

# **SOFTWARE GUIDE**

## **RFID WAP4 LF**



## Table of contents

<b>1</b>	<b>INTRODUCTION .....</b>	<b>4</b>
1.1	ABOUT THIS MANUAL .....	4
1.2	RFID READERS SUPPORTED.....	4
1.3	STANDARDS AND TAGS SUPPORTED .....	4
1.4	PLATFORMS SUPPORTED .....	5
1.5	RFID SDK STRUCTURE .....	5
1.6	RFID PROCESS.....	6
<b>2</b>	<b>GETTING STARTED .....</b>	<b>7</b>
2.1	RFID DRIVER INSTALLATION .....	7
2.2	PROGRAMMING LANGUAGES .....	9
2.2.1	<i>Dotnet</i> .....	9
2.2.1.1	Development Platforms.....	9
2.2.1.2	Create a new project .....	9
2.2.1.3	RFID Driver Library overview .....	10
2.2.1.3.1	Namespace .....	10
2.2.1.3.2	RFIDDriver class .....	10
2.2.1.3.3	Check RFID Driver installation.....	10
2.2.1.3.4	Enable driver.....	10
2.2.1.3.5	Get COM Port number.....	10
2.2.1.3.6	Disable driver .....	10
2.2.1.4	LF Reader Library overview .....	11
2.2.1.4.1	Namespace .....	11
2.2.1.4.2	Main class .....	11
2.2.1.4.3	Open reader communication.....	12
2.2.1.4.4	Dialog with reader .....	12
2.2.1.4.5	Close reader communication.....	12
2.2.2	<i>Cpp</i> .....	13
2.2.2.1	Development Platforms.....	13
2.2.2.2	Create a new project .....	13
2.2.2.2.1	Including Header files .....	13
2.2.2.2.2	Adding libraries.....	13
2.2.2.3	RFID Driver Library overview .....	14
2.2.2.3.1	Namespace .....	14
2.2.2.3.2	RFIDDriver class .....	14
2.2.2.3.3	Check RFID Driver installation.....	14
2.2.2.3.4	Enable driver.....	14
2.2.2.3.5	Get COM Port number.....	14
2.2.2.3.6	Disable driver .....	14
2.2.2.4	LF Reader Library overview .....	15
2.2.2.4.1	Open reader communication.....	15
2.2.2.4.2	Dialog with reader .....	15
2.2.2.4.3	Close reader communication.....	15
<b>3</b>	<b>READER HIGH LEVEL API .....</b>	<b>16</b>
3.1	OPENREADER .....	16
3.2	CLOSEREADER.....	16
3.3	GETREADERVERSION.....	17
3.4	GETSERIALNUMBER .....	17
3.5	RESETREADER.....	18
3.6	GETCARDID .....	18
3.7	ISCONTINUESREADMODE .....	18
3.8	SETCONTINUESREADMODE.....	19
3.9	READCONTINUESREADMODE .....	19
3.10	READEPROM .....	20
3.11	WRITEEPROM.....	21
3.12	LOGIN .....	22
3.13	READDATA.....	23
3.14	WRITEDATA .....	24
3.15	PROTOCOL .....	25

3.16	GENERAL ERROR DESCRIPTION .....	26
<b>4</b>	<b>READER PROTOCOL.....</b>	<b>27</b>
4.1	OPERATING FEATURES .....	27
4.1.1	Continuous read mode .....	27
4.1.2	On request mode .....	27
4.2	ASCII PROTOCOL .....	27
4.3	SET OF COMMANDS .....	28
4.3.1	Introduction.....	28
4.3.2	Get module version .....	28
4.3.3	Read EEPROM registry .....	29
4.3.4	Write EEPROM registry .....	29
4.3.5	Include tag type management .....	30
4.3.6	Exlude tag type management .....	30
4.3.7	Get operation mode .....	31
4.3.8	Request continuous read mode .....	31
4.3.9	Abort continuous read mode.....	32
4.3.10	Read tag.....	32
4.3.11	Read data block .....	33
4.3.12	Write data block .....	34
4.3.13	Login .....	35
4.3.14	Reset .....	35
4.3.15	Read Q5 tag programmed as EM40x2 tag .....	36
4.3.16	Write Q5 tag programmed as EM40x2 tag.....	36
4.4	EEPROM MEMORY ORGANIZATION .....	37
4.4.1	Module identification code.....	37
4.4.2	Protocol configuration.....	38
4.4.3	Baud rate.....	38
4.4.4	Codification type .....	39
4.4.5	Operation mode .....	39
4.4.6	Single shot time out value .....	39
4.4.7	Protocol configuration 2.....	39
4.4.8	RF reset off time .....	40
4.4.9	RF reset recovery time .....	40
<b>5</b>	<b>TAG INFORMATION .....</b>	<b>41</b>
5.1	EM4100 (64 BITS), EM4102 (64 BITS), EM4200 (128 BITS) .....	41
5.2	EM4450/4550 (1 KBITS) .....	41
5.3	HITAG 1 (2 KBITS, 256 BYTES) .....	41
5.4	HITAG 2 (256 BITS, 32 BYTES).....	42

# 1 Introduction

## 1.1 About this manual

This manual is a handbook for the design and development of applications using the RFID SDK. Applications developed with this SDK allow controlling RFID LF readers supplied by Zebra Technologies and run on Zebra Technologies hand-held computers.

## 1.2 RFID readers supported

Applications developed using the Zebra Technologies RFID SDK control the following RFID reader

<b>WAP4 – LF – ID1</b>	
------------------------	--

## 1.3 Standards and tags supported

Tag	Serial number	Read block	Write block	Properties
<b>Hitag 1</b>	√ (4 byte)	√ (4 byte)	√ (4 byte)	64 * 4 bytes R/W
<b>Hitag 2</b>	√ (4 byte)	√ (4 byte)	√ (4 byte)	7 * 4 bytes R/W, PWD
<b>Hitag S</b>	√ (4 byte)	√ (4 byte)	√ (4 byte)	1, 8 or 64 * 4 bytes R/W
<b>EM4x02</b>	√ (5 byte)	-	-	5 bytes RO
<b>EM4x05 (ISO FDXB)</b>	√ (8 byte)	-	-	
<b>EM4x50</b>	√ (4 byte)	√ (4 byte)	√ (4 byte)	32 * 4 bytes R/W, PWD
<b>EM4200</b>	√ (5 byte)	-	-	5 bytes RO
<b>Q5</b>	√ (5 byte)	√ (4 byte)	√ (4 byte)	8 * 4 bytes RW, PWD
<b>5567</b>	√ (8 byte)			

