Implementation and Verification of an operating system on a Smart Card

Submitted By

Divyank Sheth

15MCEC25

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2017

Implementation and Verification of an operating system on a Smart Card

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Divyank Sheth

(15MCEC25)

External Guide	Internal Guide
Mr. Rakesh Kumar Swamy	Prof. Kruti Lavingia
Technical Lead	Assistant Professor
NXP Semiconductors	Nirma University

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2017

NIRMA
UNIVERSITY
INSTITUTE OF TECHNOLOGY

Certificate

This is to certify that the major project entitled **”Implementation and Verification of an operating system on a Smart Card”** submitted by **Divyank Sheth (15MCEC25)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project part-I, to the best of my knowledge, haven’t been submitted to any other university or institution for award of any degree or diploma.

Prof. Kruti Lavingia	Dr. Priyanka Sharma
Internal Guide	Program Coordinator
Dr. Sanjay Garg	Dr. Alka Mahajan
Head of CE Department	Director, IT

Certificate

This to certify that **Mr. Divyank Sheth (15MCEC25)**, a student of M.Tech CSE (Computer Science and Engineering), Institute of Technology, Nirma University, Ahmedabad is working at NXP Semiconductors since 13/06/2016 and carried out his thesis work titled **Implementation and Verification of an operating system on a Smart Card**. He is working as intern under guidance of Mr. Rakesh Kumar Swamy (Mentor) and Mr. Yudhir Kataria (Project Manager). He is working on his assigned work and is allowed to submit his dissertation report. The results embodied in this project, to the best of our knowledge, have not been submitted to any other university or institution for award of any degree or diploma. We wish him all the success in future.

Mr. Yudhir Kataria	Mr. Rakesh Kumar Swamy
Senior Project Manager	Technical Lead
NXP Semiconductors	NXP Semiconductors

Statement of Originality

I, **Divyank Sheth**, Roll. No. **15MCEC25**, give undertaking that the Major Project entitled "**Implementation and Verification of an operating system on a Smart Card**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Signature of Student

Date:

Place:

Endorsed by

Prof. Kruti Lavingia

(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Mr. Rakesh Kumar Swamy and Mr. Yudhir Kataria** for his valuable guidance and continual encouragement throughout this work. The appreciation and continual support has been a great motivation to me in reaching a higher goal. Their guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Mr. Mahesh DC, Mr. Lokeswara Bheeman, Miss Sindhu Rajendra Prasad and Mr. Pavan Kumar Kambhampati** for kind support and providing basic infrastructure and environment.

A special thank you is expressed wholeheartedly to **Prof. Kruti Lavingia** Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation she has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Divyank Sheth
15MCEC25

Abstract

Now a day, smart card is used at all government and private sectors at all possible domains. Popularity of smart card has increased because of functionalities provided by it. But at the same time, it increases responsibility of an operating system of smart card. Smart card operating system has to deal with file management aspect and communication standards along with providing support for security mechanism. On top of it multiple application support need to be provided. So, verification of smart card operating system becomes very crucial to make sure that above functionalities are supported with highly secure mechanism for data transfer between smart card and reader.

Abbreviations

AES	Advanced Encryption Standard
APDU	Application Protocol Data Unit
Card	A Proximity Device (PD) in card form factor. so a PICC
CRC	Cyclic Redundancy Check
DES	Data Encryption Standard
DF	Dedicated File
EF	Elementary File
MF	Master File
PCD	Proximity Coupling Device
PICC	Proximity Integrated Circuit Card
Terminal	PCD

Contents

List of Figures

Chapter 1

Introduction

1.1 Smart Card Basics

Smart card is a card which is having computer chip either memory type (store data) or microprocessor type (transact data). This data can be value, information or both. Data can be stored and processed using smart cards chip. Smart card data is read by card reader. Here mechanism can be interpreted in term of master and slave, where card reader act as master and card act as slave.

Cards be divided into two categories: card without chip(magnetic strip card) and card with chip i.e. smart card. Chip is the element that differentiate memory card from processor card. It is divided into two categories:processor card without or with coprocessor.

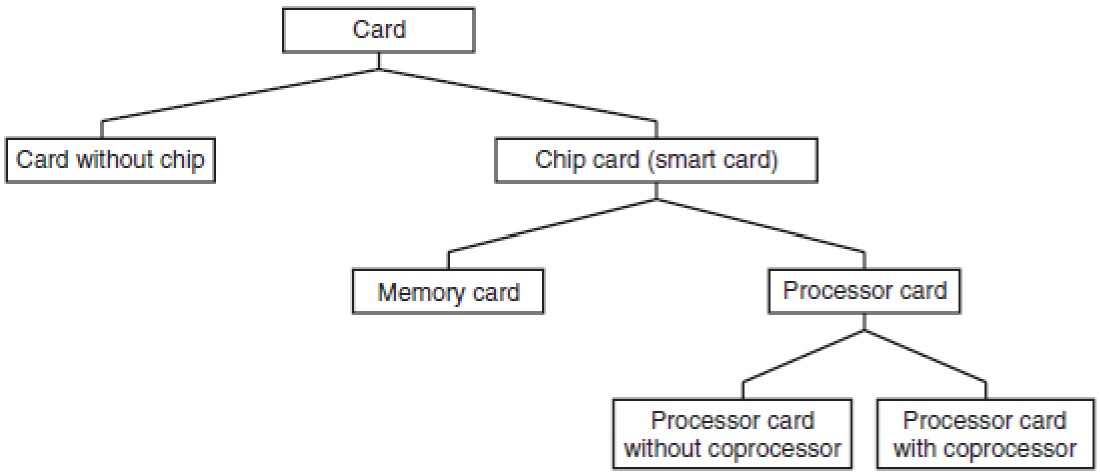


Figure 1.1: Classification of cards

Smart card is also known as PICC (Proximity Integrated Circuit Card) which contain microcontroller. The microcontroller consist of CPU, RAM, ROM and EEPROM memory. Smart card is successful because of security feature provided by it and various other application where it can be deployed. Smart card operating system is stored at ROM which is responsible for maintaining integrity and confidentiality of stored data, provide communication with PCD (Proximity Coupling Device) and access right management for data. [1]

