Proceedings of the 9th International Conference on Applied Informatics
Eger, Hungary, January 29–February 1, 2014. Vol. 2. pp. 131–138
doi: 10.14794/ICAI.9.2014.2.131

Programming an RFID reader for getting data from different semi-passive sensor tags*

Sándor Király, Tibor Radványi, Csaba Biró
Péter Szigetváry, Péter Takács

Eszterhazy Karoly University of Applied Sciences
Institute of Mathematics and Informatics
{ksanyi,dream,birocs,szigipet,takip}@aries.ektf.hu

Abstract

Alien ALH-9010 is a PDA built on the Microsoft Windows Mobile 6.5 operating system and equipped with a full featured 900 MHZ RFID tag reader. It is able to do inventory of, selectively read from, and write to tags which support the EPC Class 1 Generation 2 protocol. The demo software developed by the producer suggests that this handheld enables users to deploy different inventory software for making inventory work faster and cost-effective.

Can this PDA be usable for purposes other than inventory and automatic identification? In this paper we demonstrate the way of developing .NET software that enables this handheld to be able to read any data including temperature data from any semi-passive temperature sensor tag working in UHF range, such as CAEN 927Z or CAEN Easy2Log if they support EPC Gen2 protocol. Using the developed software any data from any sensor tag can be read if knowledge of their storage protocol is provided.

Keywords: RFID, API, PDA programming, ALH-9010, CAEN tags, .NET, method, sensor tag

1. Introduction

Semi-passive (battery assisted backscatter) UHF radio frequency tags with on-chip temperature sensor are powered by a battery and can measure the ambient temperature, and put the measured data into their own memory. They are also capable of monitoring temperature sensitive products like perishable foods and

*Supported by: FutureRFID Development possibilities in the RFID/NFC technology TÁMOP-4.2.2.C-11/1/KONV-2012-0014

pharmaceuticals during transportation and storage.[1] These tags can be used with standard UHF RFID readers without requiring any additional equipment if both the tag and the reader are compatible with the EPCGlobal C1G2 and ISO18000-6C standards. [2] For reading no additional equipment is needed but a software that can read the logged values from the memory of the tags and can interpret and save these data. Generally, a transponder can only be read by the RFID reader and the demo software of the manufacturers. In our case we have two different types of transponder created by the same manufacturer but to be able to read them two different software are required. An RFID reader (a simple interface card) originating from this manufacturer can read these tags but it works only if it is connected to the RS232 port of a PC. Unfortunately this interface card requires not only a PC connection but electricity as well.

In this article we demonstrate how we can read these different transponders by a mobile RFID reader of another manufacturer, by developing software that makes the reader be able to read and interpret the logged values knowing the storage protocol of the tags.

2. Programming PDA equipped with RFID reader

Alien ALH-9010 mobile computer is a small, ergonomically designed PDA, built on the Microsoft Windows Mobile 6.5 operating system. This gadget contains a full-featured 900 MHz UHF RFID tag reader. It is able to do inventory of, and – what is particularly important – selectively read and write RFID tags which support the EPC Gen2 protocol. [3]

The PDA is equipped with not only an RFID transponder reader, but a microSD card reader, 1D or 2D barcode scanner, and optional an 3G wireless modem, GPS and camera modules. For the programming of this PDA the following software packages are needed:

- Development Environment: Visual Studio 2005 or 2008
- .NET Compact Framework: Version 2.0 or later
- SDK (for ALH-9010/9011 only): Windows Mobile Professional SDK 6.5 or later
- For Win7-8: Windows Mobile Device Center [4]

For the programming there are a couple of DLL files created by the manufacturer. This is very typical for any PDA, we need to call API procedures and functions. In case of this PDA first step in getting data from the transponder is the instantiation of the `RfidApi` class. This class can be found in the NRfidApi.dll. Then we only need to call the proper methods of this class.