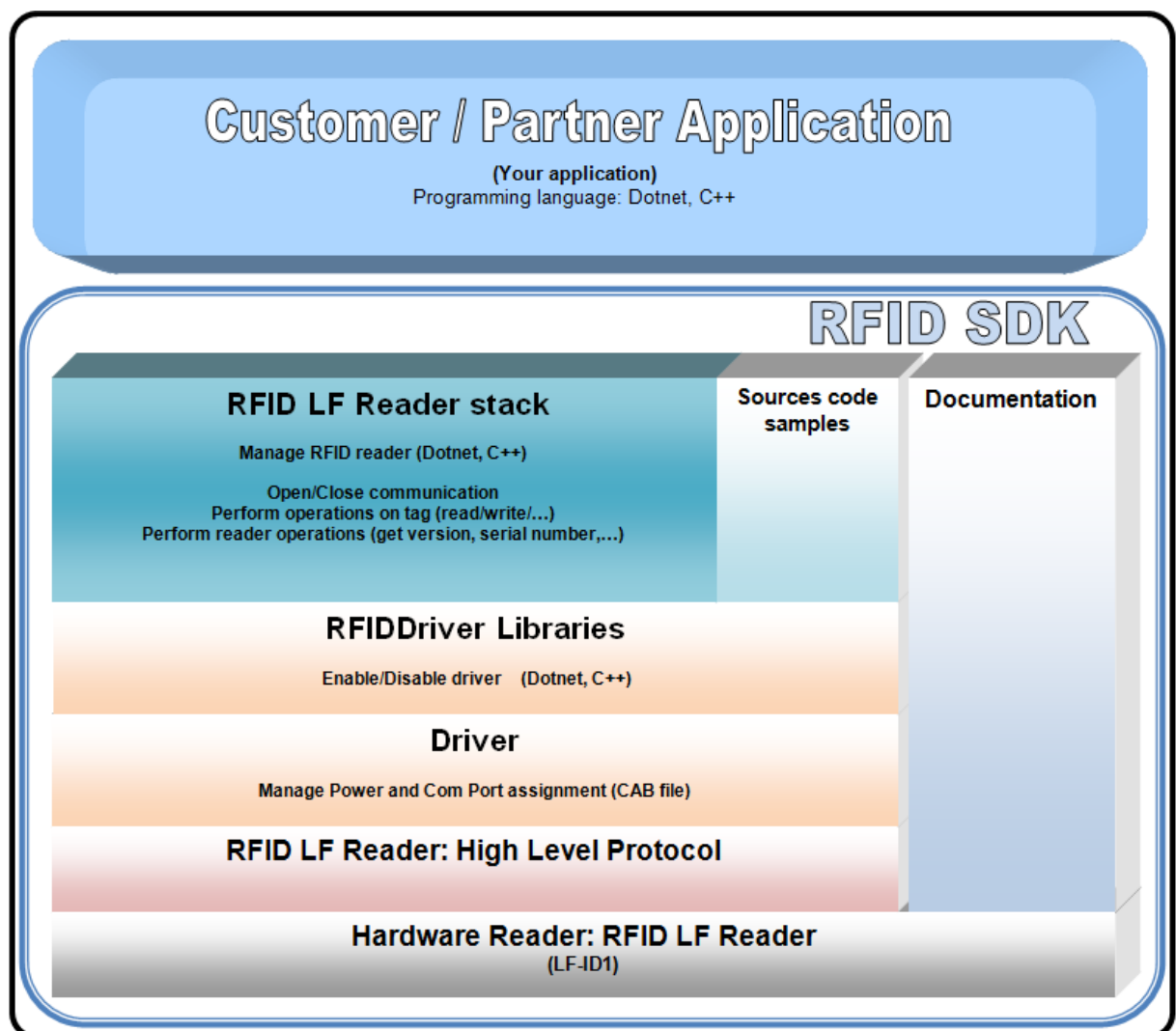
## 1.4 Platforms supported

This SDK is dedicated to Zebra Technologies WorkAbout Pro 4 handheld computer running under Windows Mobile 6.5 or Windows CE 6.

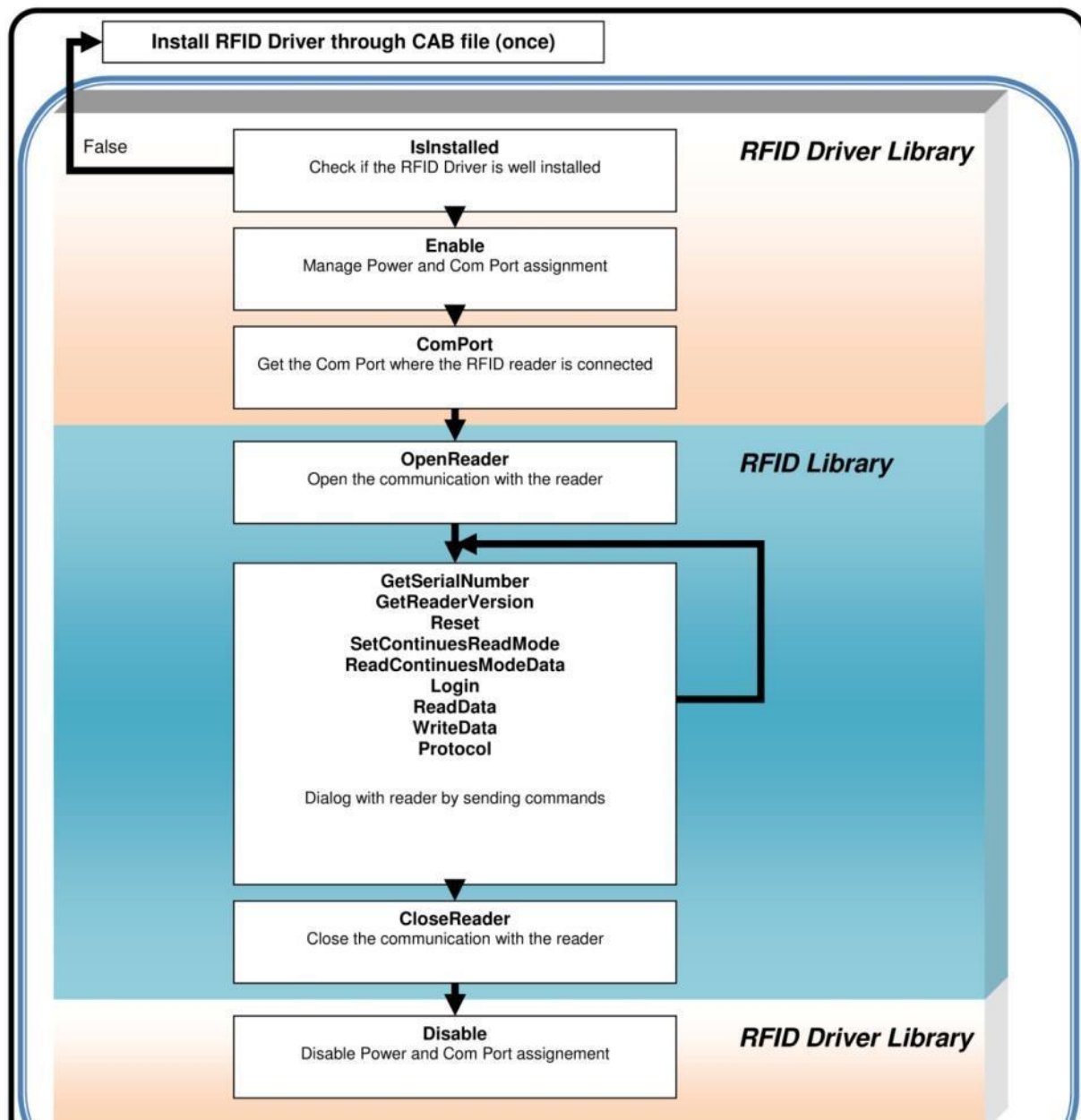
## 1.5 RFID SDK Structure

RFID SDK is based on different stacks.