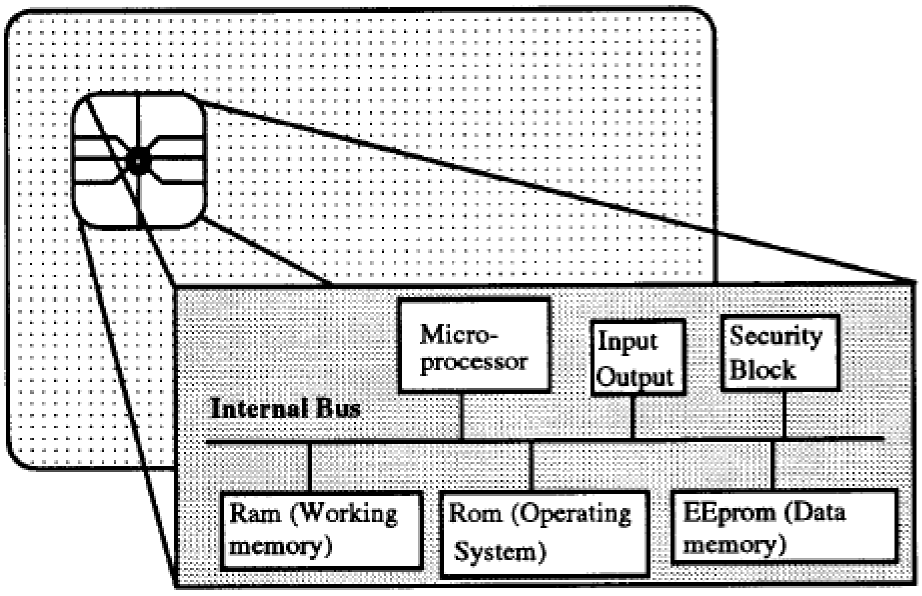


Figure 1.2: PICC architecture

Smart card is having following security feature.

- Conventional address and data busses exist in usual memory devices are not present in chip. If multiple chips are used and interconnected then it will be easy to spy but here single chip is used.
- For better security, encryption algorithm and decryption algorithm is used.
- Non-volatile memory is provided security against unauthorized access.

Smart card provide following cryptographic services:

- Pseudo random number generation

- Key generation routine
- DES encryption algorithm
- RSA and DSS digital signature generation
- Hash function

1.2 Smart Card Application

Smart card is used at both government and private domain for various applications. Broad category where smart card can be used as follows:[2]

- Financial
  Smart card can be used for financial transaction. Now a day contactless smart card is used for financial transaction. So, there is no need for physical contact between PICC and PCD. For contactless payment, contactless card need to be flash to reader within proximity of 8-10 centimetres.

- Healthcare
  Smart card can be used in healthcare domain. Smart card can be used to store patient details, medical records and ease of quickly accessible emergency medical details and use of government funds and benefits.

- Transportation
  Smart card can be used for transportation application like to pay parking fee or to pay transit fee. For these use case, stored value card is used. It has evolved from magnetic stripe card to contactless smart card. Most metropolitan areas of USA use contactless smart card for transportation purpose. Name of regions are Los Angeles, Houston, Boston, San Francisco and San Diego. Parking industry has also adopted smart card for payment purpose.

- Identity
  For security requirement, smart card is most suitable solution for identity. E.g. identification, privacy protection. Many organizations have adopted smart card technology for access control and identity purpose.

1.3 Problem statement

Smart card play very crucial role in domain like financial, healthcare, transportation and identity etc. Objective is to devised operating system that is robust enough to support above use cases. Which is followed by verification of smart card operating system.

Chapter 2

Literature Survey

2.1 Smart card operating system

Role of an operating system is to provide clear set of resources rather than expecting user to manage resources. [3] The functionality of card depends on the operating system rather than on the microcontroller. ROM is the element where operating system of smart card resides.

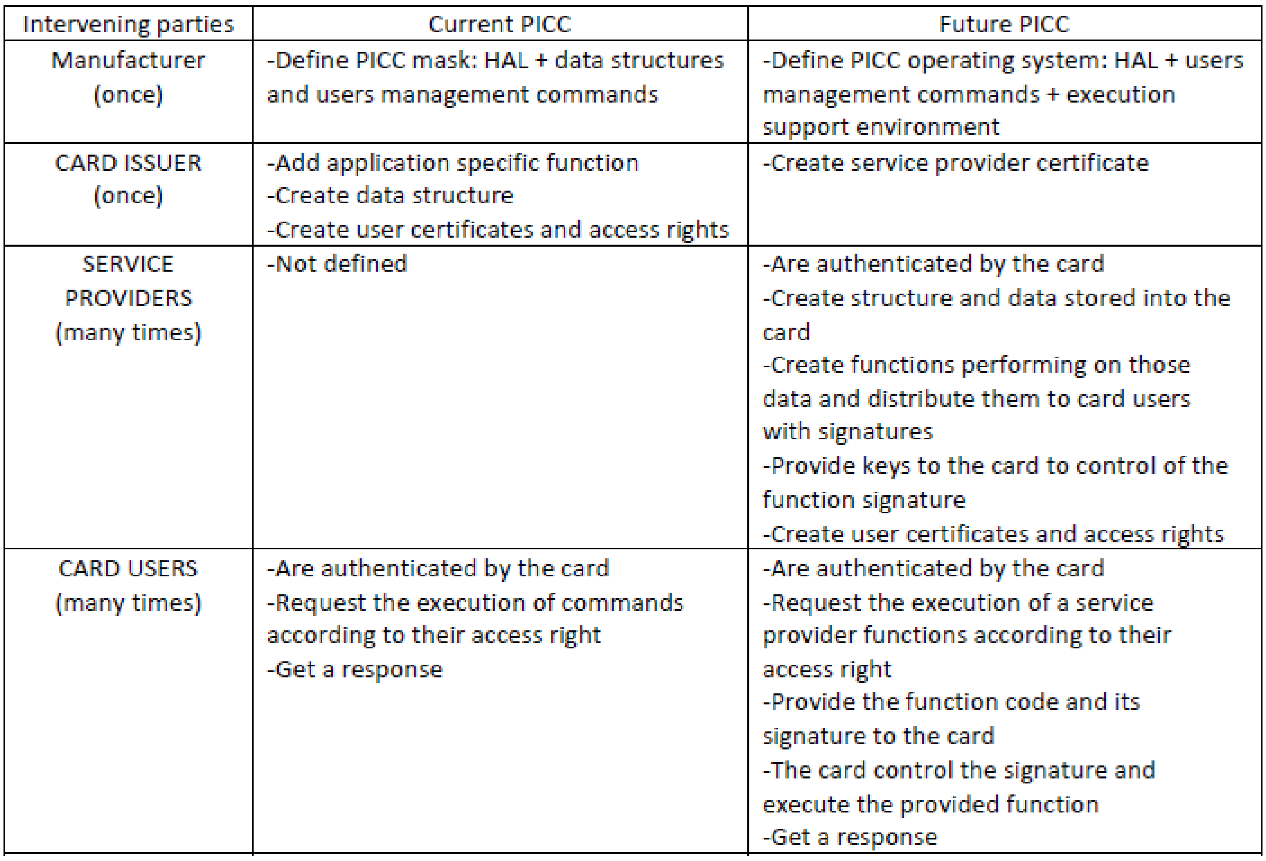
In contrast to known operating system, smart card operating system do not have user interface. This is because it is written to provide different set of functionalities.

