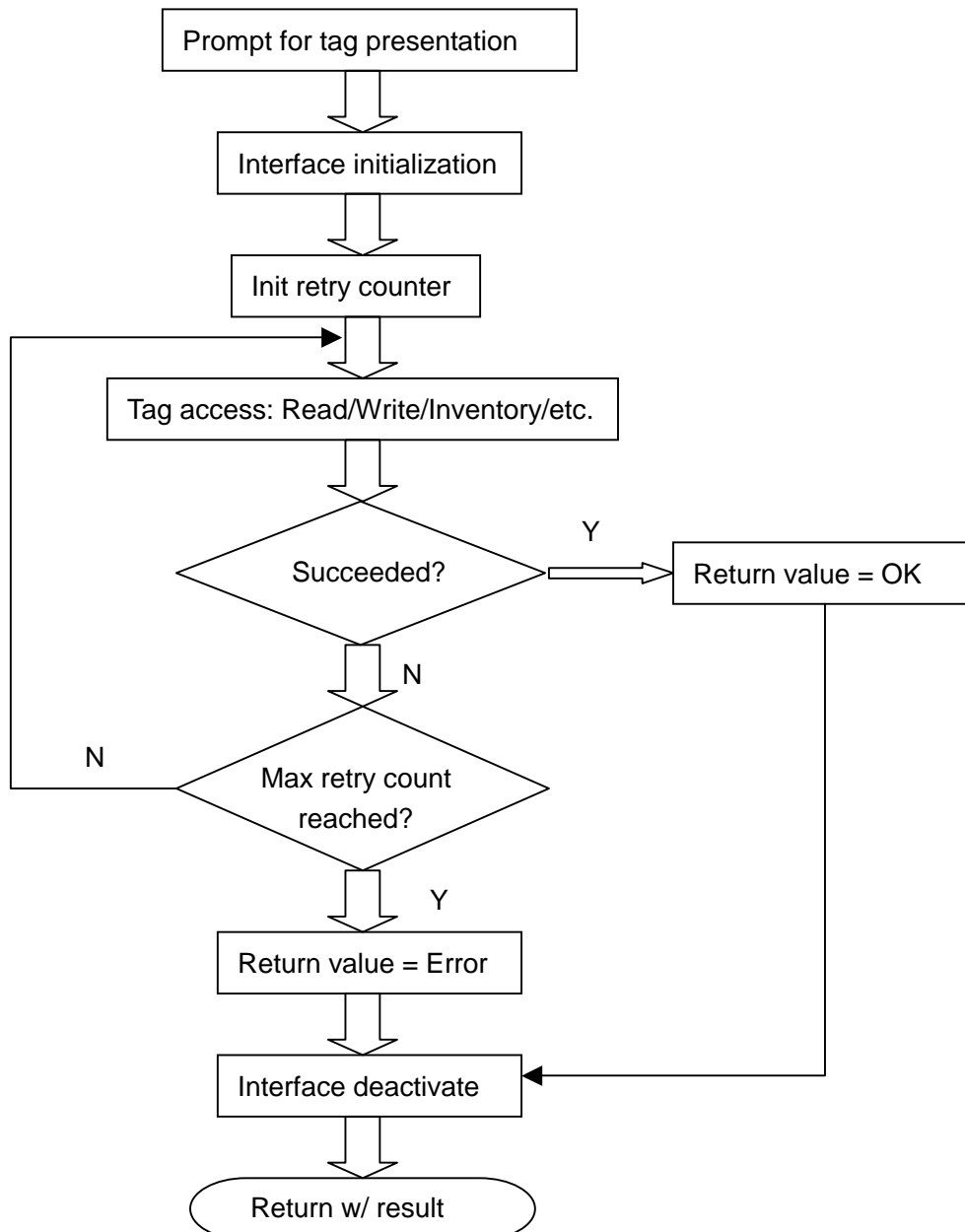
An additional library named **Rc632-15693.a** should be added to the link script file (.ld) of the project, and a header file **RC632-15693.H** should also be included in the source code.