- ➔ RFID Driver library stack allows turning on/off the power and assigning the com port assignment
- ➔ RFID LF Reader stack communicates with the reader and dialogs with tags



## 1.6 RFID Process



## 2 Getting Started



This handbook applies to the reader firmware version:

**LF3025 2.10dmo**

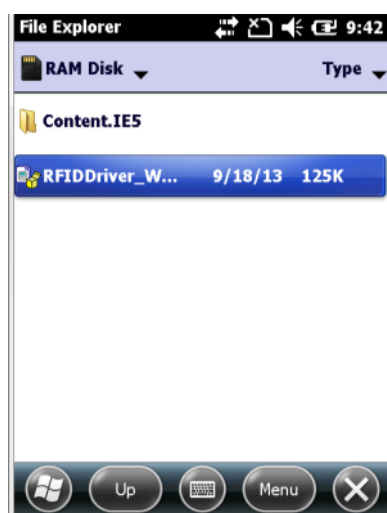
### 2.1 RFID Driver installation

The installation of RFID Driver is necessary. The driver is provided into the SDK package as CAB file.

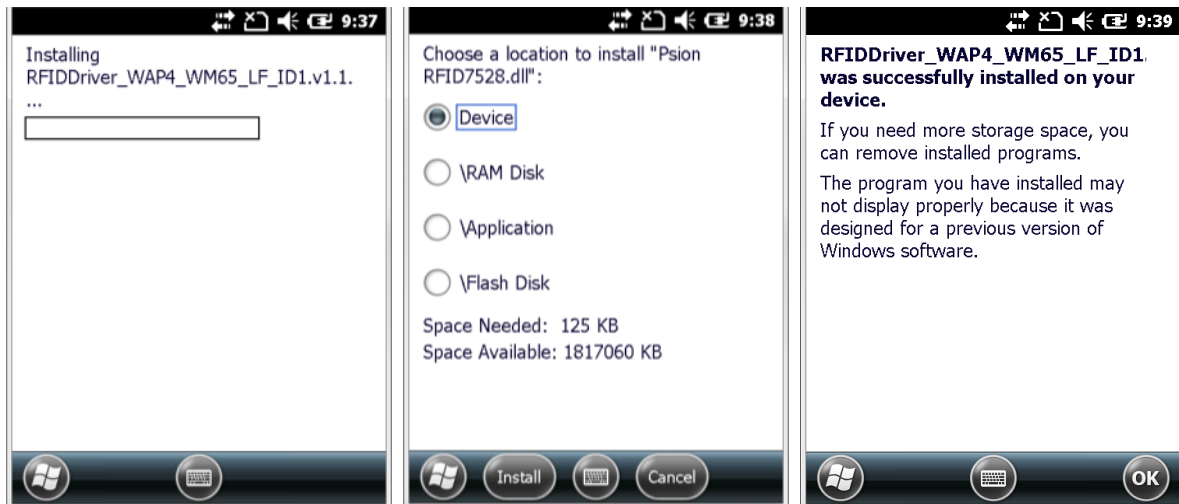
The first step is to select the right CAB file according to your operating system WinCE6 or WinWM6.5.

 RFIDDriver_WAP4_CE6_LF_ID1.v1.1.CAB	18/09/2013 12:08	Archive WinRAR	128 Ko
 RFIDDriver_WAP4_WM65_LF_ID1.v1.1.CAB	18/09/2013 12:08	Archive WinRAR	126 Ko

Then copy it from PC SDK package on Zebra Technologies WAP4 via ActiveSync.

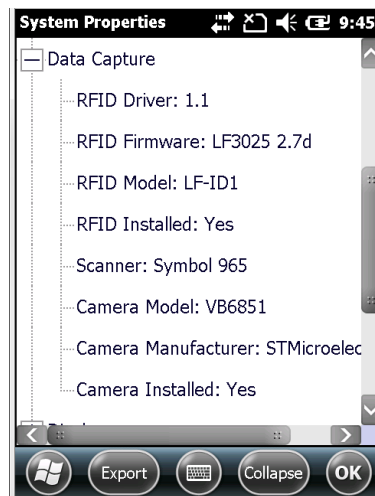


Then by double clicking on the CAB file, you should be able to install it.



**Note:** An unsuccessful installation occurs when the WAP4 can't dialog with the RFID reader. The issue may result because of the RFID reader is not mounted into the WAP4 OR the RFID reader is badly connected.

Finally, after installation, all the RFID reader information are available under Systems Properties->Data Capture





## 2.2 Programming languages

This manual describes the reader protocol.

API libraries are available for the following programming languages:

- ➔ Dotnet (C# / VB.NET)
- ➔ C/C++

Note: Other programming languages can be supported by using the reader protocol through the RFID driver.

### 2.2.1 Dotnet

The dotnet library is a high level API of FW protocol described on [section 4](#).

#### 2.2.1.1 Development Platforms

The following development platforms should be used for developing .NET:

- Microsoft Visual Studio 2005
- Microsoft Visual Studio 2008

#### 2.2.1.2 Create a new project

The following simple steps describe what is required to begin developing a .NET application by using the Zebra Technologies SDK in Visual Studio 2005 or 2008.

These are only the steps specific to incorporating the SDK; it is assumed the user is familiar with the Visual Studio environment.

In addition, sample .NET projects can be found in the **\Source code samples\Dotnet** subdirectory of your Zebra Technologies SDK installation.

#### Linking to the Zebra Technologies SDK Library

1. After creating a new project, open the Project menu and select **Add Reference...**
2. In the Browse tab, use the navigation tools to browse the **\Driver\dotnet** subdirectory of your **Zebra Technologies SDK** installation
3. Select the **RFIDDriver.DLL** and click **OK**
4. In the Browse tab, use the navigation tools to browse the **\Dotnet\CF2** or **\Dotnet\CF35** subdirectory of your **Zebra Technologies SDK** installation (according the usage of Compact Framework 2 or 3.5)
5. Select the **LF\_ID\_NET\_API.DLL** and click **OK**

### 2.2.1.3 RFID Driver Library overview

#### 2.2.1.3.1 Namespace

```
using Psion.RFID;
```

#### 2.2.1.3.2 RFIDDriver class

The RFID Driver class is the main object for piloting the RFID Driver.

```
RFIDDriver _rfidDriver= new RFIDDriver ();
```

#### 2.2.1.3.3 Check RFID Driver installation

```
// check if the driver is installed  
if (!_rfidDriver.IsInstalled)  
    throw new Exception("RFID Driver is not installed");
```

#### 2.2.1.3.4 Enable driver

Enable the RFID Driver, it manages the power (turn ON) and com port assignment.

```
// check and enable the driver  
// it means to power ON the module  
// assign com port number  
if (!_rfidDriver.IsEnabled) _rfidDriver.Enable();  
  
// necessary time to load the driver  
Thread.Sleep(250);
```

Throw an exception in case of error. Catch RFIDDriverException for more details

#### 2.2.1.3.5 Get COM Port number

Once enabled the RFID driver returns the com port number from where the communication with the reader has to be established.

```
// open the communication with the reader  
EERROR eerror = _reader.OpenReader(_rfidDriver.ComPort);
```

#### 2.2.1.3.6 Disable driver

Disable the RFID Driver, it manages the power (turn OFF) and com port assignment (suppress all the assignments).

```
// disable rfid driver  
// it means to destroy com port created  
// and turn OFF the power on the module  
if (_rfidDriver != null && _rfidDriver.IsInstalled && _rfidDriver.IsEnabled)  
    _rfidDriver.Disable();
```