To activate the reader we call the `PowerOn` method, to open the communication we call the `Open` method. The most useful methods for us (apart from the mentioned) are the `ReadMemBank` and `WriteMemBank` methods.

```
RFID_RESULT ReadMemBank(bool bSyncMode, MEM_BANK MemBank, UINT nWordPtr,
UINT nWordCount, BOOL bContinuous String AccessPassword, String MemBankData);
```

This method allows reading an arbitrary memory location of a tag in one-word units. (1 word-unit is two bytes, 16 bits in this case.) The `nWordPtr` parameter defines the starting position and the number of words to be read is stored in the fourth `NWordCount` parameter. The result value is always stored in a string variable (`MemBankData`). `WriteMembank` method works very similarly but it allows writing an arbitrary memory location of a tag in one-word units.

3. Storage protocol of sensor-tags

One of them is a Model A927Z that is compatible with EPC Class 1 Generation 2 standard and it has 16Kbyte of on-board flash memory for storage purposes (8000 samples) and can achieve a reading distance up to 10 meters. [5] The other semi-passive UHF RFID tag with temperature sensor is also compatible with EPC Class 1 Gen 2 protocol but it has only 8 Kbyte memory for samples. [6] The most important feature is the compatibility with EPC Class 1 Generation 2 for us. A Standard Gen2 memory has the following layout [7]:

Memory banks are defined as follows	
Bank 11b	USER
Bank 10b	TID
Bank 01b	EPC
Bank 00b	RESERVED

Figure 1: Standard EPC Class 1 Generation 2 memory

Obviously, temperature data is stored in the USER memory bank. There are only two unanswered question left. How do these tags store the temperature data and in which part of the USER memory bank?

If a detailed specification is available about the address layout and the interpretation of the data stored in these addresses, the process of implementing software for the PDA becomes easier. If it is not available, then resetting the tags with the RFID reader that came from the manufacturer can be a good solution. Dumping the content of the USER memory bank after resetting the tags and repeating this after taking one or two samples may also lead to the solution.

Figure 2 shows the content of the USER memory bank of A927 tag after resetting (Figure 2.a) and after 1 minute (Figure 2.b).

Analysing the content the following observations can be made:

At address 0x11 the tag stores the number of samples, in 0x1B the sampling frequency is stored, logging starts in the address of 0x17, the offset address of the last logged data is stored at address 0x17, and the temperature data is stored in 4 words. The first word works as a separator, the first two useful values are the date and the last word is the temperature.

Address (Hex)	0	1	2	3	4	5	6	7
0000	0002	0300	0007	2010	0300	2017	0110	2901
0008	0010	0601	0000	FFEC	0046	0000	00EC	52CB
0010	4380	0000	003C	0000	02BC	0000	0000	FFFF
0018	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF

a)

Address (Hex)	0	1	2	3	4	5	6	7
0000	0002	0300	0007	2010	0300	2017	0110	2901
0008	0010	0601	0000	FFEC	0046	0000	00EC	52CB
0010	4380	0001	003C	0000	02BC	00EB	0003	0000
0018	52E3	E0AC	00EB	FFFF	FFFF	FFFF	FFFF	FFFF
0020	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF

b)

Figure 2: User memory bank of A927Z after resetting (a) and after one minute (b) (Screenshot)

The Easy2 tag does not store temperature data in the first part of the USER memory bank. (Figure 3.a) It starts logging at address 0x8A and the last data is stored at address 0x67. (Figure 3. b) There are no separator zeroes and the first word is the temperature the last two are the date, so it uses only four words for logging.

Address (Hex)	0	1	2	3	4	5	6	7
0040	FFFF	FFFF	FFFF	0000	0000	0000	0000	0000
0048	0000	0000	0000	0000	0000	0000	0000	0000
0050	0000	0013	0000	0000	0000	0001	0001	0000
0058	0000	0000	0000	0000	0000	0000	0000	0000

a)

Address (Hex)	0	1	2	3	4	5	6	7
0080	0000	0000	0000	0000	0000	0000	0000	0000
0088	0000	0000	02F8	5DD6	52E6	02F8	5E12	52E6
0090	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF

b)

Figure 3: User memory bank of Easy2 (screenshot)

There is only one more thing we need to know: how do they interpret dates and temperatures. In both cases the DateTime is expressed in Unix time format. If the value is 0 that corresponds to midnight (UTC) January, 1 1970.