Chapter 2

Overview

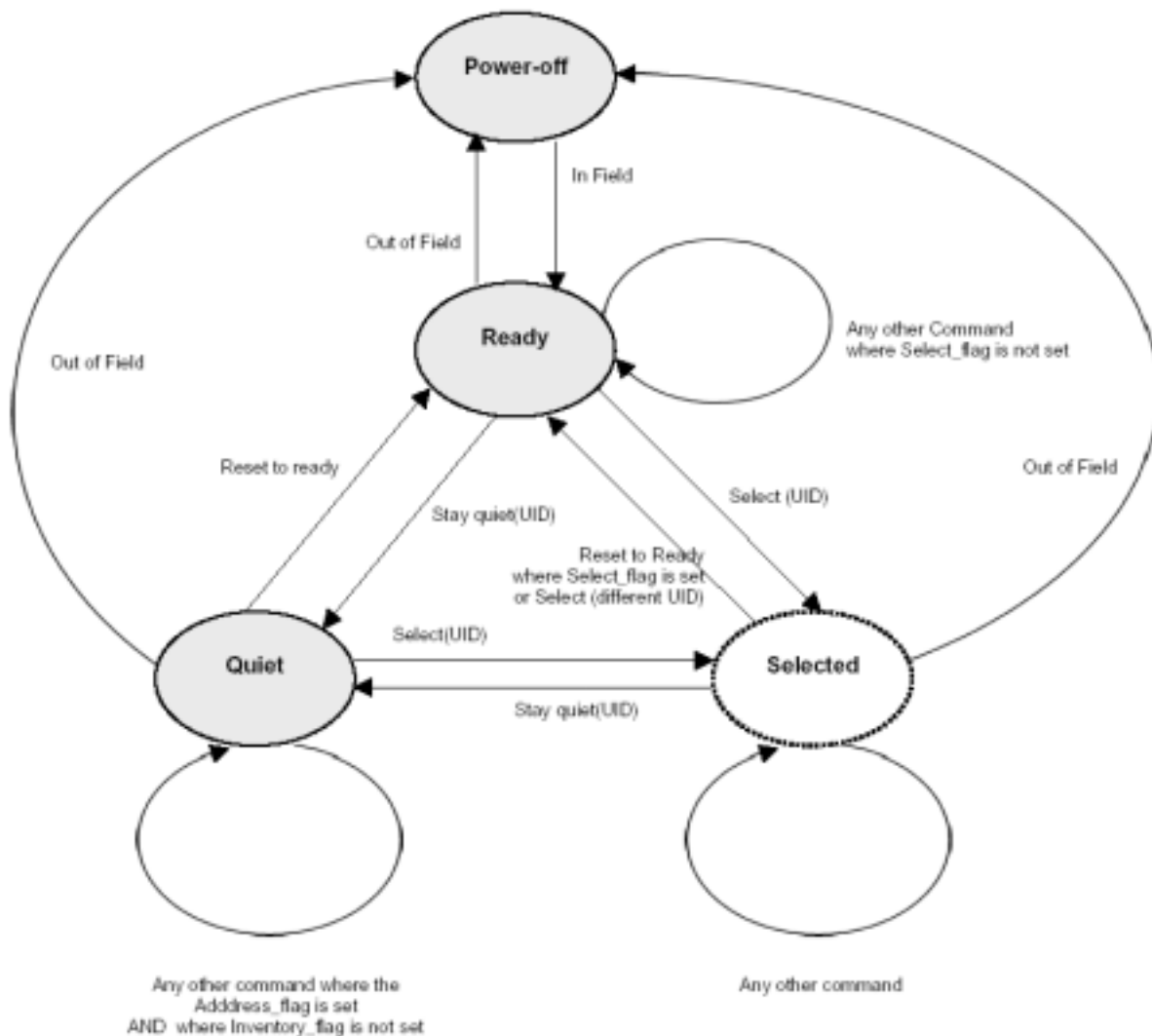
2.1 Transaction Flow

Since the vicinity tag-accessing unit is quite power consuming, it's recommended that it be turned on only when it's necessary. The recommended transaction flow should be as:



2.2 ISO15693 chips State Transition Diagram

There is a State Transition Diagram in ISO/IEC FCD 15693-3, see below:



2.3 Extended Error Code

ISO15693 standard has defined itself Error Code, so we must extend originally Error Code in < RC632-15693.H >.

```
#define CRCERROR          0xFF00//Response CRC error
#define CMDNOTSPORTED    0xFF01//The request code is not recognized
#define CMDNOTRECOGNISED 0xFF02//A format error occurred
#define OPTNOTSPORTED    0xFF03//The option is not recognized
#define UNKNOW           0xFF0F//Unknow error
#define BLOCKNOTAVAILABLE 0xFF10//The specified block is not available(doesn't exist)
#define BLOCKLOCKED      0xFF11//The block is already locked, cannot be locked again
#define BLOCKCANNTCHANGED 0xFF12//The block is locked and its content cannot be changed
#define BLOCKWRITEERROR  0xFF13//The specified block was not successfully programmed
#define BLOCKLOCKERROR   0xFF14//The specified block was not successfully locked
//0xFFA0~0xFFDF      : Custom command error code
//All other 0xFFxx   : RFU
```

2.4 Interface control

2.4.1 Interface initialization

The tag accessing unit should be initialized before any actual access begins. The interface is powered on, but the RF interface is shut down, and the current should rise up.

Subroutine used:

```
InitMC632as15693();
```

2.4.2 Interface deactivation

Interface deactivation will cut off the power of the entire tag accessing interface.

It's strongly recommended that the interface be turned off as soon as it's possible for maximum power saving. However, if the current is acceptable, the interface could be left on for the speed of next accessing.

Subroutine used:

```
MC632Off();
```

2.4.3 Reset RF field

This subroutine will cut the RF field for a moment in order to reset all chips that are in the field.

Subroutine used:

```
MC632ResetAllCard();
```

2.5 ISO15693 chips operation

The term **tag** in this section always refers to vicinity RFID tag that complies with ISO 15693, i.e., the VICC (vicinity integrated circuit card) in the standard document. C.f. ISO 15693-3 for detailed descriptions of the protocol and data format descriptions.

Both addressed and non-addressed mode could be used for all subroutines except Inventory (unnecessary for that one). Non-addressed mode is much faster but it could not handle more than one tag in the field.

According to the nature of RF communication, these commands might fail sometimes. Larger retry count may solve this problem at the cost of time and power consuming.

2.5.1 Select One Card

This subroutine envelops the Inventory Command to perform the anticollision sequence, and use 1 slot in inventory mode to select out only one chip in the RF fields.

Subroutine used:

```
ISO15693_SelectOneCard();
```

2.5.2 Stay Quiet

When receiving the Stay quiet command, the VICC shall enter the quiet state and shall NOT

send back a response. There is NO response to the Stay quiet command.

When in quiet state:

The VICC shall not process any request where Inventory_flag is set,

The VICC shall process any addressed request

The VICC shall exit the quiet state when:

Reset (power off),

Receiving a Select request. It shall then go to the selected state if supported or return an error if not supported,

Receiving a Reset to ready request. It shall then go to the Ready state.

Subroutine used:

```
ISO15693_StayQuiet();
```

2.5.3 Read Block

When receiving the Read one or multi block command, the VICC shall read the requested block(s) and send back its value in the response.

If the Option_flag is set in the request, the VICC shall return the block(s) security status, followed by the block(s) value. If it is not set, the VICC shall return only the block value.

Subroutine used:

```
ISO15693_ReadBlock();
```

2.5.4 Write Block

When receiving the Write single block command, the VICC shall write the requested block with the data contained in the request and report the success of the operation in the response.

If the Option_flag is not set, the VICC shall return its response when it has completed the write operation starting after a multiple of t_1 nominal with a total tolerance of $\pm 32/f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an 100% modulated EOF from the VCD and upon such reception shall return its response.

Example: When writing operation on Ti-Tagit, the Option_flag should be set, but Philips I-Code should not set this flag.

Subroutine used:

```
ISO15693_WriteBlock();
```

2.5.5 Lock Block

When receiving the Lock block command, the VICC shall lock permanently the requested block.

If the Option_flag is not set, the VICC shall return its response when it has completed the lock operation starting after a multiple of t_1 nominal with a total tolerance of $\pm 32/f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such

reception shall return its response.

Example: When writing operation on Ti-Tagit, the Option_flag should be set, but Philips I-Code should not set this flag.

Subroutine used:

```
ISO15693_LockBlock();
```

2.5.6 Select with UID

When receiving the Select command:

If the UID is equal to its own UID, the VICC shall enter the selected state and shall send a response.