Throw an exception in case of error. Catch RFIDDriverException for more details

### 2.2.1.4 LF Reader Library overview

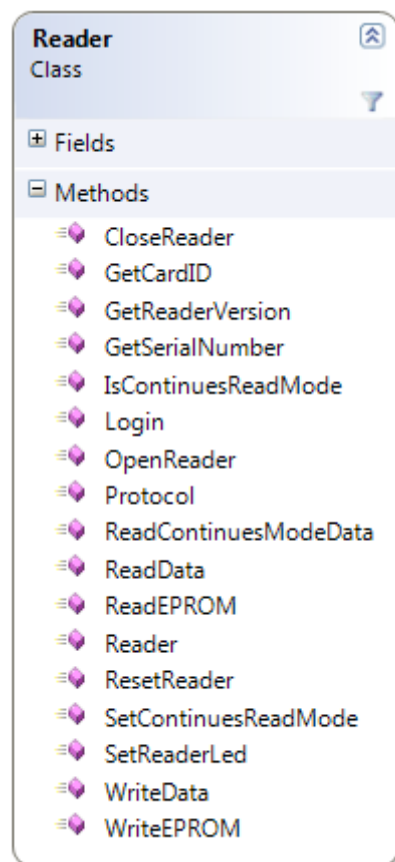
#### 2.2.1.4.1 Namespace

```
using Psion.RFID.LF;
```

#### 2.2.1.4.2 Main class

The reader class is the main object for dialoguing with the reader.

```
// instantiation of LF reader  
Reader _reader = new Reader();
```



#### 2.2.1.4.3 Open reader communication

```
// open the communication with the reader
EERROR error = _reader.OpenReader(_rfidDriver.ComPort);
if (error != EERROR.ER_OK) throw new Exception("Open reader failed: " +
error.ToString());
```

#### 2.2.1.4.4 Dialog with reader

Refer to [section 3](#) to know all the functions available.

Here an example how to get FW version information

```
// check the dialog with the reader by getting the FW version
byte[] arrReaderVer = null;
EERROR error = _reader.GetReaderVersion(ref arrReaderVer);
if (error != EERROR.ER_OK) throw new Exception("Dialog with reader failed: " +
error.ToString());
```

#### 2.2.1.4.5 Close reader communication

```
// close reader connection
if (_reader != null) _reader.CloseReader();
```

## 2.2.2 Cpp

The cpp library is a high level API of FW protocol described on [section 4](#).

### 2.2.2.1 Development Platforms

The following development platforms should be used for developing c++ applications:

- Microsoft Visual Studio 2005
- Microsoft Visual Studio 2008

### 2.2.2.2 Create a new project

The following simple steps describe what is required to begin developing a C++ application by using the Zebra Technologies SDK in Visual Studio 2005 or 2008.

These are only the steps specific to incorporating the SDK; it is assumed the user is familiar with the Visual Studio environment.

In addition, sample cpp projects can be found in the **\Source code samples\Cpp** subdirectory of your Zebra Technologies SDK installation.

#### 2.2.2.2.1 Including Header files

1. In the opening #include statements of your main program, add the following:

```
#include <RFIDDriver.h>
#include <RFIDIOCTL.h>
#include <Error.h>
#include <LF_ID_C_API.h>
```

2. Open the Project menu and select **[Project Name] Properties....**
3. Select **Configuration Properties > C/C++ > General** from the tree structure on the left.
4. Click in the field next to **Additional Include Directories** and add the **[...]\Driver\Cpp** from **SDK** installation path
5. Click in the field next to **Additional Include Directories** and add the **[...]\Cpp\Include** from **SDK** installation path
6. Click OK.

#### 2.2.2.2.2 Adding libraries

1. Open the Project menu and select **[Project Name] Properties....**
2. Select **Configuration Properties > Linker > Input** from the tree structure on the left.
3. Click in the field next to **Additional Dependencies** and add the **[...]\Driver\Cpp\RFIDDriver.lib** from **SDK** installation path
4. Click in the field next to **Additional Dependencies** and add the **[...]\Cpp\Lib\LF\_ID\_C\_API.lib** from **SDK** installation path
5. Click OK.

Important Note: The LF\_ID\_C\_API.DLL has to be copied on application folder into WAP4

### 2.2.2.3 RFID Driver Library overview

#### 2.2.2.3.1 Namespace

```
using namespace Psion::RFID;
```

#### 2.2.2.3.2 RFIDDriver class

The RFID Driver class is the main object for piloting the RFID Driver.

```
RFIDDriver m_rfidDriver;
```

#### 2.2.2.3.3 Check RFID Driver installation

```
// Check if the RFID driver is installed
if (! m_rfidDriver.IsInstalled())
{
    MessageBoxW(NULL,L"RFID Driver is not installed.",L"RFID
Driver",MB_OK|MB_ICONEXCLAMATION);
    return FALSE;
}
```

#### 2.2.2.3.4 Enable driver

Enable the RFID Driver, it manages the power (turn ON) and com port assignment.

```
// enable rfid driver - power ON the reader & get com port assignment
m_rfidDriver.Enable();
```

#### 2.2.2.3.5 Get COM Port number

Once enabled the RFID driver returns the com port number from where the communication with the reader has to be established.

```
// open the communication with the reader
SDKHANDLE hReader = OpenReader(m_rfidDriver.ComPort());
```

#### 2.2.2.3.6 Disable driver

Disable the RFID Driver, it manages the power (turn OFF) and com port assignment (suppress all the assignments).

```
// Disable RFID driver - Power OFF
if (m_rfidDriver.IsInstalled() && m_rfidDriver.IsEnabled())
    m_rfidDriver.Disable();
```

## 2.2.2.4 LF Reader Library overview

### 2.2.2.4.1 Open reader communication

```
SDKHANDLE hReader = OpenReader(m_rfidDriver.ComPort());
```

### 2.2.2.4.2 Dialog with reader

Refer to [section 3](#) to know all the functions available.

Here an example how to reset the reader

```
if(hReader != NULL)  
    ResetReader(hReader);
```

### 2.2.2.4.3 Close reader communication

```
CloseReader(hReader);
```

## 3 Reader High Level API

### 3.1 OpenReader

Main proposal of this function is to initialize and check if the LF reader is already connected to the serial host.

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR OpenReader(int comPort)</code>	
<b>Parameters</b>	ComPort	Integer – com port number got from RFID Driver class
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<code>SDKHANDLE OpenReader(UINT nPort)</code>	
<b>Parameters</b>	Port	Integer – com port number got from RFID Driver class
<b>Return value</b>	Reader handle if success NULL in case of error	

### 3.2 CloseReader

This function ends communication with the RFID device.

	Dotnet	
<b>Declaration Syntax</b>	<code>void CloseReader()</code>	
<b>Parameters</b>		
<b>Return value</b>		

	C++	
<b>Declaration Syntax</b>	<code>Void CloseReader(SDKHANDLE hReader);</code>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
<b>Return value</b>		



### 3.3 GetReaderVersion

This function returns the software version of the reader

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR GetReaderVersion(ref byte[] arrReaderVer)</code>	
<b>Parameters</b>	ReaderVer	Reference to byte array – Data returned
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<code>EERROR GetReaderVersion(SDKHANDLE hReader, BYTE* pData, UINT&amp; nSize)</code>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	Data	Reference to byte array – Data returned
	Size	Reference to length of data
<b>Return value</b>	refer <a href="#">General error description</a>	

### 3.4 GetSerialNumber

This function returns the serial number of the reader.