Smart card is being used at many private and public sectors for various applications and it is growing. To maintain uniformity among different sectors standards are specified. It introduce challenge for smart card operating system development. As smart card operating system has to support standards for application management, security and communication.

Smart card operating system should support multiple application and multiple vendor environment. Operating system must support keys and other sensitive data against attacks and malicious application must be kept isolated from other application.[4]

Smart card operating system should support communication protocol which in turn enable communication between PICC and PCD regardless of platform dependency. Smart card operating system should be such that it can be ported to other smart card with minimal efforts and cost. Operating system should be design in such a way that it can accommodate maximum number of application with minimum memory.

Operating system used for smart card is also known as mask. As PICC is used extensively, new challenge is introduced for operating system designer i.e. operating system must be able to load multiple application in one PICC. Data may be received from different sources, for different application and required to be handled in different way for each application.



Intervening parties	Current PICC	Future PICC
Manufacturer (once)	-Define PICC mask: HAL + data structures and users management commands	-Define PICC operating system: HAL + users management commands + execution support environment
CARD ISSUER (once)	-Add application specific function -Create data structure -Create user certificates and access rights	-Create service provider certificate
SERVICE PROVIDERS (many times)	-Not defined	-Are authenticated by the card -Create structure and data stored into the card -Create functions performing on those data and distribute them to card users with signatures -Provide keys to the card to control of the function signature -Create user certificates and access rights
CARD USERS (many times)	-Are authenticated by the card -Request the execution of commands according to their access right -Get a response	-Are authenticated by the card -Request the execution of a service provider functions according to their access right -Provide the function code and its signature to the card -The card control the signature and execute the provided function -Get a response

Smart card operating system should support multiple application. A scenario is like malicious application gain access over other application or sensitive data. So such scenario must be avoided. Mechanism to achieve this is no application must be able to add or remove other application without authentication.

- Card owner must be able to add application that are approved by card issuer.

- One application residing on smart card must not be able to read or remove another application.
- One application residing on smart card must not be able to add new application.
- One application residing on smart card must not be able to alter access right of existing application.
- Two application residing on card communicate only after authentication and only with appropriate access right.
- Above condition is also applicable for communication between application existing on smart card and application outside smart card.
- Confidentiality and integrity need to be taken care when application communicate with PCD.

2.2 Types of smart card operating system

Smart card operating system can be divided into following categories:[5]

- Global Native
- Global non-Native
- Global Mixed

2.2.1 Global Native Smart Card operating system

This kind of smart card operating system support specification for global platform. It also support application development which is carried out using C or Assembly i.e. use of native programming language.

Best example for global native smart card operating system is STARCOS which support high performance application. At the same time, it requires deep knowledge and expertise on machine level programming.

High level programming language smart card operating system support framework that can protect sensitive details. To get same behaviour for native application requires more expertise and more development time.

2.2.2 Global Non-Native smart card operating system

For quick development and to address security requirements of smart card, high level programming languages are used. Examples of global non-native smart card operating systems are Java based OS and MULTOS.

Java based smart card operating system

Java based smart card operating system assist verification as well as execution of Java bytecode. Java card support inter process communication.

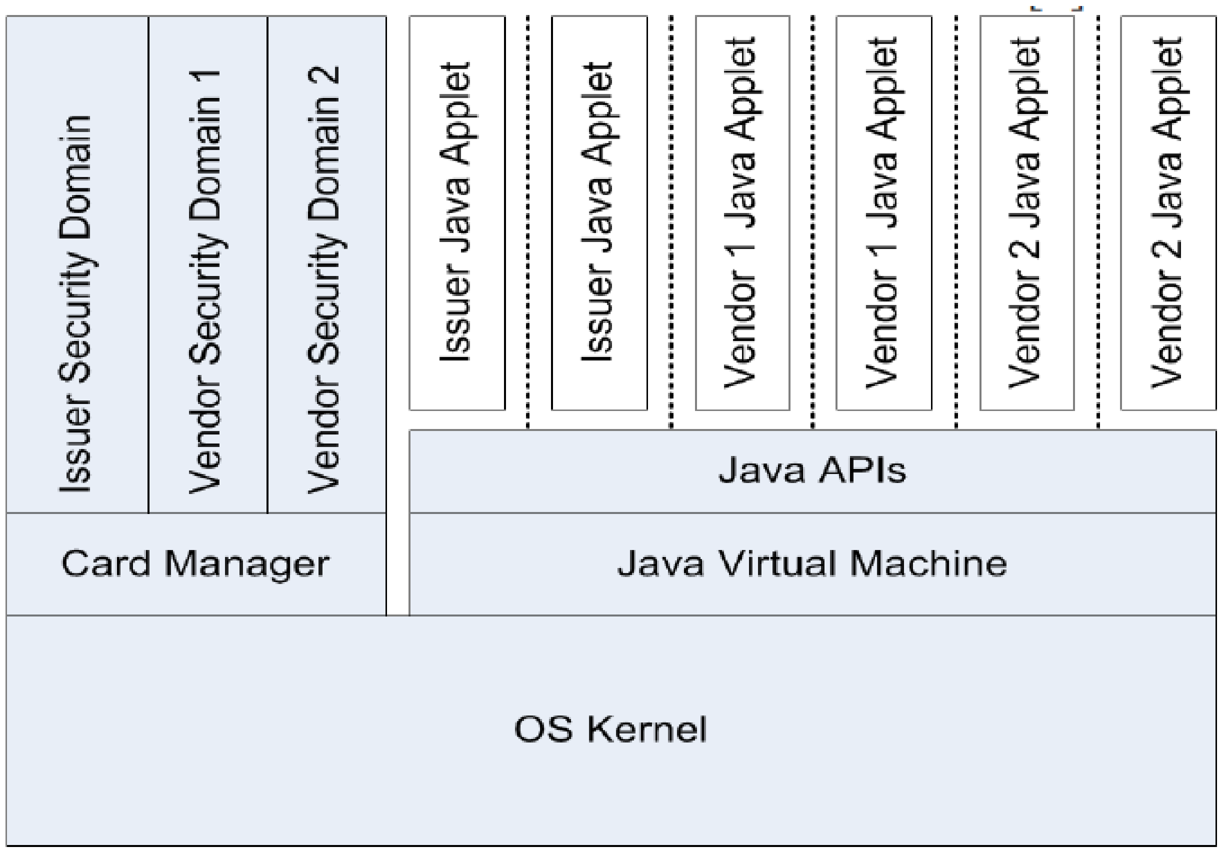


Figure 2.1: Java Card architecture

MULTOS smart card operating system

Using MULTOS one can develop smart card operating system using programming language like Java, C and MEL (MULTOS Executable Language). Separate security aspect is not supported by MULTOS. It need to be handled at application level.

2.2.3 Global mixed smart card operating system

Native and non-native both have their pros and cons so customers requirement always cannot be fulfilled by any one. So one new category has evolved which is called global mixed smart card operating system. Example is Caernarvon operating system.