If it is different, the VICC shall return to the Ready state and shall not send a response. the Select command shall always be executed in Addressed mode. (The Select_flag is set to 0. The Address_flag is set to 1.)

Subroutine used:

```
ISO15693_SelectUID();
```

2.5.7 Reseat to Ready

When receiving a Reset to ready command, the VICC shall return to the Ready state.

Subroutine used:

```
ISO15693_ResetToReady();
```

2.3.8 Write AFI

When receiving the Write AFI request, the VICC shall write the AFI value into its memory.

If the Option_flag is not set, the VICC shall return its response when it has completed the write operation starting after a multiple of t_1 nominal with a total tolerance of $\pm 32/f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

Subroutine used:

```
ISO15693_WriteAFI();
```

2.5.9 Lock AFI

When receiving the Lock AFI request, the VICC shall lock the AFI value permanently into its memory.

If the Option_flag is not set, the VICC shall return its response when it has completed the write operation starting after a multiple of t_1 nominal with a total tolerance of $\pm 32/f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

Subroutine used:

```
ISO15693_LockAFI();
```

2.5.10 Write DSFID

When receiving the Write DSFID request, the VICC shall write the DSFID value into its memory.

If the Option_flag is not set, the VICC shall return its response when it has completed the write operation starting after a multiple of t_1 nominal with a total tolerance of $\pm 32/f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

Subroutine used:

```
ISO15693_WriteDSFID();
```

2.5.11 Lock DSFID

When receiving the Lock DSFID request, the VICC shall lock the DSFID value permanently into its memory.

If the Option_flag is not set, the VICC shall return its response when it has completed the lock operation starting after a multiple of t_1 nominal with a total tolerance of $\pm 32/f_c$ and latest after 20ms.

If it is set, the VICC shall wait for the reception of an EOF from the VCD and upon such reception shall return its response.

Subroutine used:

```
ISO15693_LockDSFID();
```

2.5.12 Get System Informat

This command allows for retrieving the system information value from the VICC.

Subroutine used:

```
ISO15693_GetSystemInfo();
```

2.5.13 Get Multi-Block Security

When receiving the Get multiple block security status command, the VICC shall send back the block security status.

Subroutine used:

```
ISO15693_GetSystemInfo();
```

Chapter 3

Subroutines

3.1 Interface control

3.1.1 Interface initialization

`InitMC632as15693()` is the tag interface unit initialization subroutine. The interface would be powered on after invoking it.

```
char InitMC632as15693(void);
```

Parameters:

Void.

Return Values

N/A. Reserved for future usage.

NOTE: Before any other tag accessing subroutine is called, make sure this subroutine has been invoked, and that it is not deactivated. Otherwise, the program may hang up.

3.1.2 Interface deactivation

MC6320ff() is the tag interface unit deactivation subroutine. The interface would be powered off and consumes no power after invoking it.

```
void MC6320ff(void);
```

Parameters:

Void.

Return Values

Void.

3.1.3 Reset RF field

`MC632ResetAllCard()` is used to reset the RF fields .

```
unsigned char MC632ResetAllCard( void );
```

Parameters:

Void.

Return Values

N/A. Reserved for future usage.