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR GetSerialNumber(ref byte[] arrSerial)</code>	
<b>Parameters</b>	Serial	Reference to byte array – Data returned
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<code>EERROR GetSerialNumber(SDKHANDLE hReader, BYTE* pData, UINT&amp; nSize)</code>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	Data	Reference to byte array – Data returned
	Size	Reference to length of data
<b>Return value</b>	refer <a href="#">General error description</a>	

### 3.5 ResetReader

This function will perform a software reset on the firmware

	Dotnet
<b>Declaration Syntax</b>	<code>public EERROR ResetReader()</code>
<b>Parameters</b>	
<b>Return value</b>	refer <a href="#">General error description</a>

	C++		
<b>Declaration Syntax</b>	<code>EERROR ResetReader(SDKHANDLE hReader)</code>		
<b>Parameters</b>	<table> <tr> <td>Reader</td><td>SDKHANDLE – Handle got from previous “OpenReader”</td></tr> </table>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
Reader	SDKHANDLE – Handle got from previous “OpenReader”		
<b>Return value</b>	refer <a href="#">General error description</a>		

### 3.6 GetCardID

This function returns the card ID of the tag in the RF field

	Dotnet		
<b>Declaration Syntax</b>	<code>public EERROR GetCardID(ref byte[] arrCardID)</code>		
<b>Parameters</b>	<table> <tr> <td>CardId</td><td>Reference to byte array – Data returned</td></tr> </table>	CardId	Reference to byte array – Data returned
CardId	Reference to byte array – Data returned		
<b>Return value</b>	refer <a href="#">General error description</a>		

	C++						
<b>Declaration Syntax</b>	<code>EERROR GetCardID(SDKHANDLE hReader, BYTE* pData, UINT&amp; nSize)</code>						
<b>Parameters</b>	<table> <tr> <td>Reader</td><td>SDKHANDLE – Handle got from previous “OpenReader”</td></tr> <tr> <td>Data</td><td>Reference to byte array – Data returned</td></tr> <tr> <td>Size</td><td>Reference to length of data</td></tr> </table>	Reader	SDKHANDLE – Handle got from previous “OpenReader”	Data	Reference to byte array – Data returned	Size	Reference to length of data
Reader	SDKHANDLE – Handle got from previous “OpenReader”						
Data	Reference to byte array – Data returned						
Size	Reference to length of data						
<b>Return value</b>	refer <a href="#">General error description</a>						

### 3.7 IsContinuesReadMode

This function will perform a software reset on the firmware

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR IsContinuesReadMode(ref bool bStatus)</code>	
<b>Parameters</b>	Status	Reference to status state – return True if continuous read is running else False
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<code>EERROR IsContinuesReadMode(SDKHANDLE hReader, BYTE&amp; bStatus)</code>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	Status	Reference to status – return True if continuous read is running else False
<b>Return value</b>	refer <a href="#">General error description</a>	

### 3.8 SetContinuesReadMode

Calling this function we will turn on / turn off continuous read mode.

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR SetContinuesReadMode(bool bStatus)</code>	
<b>Parameters</b>	Status	Boolean. true or false to activate or deactivate the continuous read mode
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<code>EERROR SetContinuesReadMode(SDKHANDLE hReader, BYTE bStatus)</code>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	Data	Byte with reader mode status: <ul style="list-style-type: none"> <li>0 – turn off</li> <li>1 – turn on</li> </ul>
<b>Return value</b>	refer <a href="#">General error description</a>	

### 3.9 ReadContinuesReadMode

Receive data from continuous read mode.

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR ReadContinuesModeData(ref byte[] arrCardID)</code>	
<b>Parameters</b>	CardId	Reference to byte array – Data returned
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<code>EERROR ReadContinuesModeData(SDKHANDLE hReader, BYTE* pData, UINT&amp; nSize)</code>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	Data	Reference to byte array – Data returned
	Size	Reference to length of data
<b>Return value</b>	refer <a href="#">General error description</a>	

### 3.10 ReadEPROM

Read the memory organization of the module. Reader configuration is located on EEPROM.

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR ReadEPROM(byte bPage, ref byte bData)</code>	
<b>Parameters</b>	Page	Byte representing the EEPROM page
	Data	Reference to byte – Data returned
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	EERROR ReadEPROM(SDKHANDLE hReader, BYTE bPage, BYTE& bData)	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	Page	Byte representing the EEPROM page
	Data	Reference to byte – Data returned
<b>Return value</b>	refer <a href="#">General error description</a>	

### 3.11 WriteEPROM

Write the memory organization of the module. Reader configuration is located on EEPROM.

	Dotnet	
<b>Declaration Syntax</b>	public EERROR WriteEPROM(byte bPage, byte bData)	
<b>Parameters</b>	Page	Byte representing the EEPROM page
	Data	Byte representing the new value for the specified page
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	EERROR WriteEPROM (SDKHANDLE hReader, BYTE bPage, BYTE bData);	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	Page	Byte representing the EEPROM page
	Data	Byte representing the new value for the specified page
<b>Return value</b>	refer <a href="#">General error description</a>	

## 3.12 Login

This function is needed to prepare ReadData/WriteData operation when data on the RFID transponder is password-protected.

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR Login(byte[] arrLogin)</code>	
<b>Parameters</b>	Login	Byte array with login information
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<code>EERROR Login(SDKHANDLE hReader, BYTE* pLog, UINT nLog)</code>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	pLog	Pointer to byte array containing login info. If there’s no login info this parameter will be null
	nLog	Size of login data
<b>Return value</b>	refer <a href="#">General error description</a>	

### 3.13 ReadData

This function reads data from the RFID transponder.

In operation mode, a [GetCarId](#) is required before to read for selecting the tag.

According to the tag, a [Login](#) can be necessary.

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR ReadData(UInt32 nOffset, UInt32 nNrSect, ref byte[] arrData)</code>	
<b>Parameters</b>	Offset	UInt32 – Offset of start reading block
	NrSect	UInt32 - Number sectors to read
	Data	Reference to byte array – Data returned
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<code>EERROR ReadData(SDKHANDLE hReader, UINT nOffset, UINT nLength, BYTE* pData, UINT&amp; nData)</code>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	Offset	Buffer contains 4 bytes – number blocks to read/write (as unsigned long)
	Length	Size of data buffer – 4 bytes
	pData	Reference to byte array – Data returned
	nData	Reference to length of data
<b>Return value</b>	refer <a href="#">General error description</a>	

### 3.14 WriteData

This function writes data into the RFID transponder.

In operation mode, a [GetCarId](#) is required before to write for selecting the tag.

According to the tag, a [Login](#) can be necessary.

	Dotnet	
<b>Declaration Syntax</b>	<code>public EERROR WriteData(UInt32 nOffset, UInt32 nNrSect, byte[] arrData)</code>	
<b>Parameters</b>	Offset	UInt32 – Offset of start reading block
	NrSect	UInt32 - Number sectors to read
	Data	Byte array - Data to write
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<code>EERROR WriteData(SDKHANDLE hReader, UINT nOffset, UINT nLength, BYTE* pData, UINT nData)</code>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	Offset	Buffer contains 4 bytes – number blocks to read/write (as unsigned long)
	Length	Size of data buffer – 4 bytes
	pData	Reference to byte array – Data to write
	nData	Data size
<b>Return value</b>	refer <a href="#">General error description</a>	



### 3.15 Protocol

This functions allows sending [protocol command](#) directly to the reader.

