UNIVERSITY OF TROMSO

# INF 3910 - UICC

## Assignment 1 – TAG 4 emulation

**By Simon Engstrøm Nistad**

**15.02.2013**

This is the first assignment of the UICC course, and the task is to implement an emulation of the TAG 4 NFC Card to a UICC SIM-card using the Java Card library. The finished result should be able to run on a NFC enabled phone, and successfully emulate a tag 4 card possible of READ and WRITES

# 1.0 Introduction

## 1.1 Overview

In this assignment the task is to emulate a tag type 4 on a UICC SIM-card. It should be possible to read and write data to the card. The implementation is required to implement the Technical specification[1] for Tag 4, and should operate seamless toward other devices.

## 1.2 Requirements

The requirements for the assignment are:

- Implement the TAG 4 emulation in a java card applet
- Install and test this implementation on an actual SIM-card
- When running the test script, the implementation should have the same output as the result file posted on Fronter™
- When running on a phone, the applet should be detected ass a tag 4 card, and it should be possible to read and write to the card

Optional

- Create an Android™ application that communicates with the SIM-card

## 1.3 Technical background

### 1.3.1 APDU

APDU (Application Protocol Data Units), are the messages used to communicate with the applet. There are two kinds of APDU's; C-APDU used for command messages, and R-APDUS used for the response messages.

| CLA | INS | P1 | P2 | LC (optional) | DATA (optional) | LE (optional) |
|---|---|---|---|---|---|---|
| Class byte | Instruction byte | Param. Byte 1 | Param. Byte 2 | Length of data | Data bytes | LE field |

Table 1 - Shows the structure of the C-APDU message

| Response Body | SW 1 | SW 2 |
|---|---|---|
| Data bytes | Status Word 1 | Status Word 2 |

Table 2 - Shows the structure of the R-APDU message

[1] http://apps4android.org/nfc-specifications/NFCForum-TS-Type-4-Tag_2.0.pdf

### 1.3.2 TAG 4

Tag type 4 is a simple RFID format used on some NFC tags. The common sizes are between 4 → 32 KB[2] and the tag supports READ/WRITE and Read-only modes. The Tag has two files deployed on card; A Compability Container (CC file) used to store information about the tag (See details on page 10 in source[1]). Note that this file cannot be updated, and a NDEF file holding the actual data stored on the tag.

## 2.0 Design and Implementation

The design of this implementation is a simple state machine having the three states as shown in Figure 1

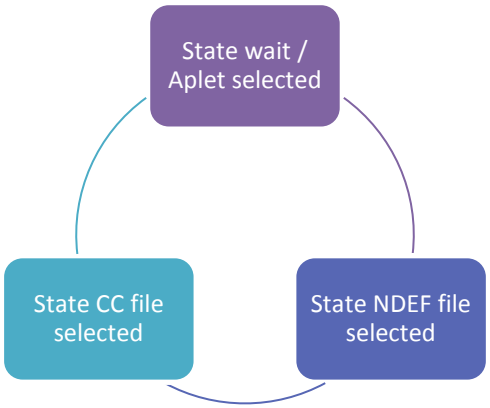


Figure 1 - Shows the simple state machine implemented

There are basically three commands as shown in Table 3 that needs to be supported to implement emulation for the Tag 4 card.

| Command / Response | Description |
|---|---|
| **Select** | Selection of applications, or files |
| **Read Binary** | Read data from file |
| **Update Binary** | Update (erase and write) data to file |

Table 3 - Shows the basic commands needed

The INS (instruction) field in the C-APDU command showed in Table 1 is used to specify the command to be operated. To read or write from the NDEF file where data is stored as mentioned above, the following sequence of commands has to be executed:

1. **SELECT: applet**
2. **SELECT: CC file**
3. **READ BINARY: CC file (to fine specifications about NDEF file)**
4. **SELECET NDEF: file**
5. **READ / UPDATE BINARY: NDEF file**

[2] http://www.oprfid.com/NFC-Tag.html

Because Tag 4 cards have a limited command set, the applet was quite simple to implement, and the functionally only took ~150 lines of code. The first thing that was needed was to setup the basic structure needed for the tag, the CC file and the NDEF file was created using two byte arrays, and the data for the cc file was manually set upon initialization. Next a simple state machine as shown in figure 1 was setup using a switch, and this is where the core functionality of the applet was written. As mentioned above, to do a successful READ / WRITE operation, the steps above has to be executed in that specific order. The state machine will return an exception if an operation is executed in the wrong state.

The select operation is the actual command that changes the state on the card, and the states are set as shown below:

- SELECT Applet → state wait
- SELECT CC file → state CC selected
- SELECT NDEF file → state NDEF selected

Because the select command has the same INS (0xA4) in the C-APDU more information has to be collected from the APDU to correctly verify what the command is intended to select. This is done by checking the P1, P2 and DATA parameters.

| | P1 | P2 | DATA |
|---|---|---|---|
| **Applet** | 0x4h : by name | 0x00 : first only occurrence | Applet ID |
| **CC file** | 0x00 : by ID | 0xCH: first only occurrence | File Identifier : 0xE103 |
| **NDEF File** | 0x00 : by ID | 0xCH: first only occurrence | FILE Identifier from CC file |

Table 4 – Shows the input required to select the different files

For the READ operation the result will depend on what state the card is in; state CC selected while return data from the CC file and state NDEF selected will return data from the NDEF file. If the card is in state wait, the applet will throw an exception to the caller. The data read is packet in an R-APDU and returned to caller.

For a Binary update command, the card has to be in the NDEF selected state, otherwise an exception is raised. Because of the slow speed of the SIM-card, a busy loop loading bytes from the C-APDU buffer has to be used to assure that all bytes are successfully loaded into the applet before it is stored in the NDEF file. This is also required for the select command to be sure that the complete file identifier is received.

For any other C-APDUS, an invalid command exception is raised and returned to caller.

# 3.0 Results

## 3.1 Measurements

The implementation works booth on the script in Netbeans and works on the Samsung I9300 phones from the ODS lab. The application handles booth READ and Writes, and when checking the info about the tag when read by another phone NFC application, the CC file also verifies to be set correct.

### 3.2 Discussion

The assignment worked out to be quite easy after reading the TAG 4 documentation, and everything worked more or less on the first run. The code itself is as mentioned only ~150 lines, and all the logic is placed into one method. This could been refactored to be more in the “clean code” principle by dividing the logic into more methods and classes, but at this point, the author don’t see any good reasons for doing this, and the code is readably and understandable at its current state.

## 4.0 Conclusion

To conclude everything works has I should and all the requirements are met. At this point I did not have the time to implement the extra part with communication towards an android application, but this will probably be done in the future.

At the beginning of this course I must confess that I did not have that much of an interest towards SIM-card programming, but after this assignment I grown fond of it, and are looking forward to the next assignment.