3.2 ISO15693 chips operation

3.2.1 Select One Card

`ISO15693_selectOneCard()` is used to select only one chip in the RF fields .

```
short ISO15693_SelectOneCard(
    unsigned char Flags,
    unsigned char AFI,
    Tag_Info *Tag );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not. It consists of eight bits, see below (Copy from ISO/IEC FCD 15693-3).

Table 3 — Request flags 1 to 4 definition

Bit Nb	Flag name	State	Description
Bit 1	Sub-carrier_flag	0	A single sub-carrier frequency shall be used by the VICC
		1	Two sub-carriers shall be used by the VICC
Bit 2	Data_rate_flag	0	Low data rate shall be used
		1	High data rate shall be used
Bit 3	Inventory_flag	0	Flags 5 to 8 meaning is according to table 4
		1	Flags 5 to 8 meaning is according to table 5
Bit 4	Protocol Extension_flag	0	No protocol format extension
		1	Protocol format is extended. Reserved for future use

Table 4 — Request flags 5 to 8 definition when inventory flag is NOT set

Bit Nb	Flag name	State	Description
Bit 5	Select_flag	0	Request shall be executed by any VICC according to the setting of Address_flag
		1	Request shall be executed only by VICC in selected state
Bit 6	Address_flag	0	Request is not addressed. UID field is not present. It shall be executed by any VICC.
		1	Request is addressed. UID field is present. It shall be executed only by the VICC whose UID matches the UID specified in the request.
Bit 7	Option_flag	0	Meaning is defined by the command description. It shall be set to 0 if not otherwise defined by the command.
		1	Meaning is defined by the command description.
Bit 8	RFU	0	Shall be set to 0.

Table 5 — Request flags 5 to 8 definition when inventory flag is set

Bit Nb	Flag name	State	Description
Bit 5	AFI_flag	0	AFI field is not present
		1	AFI field is present
Bit 6	Nb_slots_flag	0	16 slots
		1	1 slot
Bit 7	Option_flag	0	Meaning is defined by the command description. It shall be set to 0 if not otherwise defined by the command.
		1	Meaning is defined by the command description.
Bit 8	RFU	0	Shall be set to 0.

AFI

AFI(Application Family Identifier) represents the type of application targeted by the VCD and is used to extract from all the VICCs present only the VICCs meeting the required application criteria. It may be programmed and locked by the respective commands. AFI is coded on one byte, which constitutes 2 nibbles of 4 bits each. Refer to ISO/IEC FCD 15693-3.

Tag

Point to the `Tag_Info` structural type which including UID and DSFID fields.

```
typedef struct{
    unsigned char UID[8]; //Unique Identifier
    unsigned char DSFID; //Data Storage Format Identifier
}Tag_Info;
```

Return Values

MI_OK: Succeeded

Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

3.2.2 Stay Quiet

`ISO15693_stayQuiet()` is used to set one chip into Quiet State. The Stay quiet command shall always be executed in Addressed mode (Select_flag is set to 0 and Address_flag is set to 1).

```
ISO15693_stayQuiet(  
    unsigned char Flags,  
    unsigned char *UID );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode
Any other value: pointer to the UID

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

3.2.3 Read Block

`ISO15693_ReadBlock()` is used to read one or multi block(s) from the user data area in the tag.

```
short ISO15693_ReadBlock(  
    unsigned char Flags,  
    unsigned char *UID,  
    unsigned char FirstBlock,  
    unsigned char BlockNumber,  
    unsigned short *RecLength,  
    unsigned char *RecData );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode
Any other value: pointer to the UID

FirstBlock

The first block to be read

BlockNumber

The amount of the blocks to be read

RecLength

The receive data length

RecData

Pointer to an array of block data buffer storing data read from the designated blocks (Discard the data if the subroutine returns error)

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

3.2.4 Write Block

`ISO15693_WriteBlock()` is used to write one or multi block(s) into the user data area in the tag.

```
short ISO15693_WriteBlock(  
    unsigned char Flags,  
    unsigned char *UID,  
    unsigned char FirstBlock,  
    unsigned char BlockNumber,  
    unsigned short BlockLength,  
    unsigned char *Data );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.

Any other value: pointer to the UID

FirstBlock

The first block to be read

BlockNumber

The amount of the blocks to be read

BlockLength

The data length of one block

Data

Pointer to a buffer storing data to be written into the designated block

Return Values

MI_OK: Succeeded

Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

3.2.5 Lock Block

`ISO15693_LockBlock()` is used to lock permanently the requested block.

```
short ISO15693_LockBlock(  
    unsigned char Flags,  
    unsigned char *UID,  
    unsigned char BlockNO );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.
Any other value: pointer to the UID

BlockNO

The first block to be locked

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

3.2.6 Select with UID

`ISO15693_selectUID()` is used to set a special chip into selected state.

```
short ISO15693_selectUID(  
    unsigned char Flags,  
    unsigned char *UID );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.
Any other value: pointer to the UID

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

3.2.7 Reset to Ready

`ISO15693_ResetToReady()` is used to set a special chip into ready state.

```
short ISO15693_ResetToReady(  
    unsigned char Flags,  
    unsigned char *UID );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.
Any other value: pointer to the UID

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

2.3.9 Write AFI

`ISO15693_WriteAFI()` is used to write the AFI value into its memory.