	Dotnet	
<b>Declaration Syntax</b>	<pre>public EERROR Protocol(byte[] arrCmdIn, byte[] arrDataIn, ref byte[] arrDataOut , uint nRetry)</pre>	
<b>Parameters</b>	CmdIn	Byte array – Command defined on Protocol section
	DataIn	Byte array – Data of the command
	DataOut	Reference to byte array – Data returned
	Retry	Byte – Number of retry before to return an error By default 10
<b>Return value</b>	refer <a href="#">General error description</a>	

	C++	
<b>Declaration Syntax</b>	<pre>EERROR Protocol(SDKHANDLE hReader, BYTE* pDataIn, UINT nDataIn, BYTE* pDataOut, UINT&amp; nDataOut)</pre>	
<b>Parameters</b>	Reader	SDKHANDLE – Handle got from previous “OpenReader”
	pDataIn	Pointer to byte array containing command and data from Protocol section
	nDataIn	Size of pDataIn
	pDataOut	Reference to byte array – Data returned
	nDataOut	Reference to length of data
<b>Return value</b>	refer <a href="#">General error description</a>	

## 3.16 General error description

One of the following error codes is returned by every function:

ERROR	DESCRIPTION
ER_OK	No error
ER_INVALID_POINTER	Invalid pointer
ER_OPEN	Error by opening serial com
ER_CLOSE	Error by clsing the serial com
ER_WRITE	Error by writing data
ER_READ	Error by reading data
ER_SET_SPEED	Error by setting speed
ER_SET_TIMEOUT	Error by setting timeout
ER_INVALID_DATA	Invalid data
ER_SIZE	Error wrong size
ER_NO_TRANSPORT	No transport layer
ER_MORE_DATA	
ER_TAG_DATA	
ER_EXEC_CMD	Error exec command
ER_NO_TAG	No tag detected
ER_UNKNOWN	Unknown error

## 4 Reader Protocol

Reader protocol is the lower level of communication with the LF reader module. Basically, [reader high level API](#) is an overlay of this protocol. The reading of this section will help to better understand all the commands.

Protocol command can be sent via [Protocol](#) function of API

### 4.1 Operating features

The reader can operate in two operation modes: the 'continuous read' mode and the 'on request' mode.

Specific commands of the communication protocol are foreseen to allow the transition between these two operation modes.

By default the RFID LF module is 'On request' mode

#### 4.1.1 Continuous read mode

'Continuous read' mode: when a tag is identified, the **RFID LF** module transmits its serial number on the serial line. While the tag is present and identified, the module continues to transmit its serial number approximately every 60 ms. The repetition rate can be modified through the 'single shot' and 'single shot time out value' parameters (refer [EEPROM Memory Organization](#)).

#### 4.1.2 On request mode

'On request' mode: the **RFID LF** module executes readily the received command, the set of commands are defined in the communication protocol.

## 4.2 ASCII Protocol

The communication with the reader is done through serial interface. The data exchanges between device and reader is formatted in ASCII.

The ASCII protocol has been designed for easy handling: the commands can be issued using a terminal program (HYPERTERMINAL for example) and the answers of the **RFID LF** module can be displayed by the same program.

The communication parameters are: selectable baud rate, 8 data bits, no parity bit (none) and 1 stop bit.

Generally the numeric value of a byte is expressed in hexadecimal (0x00 ... 0xFF) and are transmitted with two ASCII characters (examples: 0x00 → '0' '0', 0x0A → '0' 'A', 0xFF → 'F' 'F').

## 4.3 Set of commands

### 4.3.1 Introduction

The following table lists all the commands of the **RFID LF** module. Each command returns an answer to the host. If a wrong command has been received, the module answers with '?'.

#### Set of commands

command	Description
'v'	To get module version
'rp' / 'wp'	To read / write EEPROM register
'o+' / 'o-'	To include / exclude tag type management
'l'	To get operation mode
'c'	To request continuous read mode
's'	To read a tag (single request)
'r' / 'rb' / 'w' / 'wb'	To read / write a data block of the tag
'l'	To work with a password protected tag
'x'	To reset the module
'qr' / 'qw'	To read / write the Q5 tag programmed as EM40x2 tag

Note: commands can be issued also using capital letters.

### 4.3.2 Get module version

The command 'get module version' gets back the data concerning the module version.

command	note
'v'	---

answer	note
'd' ... 'd'	10 characters string 'L' 'F' '3' '0' '2' '5' ' ' 'v' '.' 'v' 'd' with 'v' '.' 'v' the version (example: '1' '.' '0')

### 4.3.3 Read EEPROM registry

The command 'read EEPROM register' gets back the value of the specified register.

command	note
'r' 'p'	---
'a' 'a'	register address 0x00 ... 0xEF → '0' '0' ... 'E' 'F'

Answers:

a) successful operation:

answer	note
'd' 'd'	read value 0x00 ... 0xFF → '0' '0' ... 'F' 'F'

b) address error:

answer	Note
'R'	---

### 4.3.4 Write EEPROM registry

The command 'write EEPROM register' sets the value of the specified register.

command	note
'w' 'p'	---
'a' 'a'	register address 0x00 ... 0xEF → '0' '0' ... 'E' 'F'
'd' 'd'	value to set 0x00 ... 0xFF → '0' '0' ... 'F' 'F'

Answers:

a) successful operation:

answer	note
'd' 'd'	written value 0x00 ... 0xFF → '0' '0' ... 'F' 'F'

Note: these changes will only be taken into effect after a reset of the module.

b) address error:

answer	note
'R'	---

c) write error:

answer	note
'F'	---

### 4.3.5 Include tag type management

The command 'include tag type' adds the specified tag type to the list of the tag types managed at this time.

#### command note

'o' '+'	---
't'	tag type to include
	't' = 'U' for EM40x2, EM4200
	't' = 'Q' for Q5 tag
	't' = 'M' for 5567 (Atmel/Temic)
	't' = 'T' for EM4450
	't' = 'h' for Hitag 1, Hitag S
	't' = 'H' for Hitag 2
	't' = 'Z' for ISO FDX

#### answer note

'o' '+'	---
't'	included tag type
	't' = 'U' for EM40x2 tag
	...

### 4.3.6 Exclude tag type management

The command 'exclude tag type' removes the specified tag type from the list of the tag types managed at this time.

#### command note

'o' '-'	---
't'	tag type to exclude
	't' = 'U' for EM40x2, EM4200
	't' = 'Q' for Q5 tag
	't' = 'M' for 5567 (Atmel/Temic)
	't' = 'T' for EM4450
	't' = 'h' for Hitag 1, Hitag S
	't' = 'H' for Hitag 2
	't' = 'Z' for ISO FDX

#### answer note

'o' '-'	---
't'	Excluded tag type
	't' = 'U' for EM40x2 tag
	...