Smart card operating system need to perform following main tasks:

- Transfer of data to and from smart card
- Management of files
- Execution of program code
- Execution of cryptographic algorithms

2.3 File management

File management is considered as a main responsibility of a smart card operating system. File management covers creation and deletion of files and provision of read and write access to particular file. In addition to that file management has to deal with assigning access rights and observing conflict with access rights if any. Smart card applications depend on file structure so file management is very crucial.[6]

2.3.1 File types

Smart card operating system support two categories of files. Dedicated File (DF) and Elementary file (EF).

Smart card file structure is similar to tree structure with root directory. The root directory is known as MF (Master File). For smart card operating system file structure onlu one MF exist. MF cannot store data but it can only contain other directories. MF is identified using FID 3FOO, that file identifier is reserved.

Smart card directories are called DFs (Dedicated Files). Theoretically one can nest it infinitely. Actually, in application five levels exist. Smart card operating system support up to eight level. EFs (Elementary file) contains actual application data and operating system data. EFs are of two types. Internal EF that is used for storage of data required by card (data is used by smart card for control and management requirement). Working EF that is used for storage of data not required by smart card (data that is used by reader for any given transaction).

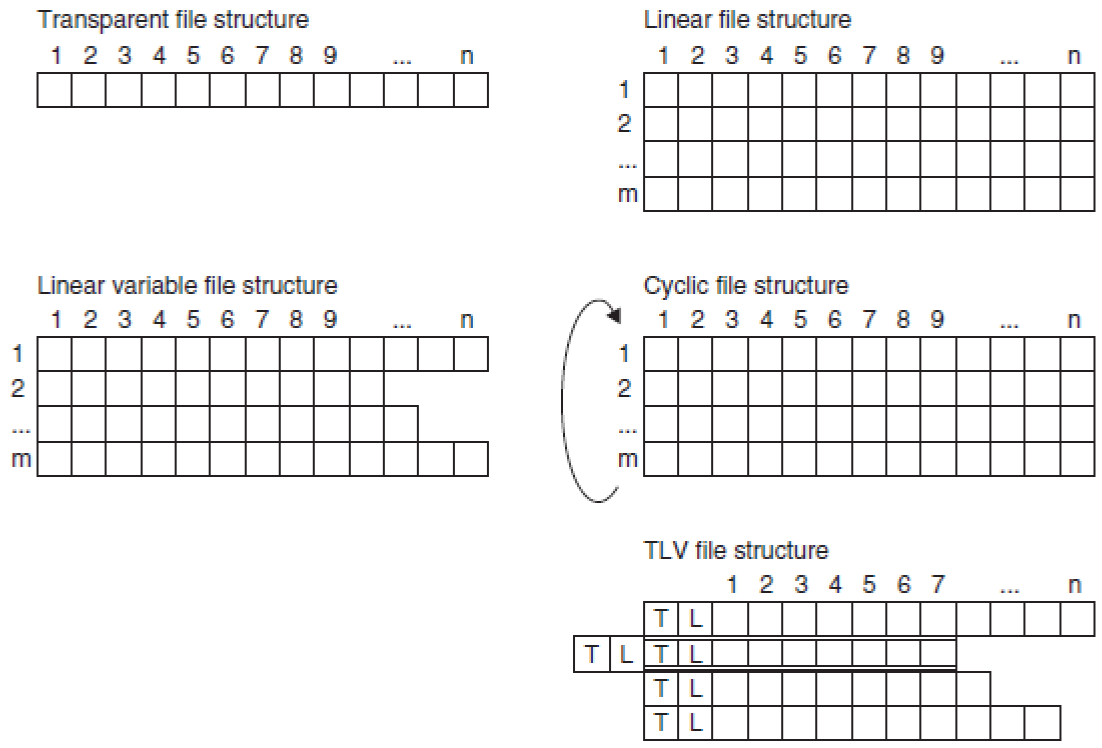


Figure 2.2: Structure of data files (EFs)

2.3.2 File structures

For EF files, data can be stored in various ways. Five different ways are available to store data as shown in figure 2.2.[7]

In transparent file structure, the data items are stored as series of bytes. The EF is seen as sequence of data units. The command READ BINARY and UPDATE BINARY can be used to read data from or write data to this file structure. Here number of bytes and an offset from the start of file is required to specify.
In addition to transparent file structure, there are record oriented file structures. Equal length record can be stored at EFs with linear fixed file structure. For records with variable file structure one can use EFs with linear variable file structure. If records with variable length need to be stored then EFs with linear variable file structure will be more optimize in terms of memory space than EFs with linear fixed file structure.
In case when data stored in past is not required then one can overwrite it. This purpose is achieved using cyclic file structure. Cyclic file structure maintain pointer which indicate which record is most recently written.
To deal with records stored at record oriented file can be read and written using READ RECORD and UPDATE RECORD command.
In TLV data structure, each data object is identified by tag (T) and length (L). It is followed by value (V) i.e. actual data. The commands GET DATA and PUT DATA are used to read and store data object.
The card must support at least one of the following methods for EFs:

- Transparent EF
- Linear EF with fixed size record
- Linear EF with variable size record
- Cyclic EF with fixed size record

2.4 NUnit

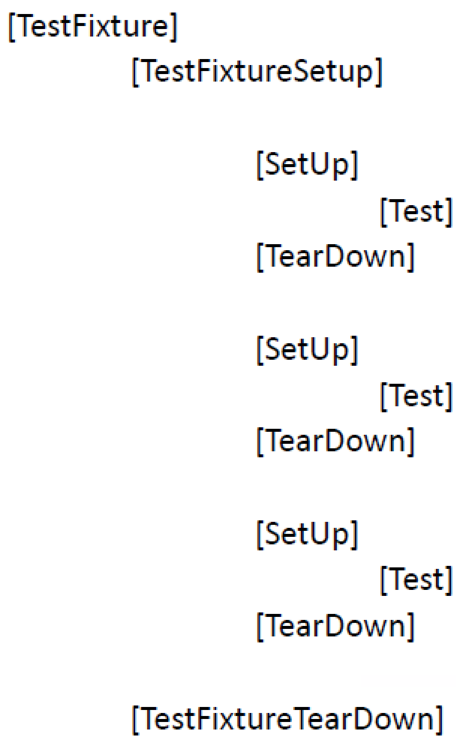
NUnit is unit testing framework for Microsoft .NET. It is open source software. It makes tester job easy. While executing test case it is required to initialize value to variables and after completion of test it is required to reset value of variables. For large number of tests it is tedious to do this job manually.

Another scenario is in certain iteration, it is required that test of particular category is only to be run. For large project or for multiple test, it is difficult to find test of required category. NUnit take out this burden form tester and handle it very nicely. NUnit support set of attributes via which it can handle these situations.