The A927Z stores the temperature tenfold unlike Easy2 that requires a special converter.

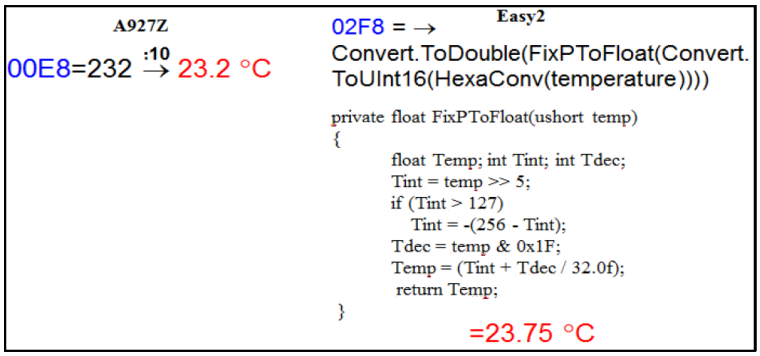


Figure 4: Getting temperature values from stored hexadecimal values

4. The developed software

The developed program not only saves the obtained data from the tags to the microD card but also displays the required data (for example the last temperature, the date of the last temperature, etc.), and if it is necessary it can send data in file to a PC.

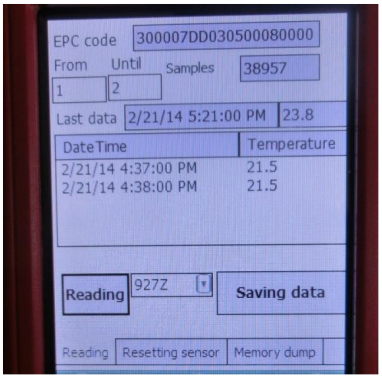


Figure 5: Screenshot of the developed software

The general class includes the variables (result, ReadType, etc.) that are needed for the `ReadMemBank` method of the Rfid class. The instantiation of the RFID API class is placed in the constructor of this class in order to be visible to the other classes. [8][9] To identify the type of the tag (`tagDetect` method) reading the TID memory bank seems to be the best solution where tags store their model number. To check whether our software works as intended a dump method is required that can display the content of any memory part of the selected tag.

The `readingFromTag` method reads the given number of values (maximum 255

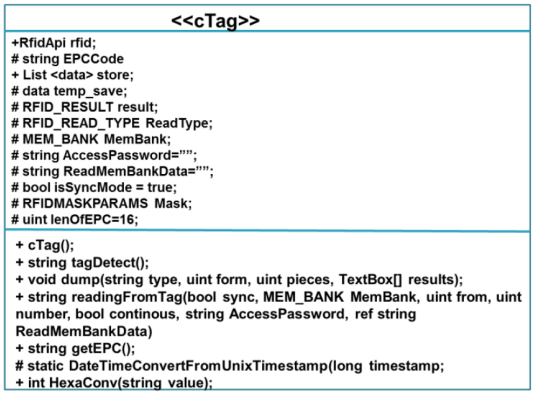
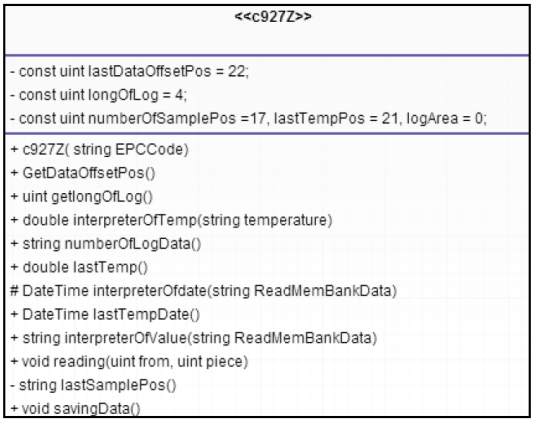
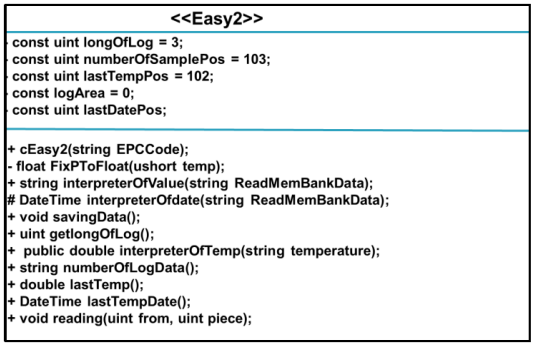


Figure 6: The general class



a)



b)

Figure 7: c927Z (a) and Easy2 (b) specific classes

words) from the given address of the tag and is used in the derived classes as well.