### 4.3.7 Get operation mode

The command gets back the operation mode that is in progress: 'continuous read' or 'on request'. Note that this command do not stop the 'continuous read' mode if it is in progress.

command	note
---------	------

'!	---
----	-----

Answers:

a) 'continuous read' mode in progress

answer	note
--------	------

'!	'continuous read' mode in progress
----	------------------------------------

b) 'continuous read' mode not in progress ('on request' mode in progress)

answer	note
--------	------

'F'	'continuous read' mode not in progress (→ 'on request' mode in progress)
-----	---

### 4.3.8 Request continuous read mode

The command 'request continuous read mode' sets the module in 'continuous read' operation mode.

command	note
---------	------

'c'	---
-----	-----

Answer of the module when a tag is identified:

answer	note
--------	------

't'	tag type 't' = 'U' for EM40x2, EM4200 't' = 'Q' for Q5 tag 't' = 'M' for 5567 (Atmel/Temic) 't' = 'T' for EM4450 't' = 'h' for Hitag 1, Hitag S 't' = 'H' for Hitag 2 't' = 'Z' for ISO FDX
's' ... 's'	serial number of the tag example: for EM40x2 and Q5 tags the serial number has 5 bytes giving 10 ASCII characters 0x02 0x60 0x4A 0x9B 0x58 → '0' '2' '6' '0' '4' 'A' '9' 'B' '5' '8'

Note that when the tag is present the message is transmitted again following the setting of the parameters 'single shot' (0x0B register 'protocol configuration') and 'single shot time out value' (0x0F register).

### 4.3.9 Abort continuous read mode

The command 'abort continuous read mode' sets the module in 'on request' operation mode. In function of the 'Noisy environment' parameter (0x10 register 'protocol configuration 2'), the command can assume two forms.

Commands:

a) 'Noisy environment' parameter not set

command	note
'x'	any character

b) 'Noisy environment' parameter set

command	note
'.'	---

Answer:

answer	note
'S'	---

### 4.3.10 Read tag

The command 'read tag' verifies the presence of a tag in the antenna rf field. This command is used when the module is in 'on request' operation mode. Execution time of the command: from 150 ms min to 320 ms max.

command	note
's'	---

a) tag is present

answer	note
't'	tag type 't' = 'U' for EM40x2, EM4200 't' = 'Q' for Q5 tag 't' = 'M' for 5567 (Atmel/Temic) 't' = 'T' for EM4450 't' = 'h' for Hitag 1, Hitag S 't' = 'H' for Hitag 2 't' = 'Z' for ISO FDX
's' ... 's'	serial number of the tag example: for EM40x2 and Q5 tags the serial number has 5 bytes giving 10 ASCII characters 0x02 0x60 0x4A 0x9B 0x58 → '0' '2' '6' '0' '4' 'A' '9' 'B' '5' '8'

b) no tag present

answer	note
'N'	---



### 4.3.11 Read data block

The command 'read data block' gets back the value of the bytes of the specified data block. The dimension of the data block depends on the tag type (example: for the Q5 tag, the data block has 4 bytes). Execution time of the command for the Q5 tag: from 50 ms min to 90 ms max.

Commands:

a) restricted form, do not use with block address greater than 0x40

command	note
'r'	---
'a' 'a'	data block address 0x00 ... 0x40 → '0' '0' ... '4' '0'

b) general form

command	note
'r' 'b'	---
'a' 'a'	data block address 0x00 ... 0xFF → '0' '0' ... 'F' 'F'

Answers:

a) successful operation

answer	note
'd' ... 'd'	data of the block for the Q5 tag the data block has 4 bytes giving 8 ASCII characters (example: 0x02 0x60 0x4A 0x9B → '0' '2' '6' '0' '4' 'A' '9' 'B')

b) address error

answer	note
'R'	---

c) read error

answer	note
'F'	---

d) no tag

answer	note
'N'	---

### 4.3.12 Write data block

The command 'write data block' sets the value of the bytes of the specified data block. The dimension of the data block depends on the tag type (example: for the Q5 tag, the data block has 4 bytes). Execution time of the command for the Q5 tag: from 60 ms min to 100 ms max.

Commands:

a) restricted form, do not use with block address greater than 0x40

Command	note
'w'	---
'a' 'a'	data block address 0x00 ... 0x40 → '0' '0' ... '4' '0'
'd' ... 'd'	data to be written in the block for the Q5 tag the data block has 4 bytes giving 8 ASCII characters (example: 0x02 0x60 0x4A 0x9B → '0' '2' '6' '0' '4' 'A' '9' 'B')

b) general form

command	note
'w' 'b'	---
'a' 'a'	data block address 0x00 ... 0xFF → '0' '0' ... 'F' 'F'
'd' ... 'd'	data to be written in the block for the Q5 tag the data block has 4 bytes giving 8 ASCII characters (example: 0x02 0x60 0x4A 0x9B → '0' '2' '6' '0' '4' 'A' '9' 'B')

Answers:

a) successful operation

answer	note
'd' ... 'd'	data of the block for the Q5 tag the data block has 4 bytes giving 8 ASCII characters (example: 0x02 0x60 0x4A 0x9B → '0' '2' '6' '0' '4' 'A' '9' 'B')

b) address error

answer	note
'R'	---

c) write error

answer	note
'F'	---

d) no tag

answer	note
'N'	---

### 4.3.13 Login

This command is usable with a password protected tag.

In the particular case of a Q5 tag, the tag must be opportunely programmed: the password must be set (data block 7) and the 'pwd' bit in the configuration (data block 0) must be set (for more information refer to the data sheet of the manufacturer). In this case, the commands to read/write a tag data block have to be transmitted with the password. The 'login' Command allows to load in the module the password that must be used to access the tag that is present in the field and enables the automatic inclusion of the password in the in the subsequent read/write commands. Use the 's' (read tag) or 'c' (request continuous read mode) command to disable the automatic inclusion of the password in the read/write commands.

command	note
'l'	---
'p' ... 'p'	password for the Q5 tag the password has 4 bytes giving 8 ASCII characters (example: 0x02 0x60 0x4A 0x9B → '0' '2' '6' '0' '4' 'A' '9' 'B')

answer	note
'L'	---

### 4.3.14 Reset

The command 'reset module' restarts the module like a power on reset.

command	note
'x'	---

Answer only if the parameter 'Disable startup message' is reset (0x10 register 'protocol configuration 2'):

answer	note
'd' ... 'd'	'A' 'S' 'E' '3' '0' '2' '5' ' ' 'x' '.' 'x' con 'x' '.' 'x' the version (example: '1' '.' '0')

### 4.3.15 Read Q5 tag programmed as EM40x2 tag

The command 'read Q5 tag programmed as EM40x2 tag' verifies the presence of a tag in the antenna rf field, the tag must be a Q5 tag working like an EM40x2 tag. Execution time of the command for the Q5 tag: from 150 ms min to 320 ms max.

command	note
---------	------

'q' 'r'	---
---------	-----

Answers:

a) successful operation

answer	Note
--------	------

'c' ... 'c'	Read code (EM40x2 format) with 5 bytes giving 10 ASCII characters (example: 0x02 0x60 0x4A 0x9B 0x58 → '0' '2' '6' '0' '4' 'A' '9' 'B' '5' '8')
-------------	--

b) no tag

answer	note
--------	------

'N'	---
-----	-----

c) not a Q5 tag (→ operation error)

answer	note
--------	------

'O'	---
-----	-----

d) not programmed Q5 tag (→ operation fail)

answer	note
--------	------

'F'	---
-----	-----

### 4.3.16 Write Q5 tag programmed as EM40x2 tag

The command 'write Q5 tag programmed as EM40x2 tag' sets a Q5 tag with the specified code in order to simulate an EM40x2 tag. . Execution time of the command for the Q5 tag: from 330 ms min to 500 ms max.