Primarily two ways to run tests.

- Console runner
- GUI runner

NUnit attribute hierarchy

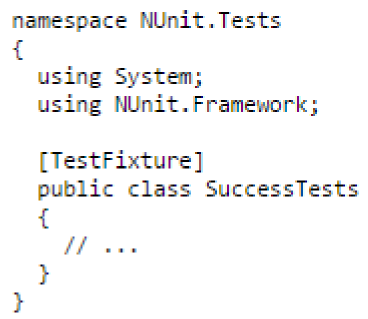


```
[TestFixture]
        [TestFixtureSetup]

                [SetUp]
                        [Test]
                [TearDown]

                [SetUp]
                        [Test]
                [TearDown]

                [SetUp]
                        [Test]
                [TearDown]

        [TestFixtureTearDown]
```

Figure 2.3: Attributes hierarchy

2.4.1 TestFixture

This attribute is used to indicate class that contain tests and optionally setup and teardown.

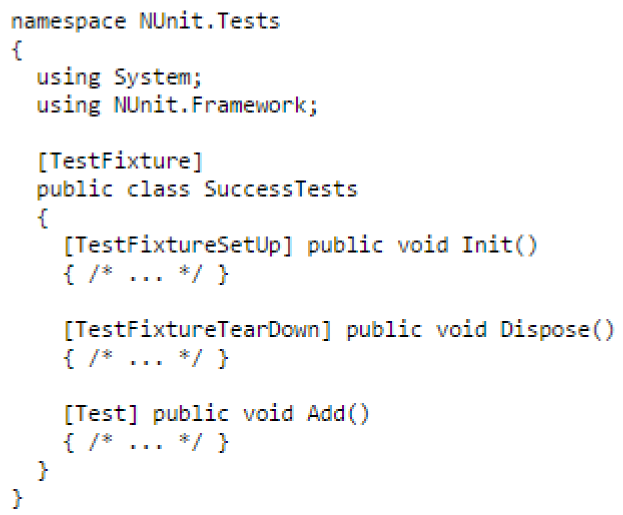


```
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    // ...
  }
}
```

Figure 2.4: TestFixture attribute

2.4.2 TestFixtureSetUp

TestFixtureSetUp is used inside TestFixture. It is used to list set of functions that are performed prior to execution of any test.



```
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    [TestFixtureSetUp] public void Init()
    { /* ... */ }

    [TestFixtureTearDown] public void Dispose()
    { /* ... */ }

    [Test] public void Add()
    { /* ... */ }
  }
}
```

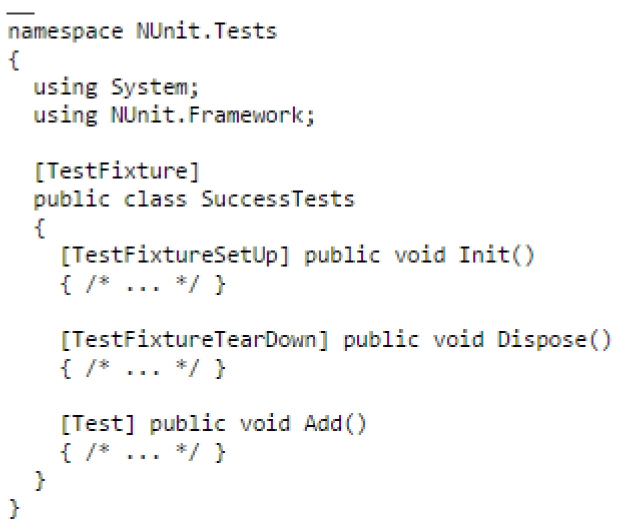
Figure 2.5: TestFixtureSetUp attribute

Here Init() is TestFixtureSetUp function. It will get executed before execution of any test.

2.4.3 TestFixtureTearDown

It is used inside TestFixture. It is used to list set of functions that are performed after execution of all test.

Here Cleanup() is TestFixtureTearDown function. It will get executed after all tests



```
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    [TestFixtureSetUp] public void Init()
    { /* ... */ }

    [TestFixtureTearDown] public void Dispose()
    { /* ... */ }

    [Test] public void Add()
    { /* ... */ }
  }
}
```

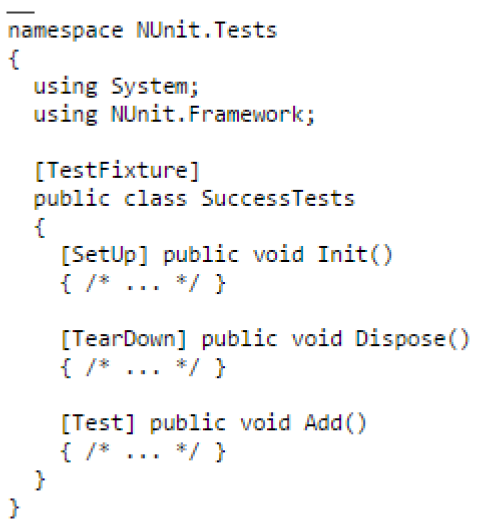
Figure 2.6: TestFixtureTearDown attribute

are completed.

2.4.4 SetUp

It list functions that are performed just before each test method.

namespace Examples.First



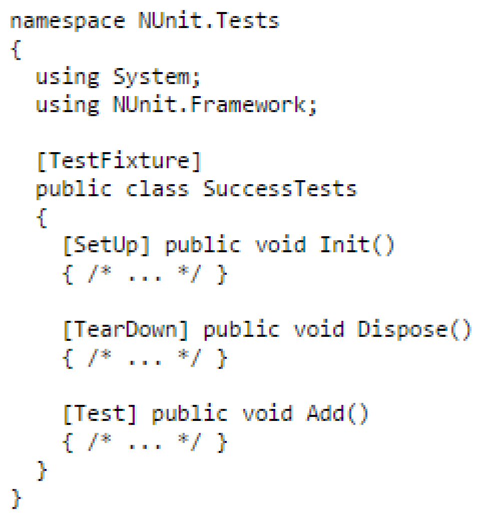
```
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    [SetUp] public void Init()
    { /* ... */ }

    [TearDown] public void Dispose()
    { /* ... */ }

    [Test] public void Add()
    { /* ... */ }
  }
}
```

Figure 2.7: SetUp attribute

Here Init() is SetUp function. It will get executed before execution of Add() test.