```
short ISO15693_WriteAFI(  
    unsigned char Flags,  
    unsigned char *UID,  
    unsigned char AFI );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.
Any other value: pointer to the UID

AFI

Application Family Identifier

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

2.5.9 Lock AFI

`ISO15693_LockAFI()` is used to lock the AFI value into its memory.

```
short ISO15693_LockAFI(  
    unsigned char Flags,  
    unsigned char *UID );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.
Any other value: pointer to the UID

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

2.5.10 Write DSFID

`ISO15693_WriteDSFID()` is used to write the DSFID value into its memory.

```
short ISO15693_WriteDSFID(  
    unsigned char Flags,  
    unsigned char *UID,  
    unsigned char DSFID );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.
Any other value: pointer to the UID

DSFID

Data storage format identifier

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

2.5.11 Lock DSFID

`ISO15693_LockDSFID()` is used to lock the DSFID value into its memory.

```
short ISO15693_LockDSFID(  
    unsigned char Flags,  
    unsigned char *UID );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.
Any other value: pointer to the UID

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

2.5.12 Get System Information

`ISO15693_GetSystemInfo()` is used to retrieving the system information value from the VICC.

```
short ISO15693_GetSystemInfo(  
    unsigned char Flags,  
    unsigned char *UID,  
    unsigned short *RecLength,  
    unsigned char *RecData );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.
Any other value: pointer to the UID

RecLength

The receive data length

RecData

Pointer to an array of block data buffer storing data read from the designated blocks (Discard the data if the subroutine returns error)

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

2.5.13 Get Multi-Block Security

`ISO15693_GetMultiBlockSecurity()` is used to get multiple block security status.

```
short ISO15693_GetMultiBlockSecurity(  
    unsigned char Flags,  
    unsigned char *UID,  
    unsigned char FirstBlock,  
    unsigned char BlockNumber,  
    unsigned short *RecLength,  
    unsigned char *RecData );
```

Parameters:

Flags

In a request, the flags specify the actions to be performed by the VICC and whether corresponding fields are present or not.

UID

For addressed mode, it's a pointer to an 8-byte buffer storing the UID of the tag to be read. Possible values:

NULL: non-addressed mode.
Any other value: pointer to the UID

FirstBlock

The first block to be read

BlockNumber

The amount of the blocks to be read

RecLength

The receive data length

RecData

Pointer to an array of block data buffer storing data read from the designated blocks (Discard the data if the subroutine returns error)

Return Values

MI_OK: Succeeded
Any other value: Error occurred, refer to error code define in <RC632-15693.h>.

Chapter 4

Samples

4 Write input data to a designated block of a Ti-tagit

This is a sample subroutine of writing data to a designated block of a tag. Both the data to be written and the block number are input from the keypad of the handheld. The writing is in non-addressed mode, so if you put more than one tag in the field, it will fail to write any of them.

```
long write_tag_to_block(void){
    long block_num, rc, tcnt;
    unsigned char p_data[128];

    //Get input data via an alphanumeric input dialogue and put the data in p_data
    memset(p_data, 0, 4);
    if(gets_dialog_alpha(p_data,"data",GETS_HINT_INVERSED)!=GETS_DIALOG_CONFIRM)
        memset(p_data, 0, 4);

    //Get the block number via a numeric input dialogue and save it in block_num
    do{
        clr_scr();
        move_cursor(0,0);
        puts("pls enter the block#");
        block_num = get_numeric(3,2,2,0);
    }while ((block_num < 0) || (block_num > 63));

    //Arrange the hints
    move_cursor(0,4);
    puts("writing");
    move_cursor(0,5);
    printf(" %02X %02X %02X %02X to #%02ld",
        p_data[0],p_data[1],p_data[2],p_data[3], block_num);

    //Here's some real stuff:
    //1) turn the interface on:
    InitMC632as15693();
    MC632ResetAllCard();

    //2) wait for a while: about 200ms
    for(tcnt = 0; tcnt < 200; tcnt++)
        delay_1ms();

    //3) write it, without UID
    rc = Tag_it_write_block( block_num, p_data, NULL);
    ISO15693_WriteBlock(0x42, NULL, block_num, 0x01, 0x04, p_data)

    //4) and now, set him free.
    MC632Off();

    //5) return the result code no matter what.
    return rc>0?0:rc;
}
```