command	note
---------	------

'w' 'q'	---
'c' ... 'c'	code to be programmed EM40x2 format with 5 bytes giving 10 ASCII characters (example: 0x02 0x60 0x4A 0x9B 0x58 → '0' '2' '6' '0' '4' 'A' '9' 'B' '5' '8')

Answers:

a) successful operation

answer	note
--------	------

'c' ... 'c'	programmed code EM40x2 format with 5 bytes giving 10 ASCII characters (example: 0x02 0x60 0x4A 0x9B 0x58 → '0' '2' '6' '0' '4' 'A' '9' 'B' '5' '8')
-------------	---

b) no tag

answer	note
'N'	---

c) not a Q5 tag (→ operation error)

answer	note
'O'	---

d) not programmed Q5 tag (→ operation fail)

answer	note
'F'	---

## 4.4 EEPROM Memory Organization

The **RFID LF** module is equipped with an EEPROM memory of 240 bytes (or registers) that is described in the following table. Some registers hold the configuration parameters of the module that set the behavior of the module at start up.

**EEPROM memory organization**

Address (hex)	Description	Default	Note
0x00 ... 0x04	Module identification code	---	RO
0x05 ... 0x09	Reserved	---	
0x0A	Reserved	0x01	
0x0B	Protocol configuration	0x01	
0x0C	Baud rate	0x00	9600 bd
0x0D	Codification type	0x00	0x0D
0x0E	Operation mode	0x01	
0x0F	'single shot' time out value	0x0A	0.1s unit
0x10	Protocol configuration 2	0x00	
0x11	Reserved	---	
0x12	Reserved	---	
0x13	RF reset off time	0x0A	1ms unit
0x14	RR reset recovery time	0x0A	1ms unit
0x15	Reserved	---	
0x16	Reserved	---	
0x17	Reserved	---	
0x18	Reserved	---	
0x19	Reserved	---	
0x1A	Reserved	---	
0x1B	Reserved	---	
0x13	Reserved	---	
0x1C ... 0x1F	Reserved	---	
0x20 ... 0xEF	User data	---	

### 4.4.1 Module identification code

The 'module identification code' (registers 0x00 ... 0x04) is factory programmed and cannot be changed (Read Only).

### 4.4.2 Protocol configuration

The 'protocol configuration' (register 0x0B) specifies the general behavior of the module (default value 0x10).

**Protocol configuration (0x0B)**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
'0'	'0'	Single shot	Led	'0'	'0'	'0'	Auto start

- 'Auto start' (bit 0): if set to '1', the module will start up in 'continuous read' operation mode automatically. Default value is '0'
- 'Led' (bit 4): if set to '1', the module do not manage the LEDs, the user can o it through the foreseen commands. Default value is '1'
- 'Single shot' (bit 5): if set to '1', the module will transmit the serial number of an identified tag only one time within a specified time out specified in the register 0x0F.

### 4.4.3 Baud rate

The 'baud rate' (register 0x0C) defines the speed of the communication.

**Baud rate (0x0C)**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
'0'	'0'	'0'	'0'	'0'	'0'	a	b

- Three baud rates are defined.

a	b	baud rate
'0'	'0'	9600 (default)
'0'	'1'	19200
'1'	'0'	38400
'1'	'1'	---

#### 4.4.4 Codification type

The 'codification type' (register 0x0D) defines the bits arrangement of a byte.

Codification type (0x0D)							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
'0'	'0'	'0'	'0'	'0'	'0'	a	b

- Four types of codification are defined.

a	b	codification
'0'	'0'	ACG
'0'	'1'	SOKYMAT
'1'	'0'	BLUEBOX
'1'	'1'	X

#### 4.4.5 Operation mode

The 'operation mode' (0x0E register) defines which tag types the module will support.

Operation mode (0x0E)							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
'0'	'0'	Q5	'0'	'0'	'0'	'0'	EM40x2

- 'EM40x2' (bit 0): if set to '1' enables the management of the EM40x2 tag.
- 'Q5' (bit 5): if set to '1' enables the management of the Q5 tag.

#### 4.4.6 Single shot time out value

The 'single shot time out value' (0x0F register) defines the delay time between two responses of the module; it has only effect in 'continuous read' operation mode. To enable the time out, the 'single shot' flag has to be set (see the 'protocol configuration' register). The time unit is about 0.1s. The value 0x00 disables the feature (like the 'single shot' flag set to '0'), the value 0xFF disables the repetition (the time out value is infinite), the default value 0x0A means a repetition rate of 1s.

#### 4.4.7 Protocol configuration 2

The 'protocol configuration 2' (0x10 register) specifies additional general behavior of the module (default value 0x00).

Protocol configuration 2 (0x10)							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
'0'	'0'	'0'	'0'	Noisy envir.	'0'	Disable startup msg	'0'

- 'Disable startup message' (bit 1): if set to '1', the automatic message (module version) is suppressed at startup.
- 'Noisy environment' (bit 3): if set to '1', the 'continuous read' operation mode can only

be aborted by the command composed of the '.' Character (any other character is invalid).

#### **4.4.8 RF reset off time**

The 'rf reset off time' (0x14 register) specifies the field off time in 1ms unit, the default value 0x0A means 10ms off time.

#### **4.4.9 RF reset recovery time**

The 'rf reset recovery time' (0x15 register) specifies the recovery time in 1ms after the rf field is turned on, the default value 0x0A means 10ms recovery time.



## 5 Tag Information

### 5.1 EM4100 (64 bits), EM4102 (64 bits), EM4200 (128 bits)

This are read-only chip types, so you can only retrieve a UID.

### 5.2 EM4450/4550 (1 kbits)

Memory blocks (pages) of 32 bits/4 Bytes.

Block #	Hex Address	Access	Description
1	00	Read-only	Password, default 00000000h
2	01	Read-only	Protection Word
3	02	Read-only	Control Word
4	03	Read/write	User Memory
...	...	...	User Memory
31	1F	Read/write	User Memory
32	20	Read-only	Device Serial Number (UID)
33	21	Read-only	Device Identification

### 5.3 Hitag 1 (2 kbits, 256 bytes)

Memory blocks (pages) of 32 bits/4 Bytes.

Block #	Hex Address	Access	Description
1	00	Read-only	UID
2	01	Read/write	Configuration Word
3	02	No access	—
...	...	No access	—
16	0F	No access	—
17	10	Read/write	User Memory
...	...	...	User Memory
64	3F	Read/write	User Memory

## 5.4 Hitag 2 (256 bits, 32 bytes)

Memory blocks (pages) of 32 bits/4 Bytes.

Block #	Hex Address	Access	Description
1	00	Read-only	UID
2	01	Read/write	Password RWD, default 4D494B52h
3	02	No access	—
4	03	Read/write	Configuration Word, password protected, default PW: 0000 0000 h
5	04	Read/write	User Memory / 64 bits read-only memory layout for read-only emulation
6	05	Read/write	User Memory / 64 bits read-only memory layout for read-only emulation
7	06	Read/write	User Memory
8	07	Read/write	User Memory