2.4.5 TearDown

It list functions that are performed just after each test method



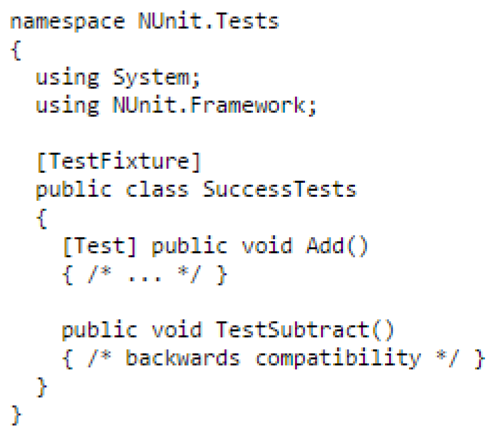
```
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    [SetUp] public void Init()
    { /* ... */ }

    [TearDown] public void Dispose()
    { /* ... */ }

    [Test] public void Add()
    { /* ... */ }
  }
}
```

Figure 2.8: TearDown attribute

Here Cleanup() is TearDown function. It will get executed after execution of Add() test.

2.4.6 Test

It is used to mark method as a test e.g. public static void methodname()



```
namespace NUnit.Tests
{
  using System;
  using NUnit.Framework;

  [TestFixture]
  public class SuccessTests
  {
    [Test] public void Add()
    { /* ... */ }

    public void TestSubtract()
    { /* backwards compatibility */ }
  }
}
```

Figure 2.9: Test attribute

2.5 Testing

Testing play very crucial role in quality of software. depending on approach adopted for testing there are mainly two types of testing:

- White-box testing
- Black-box testing

2.5.1 White-box testing

White box testing is also known as glass box testing, clear box testing, structural testing and transparent box testing. White-box testing tests working of a program. White-box testing of software focuses on procedural detail. White-box testing can be applied at following level.

- Unit level
- Integration level
- System level

At unit level, white-box testing checks path within a unit. At integration level, it tests paths between units. At system level, it tests path between subsystems. Code coverage is used to indicate amount of code that has been covered during testing. Code coverage can be represented in following forms:

- Functional coverage: which focuses on coverage of function
- Statement coverage: which focuses on numbers of line covered
- Decision coverage (condition coverage): which focuses on coverage of conditions

2.5.2 Black-box testing

In black-box testing focuses on fundamental aspect of software. Black-box testing does not focuses on internal logic of software. This methodology consider software as black box, and examining functionalities without internal implementation details.

2.5.3 Characteristics of test cases

- A designed test case must have high probability to find an error.
- Each designed test case must be unique. A test case must not be redundant.
- A designed test case must not be highly complex or simple.

To increase quality of software testing play very important role. In search based software testing one can use metaheuristic optimization algorithm for creating test cases. Many organizations have adopted Search Based Software Testing(SBST). SBST can be used in another fashion called ISBST (Interactive Search Based Software Testing) tool. It will use the knowledge and experience of tester to improve search process. [8]

Chapter 3

File selection command

3.1 Introduction

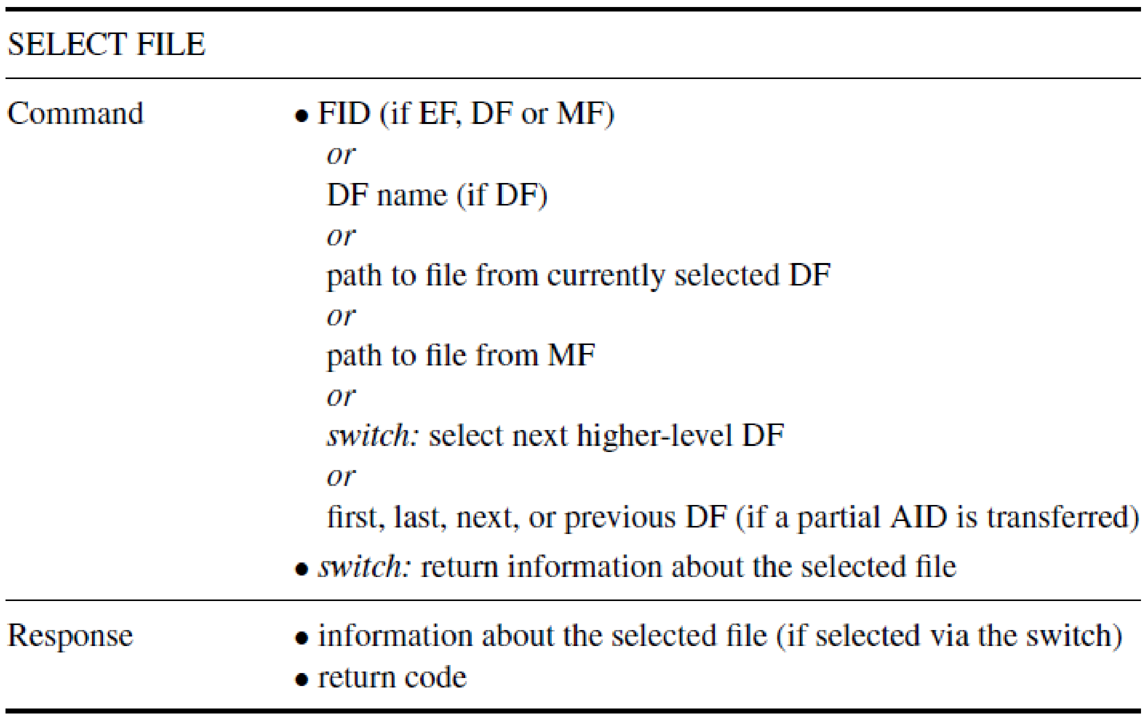
A successful execution of file selection command select specified file within a logical channel. After answer to reset, the MF is implicitly selected. Other files are subsequently selected via execution of file selection command. To select file 2 byte file identifier (FID) is used. To select directory file (DF), 1 byte to 16 byte DF name is used.[9]

File selection command support file selection using path of the file. This path can be relative where file is selected starting from currently selected DF or absolute, where file is selected starting from MF.

Previously selected file is deselected only after successful execution of file selection command. If command cannot be successfully executed then previous selection remain as it is. This makes sure that file is always selected.

Two ways are supported to select file, explicit file selection which uses FID, DF name or path specification. In case of implicit file selection it simplifies command execution and increasing processing speed as it is not required to send File selection command explicitly to select a file provided that file must be an EF and it must be present within current select DF.