The class of c927Z includes the specific attributes of the tag as constants and also the tag specific methods (`interpreterofValue`, `interpreterOfDate`, `savingData`, `lastSamplePos`, etc). If another action is required, for example resetting the tag, the extension of this class is possible.

The class of Easy2 has almost the same specific methods but it also includes a `FixToFloat` method for converting the stored temperature values into real values.

The specific classes are derived from the general parent tag.

Calling these methods we can obtain any logged data from the tags and displaying and saving them are also possible.

5. Summary

The main aim of this paper was to demonstrate the way of developing software that enables a PDA with built in RFID reader to read any temperature sensor tag compatible with EPC Class 1 Generation 2 standards. For programming a PDA the first step is providing the software environment then using the necessary DLL files exposing the RFID methods that are needed in order to read the available transponders. If these RFID tags are compatible with Gen2 protocol the logged date can be obtained from the USER memory bank. In case of different types of tags the reading of the content of the TID memory bank is required as the model numbers can be used to identify the tag before reading the USER memory bank. Naturally, this is not the only way of identifying the tags. In our case the model of A927Z has 0x2 at the start address of USER memory bank because it stores the number of sensors integrated in the tag as opposed to model Easy2 that always has 0x1 here.

These tags log not only the ambient temperature but the date of the measuring as well. Exploring the values at the available addresses and interpreting the logged data are also essential. Knowledge about this information is needed before the implementation of the necessary classes can be started. Storing the variables required by the API methods, implementing the methods identifying the tags as well as methods that are inherited by the derived classes must also be coded in the ancestor class. The specific attributes of different types of tags as well as the specific method of reading the tags and interpreting the scanned hexadecimal temperature and date must be implemented in the derived tag classes.

After developing this software the RFID readers of different manufacturers and their demo software have become obsolete. Using this software any sensor tag is readable. In case a new type of sensor tag is developed and introduced only the implementation of a new specific derived class and its specific methods (`interpreterOfDate`, `interpreterOfValue`, etc.) are necessary. In case another type of PDA only the rewriting of the `ReadingFromTag` method is necessary the other methods need not be unchanged.

References

[1] Klaus Finkenzeller: RFID HANDBOOK Fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication, third edition, WILEY, ISBN: 978-0-470-69506-7

[2] EPCglobal Inc.: Class-1 Generation-2 UHF RFID Conformance Requirements Specification v. 1.0.2. (2005).

[3] ftp://ftp.alientechnology.com/pub/handheld/alh901x/docs/ALH-901x_Users-Guide_2013-12.pdf, 2013

[4] ftp://ftp.alientechnology.com/pub/handheld/alh901x/ALH-900x-901xDevelopersGuide.pdf, 2013

[5] http://www.caenrfid.it/en/CaenProd.jsp?mypage=3&parent=65&idmod=781, 2013

[6] http://www.caenrfid.it/en/CaenProd.jsp?mypage=3&parent=65&idmod=780, 2013.

[7] Tounsi, Wiem, Nora Cuppens-Boulahia, Joaquin Garcia-Alfaro, Yannick Chevalier, and Frédéric Cuppens. "KEDGEN2: A key establishment and derivation protocol for EPC Gen2 RFID systems." Journal of Network and Computer Applications (2013).

[8] Hernyák Zoltán: Magasszintű programozási nyelvek II. Eszterházy Károly College, 2013.

[9] Juhász István: Programozás 2. http://www.inf.unideb.hu/~panovics/Programozas220040330.pdf, 2013.