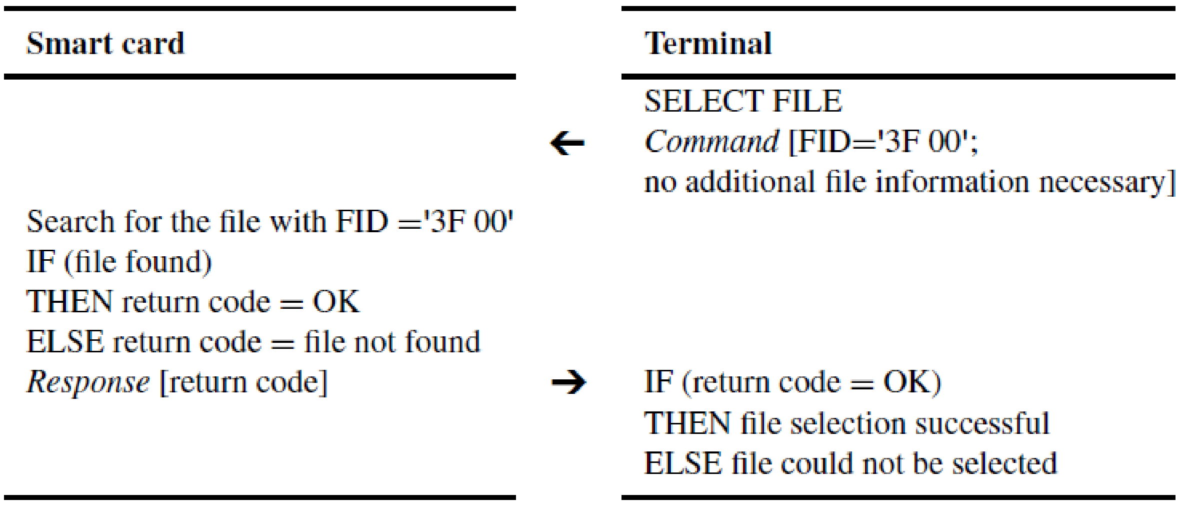
Different test cases are written and run. Result is shown in 3.4.



SELECT FILE	
Command	• FID (if EF, DF or MF) *or* DF name (if DF) *or* path to file from currently selected DF *or* path to file from MF *or* *switch:* select next higher-level DF *or* first, last, next, or previous DF (if a partial AID is transferred) • *switch:* return information about the selected file
Response	• information about the selected file (if selected via the switch) • return code

Figure 3.1: Functionality of file selection command

In figure 3.1 functionalities of file selection command is shown.



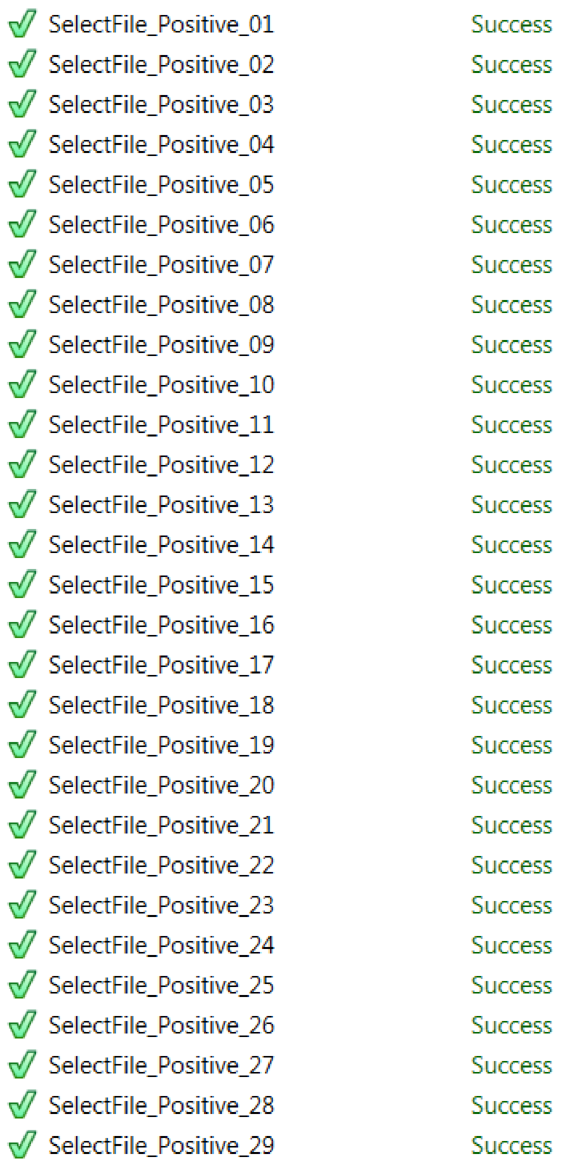
Smart card		**Terminal**

	←	SELECT FILE
		Command [FID='3F 00';
		no additional file information necessary]

Search for the file with FID ='3F 00'
IF (file found)
THEN return code = OK
ELSE return code = file not found
Response [return code]
	→	IF (return code = OK)
		THEN file selection successful
		ELSE file could not be selected

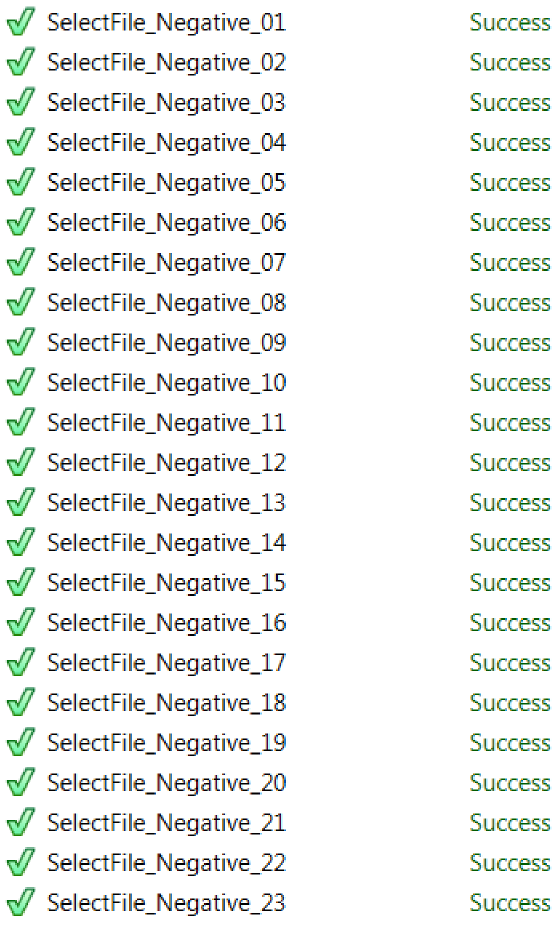
Figure 3.2: command sequence for Select File

In figure 3.2, command exchange between smart card and card reader is shown.



✓	SelectFile_Positive_01	Success
✓	SelectFile_Positive_02	Success
✓	SelectFile_Positive_03	Success
✓	SelectFile_Positive_04	Success
✓	SelectFile_Positive_05	Success
✓	SelectFile_Positive_06	Success
✓	SelectFile_Positive_07	Success
✓	SelectFile_Positive_08	Success
✓	SelectFile_Positive_09	Success
✓	SelectFile_Positive_10	Success
✓	SelectFile_Positive_11	Success
✓	SelectFile_Positive_12	Success
✓	SelectFile_Positive_13	Success
✓	SelectFile_Positive_14	Success
✓	SelectFile_Positive_15	Success
✓	SelectFile_Positive_16	Success
✓	SelectFile_Positive_17	Success
✓	SelectFile_Positive_18	Success
✓	SelectFile_Positive_19	Success
✓	SelectFile_Positive_20	Success
✓	SelectFile_Positive_21	Success
✓	SelectFile_Positive_22	Success
✓	SelectFile_Positive_23	Success
✓	SelectFile_Positive_24	Success
✓	SelectFile_Positive_25	Success
✓	SelectFile_Positive_26	Success
✓	SelectFile_Positive_27	Success
✓	SelectFile_Positive_28	Success
✓	SelectFile_Positive_29	Success

Figure 3.3: Result of execution of test cases for file selection command



```
✓ SelectFile_Negative_01    Success
✓ SelectFile_Negative_02    Success
✓ SelectFile_Negative_03    Success
✓ SelectFile_Negative_04    Success
✓ SelectFile_Negative_05    Success
✓ SelectFile_Negative_06    Success
✓ SelectFile_Negative_07    Success
✓ SelectFile_Negative_08    Success
✓ SelectFile_Negative_09    Success
✓ SelectFile_Negative_10    Success
✓ SelectFile_Negative_11    Success
✓ SelectFile_Negative_12    Success
✓ SelectFile_Negative_13    Success
✓ SelectFile_Negative_14    Success
✓ SelectFile_Negative_15    Success
✓ SelectFile_Negative_16    Success
✓ SelectFile_Negative_17    Success
✓ SelectFile_Negative_18    Success
✓ SelectFile_Negative_19    Success
✓ SelectFile_Negative_20    Success
✓ SelectFile_Negative_21    Success
✓ SelectFile_Negative_22    Success
✓ SelectFile_Negative_23    Success
```

Figure 3.4: Result of execution of test cases for file selection command

Chapter 4

Read and update commands

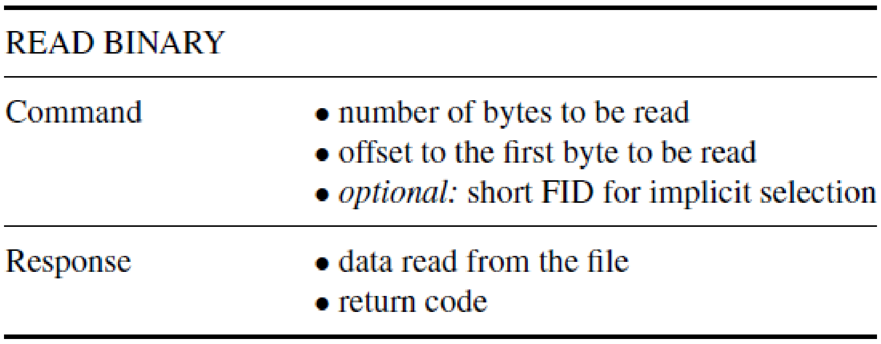
4.1 Introduction

Read and update command are used to write data to EF and read data from EF. Access to these EF is limited by access condition, as unauthorized person will not be allowed to read and/or update.

Depending upon various types of data structure exist for EFs, various commands are exist for read and/or write. E.g. ReadBinary, UpdateBinary, ReadRecord, UpdateRecord.

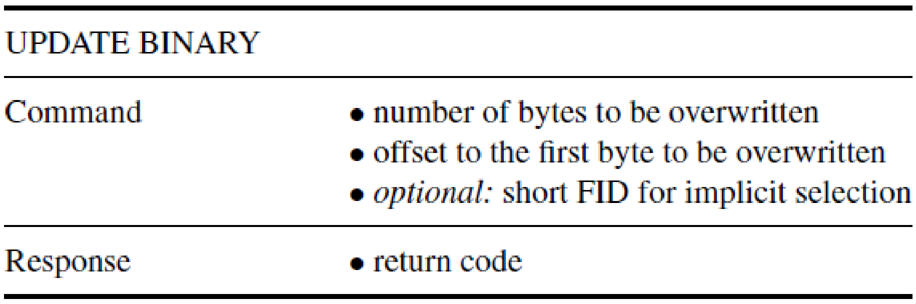
ReadBinary is used to read file such as EF with transparent logical structure i.e. which does not have any internal structure. UpdateBinary is used for writing.

UpdateBinary is equivalent to EraseBinary and WiteBinary. As command name suggest ReadBinary is read command and Update Binary is write command, it is required to pass an offset to first byte to be addressed and length parameter.



READ BINARY	
Command	• number of bytes to be read • offset to the first byte to be read • *optional:* short FID for implicit selection
Response	• data read from the file • return code

Figure 4.1: Functionality of ReadBinary



UPDATE BINARY	
Command	• number of bytes to be overwritten • offset to the first byte to be overwritten • *optional:* short FID for implicit selection
Response	• return code

Figure 4.2: Functionality of UpdateBinary

Smart card		**Terminal**
	←	READ BINARY *Command* [offset = 2 bytes, number of bytes to be read = 5]

requested data := '03' || 'FF' || '00'
|| 'FF' || '00'
Response[requested data || return code]
	→	IF (return code = OK)
		THEN READ BINARY successful
		ELSE abort

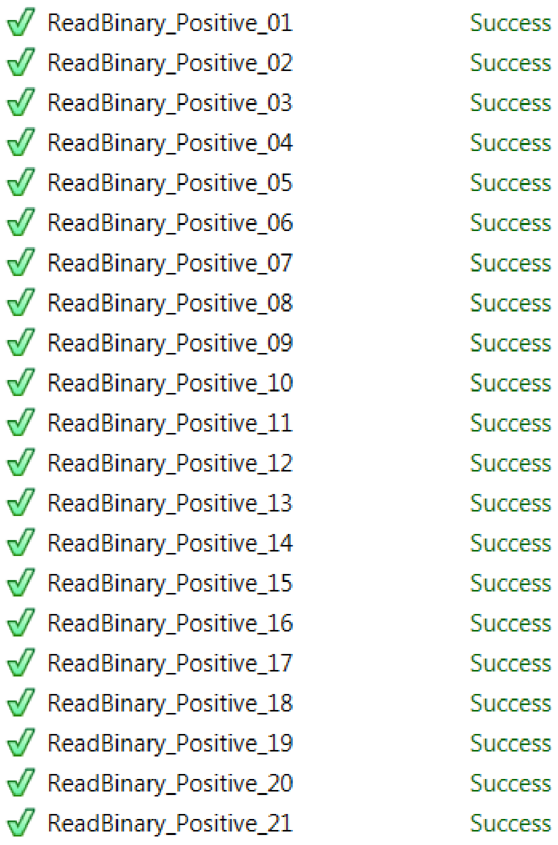
	←	WRITE BINARY *Command* [offset = 3 bytes, number of bytes to be written = 2, data = 'F0 F0']

Response [return code]	→	IF (return code = OK)
		THEN WRITE BINARY successful
		ELSE abort

	←	UPDATE BINARY *Command* [offset = 5 bytes, number of bytes to be written = 2, data = 'F0 F0']

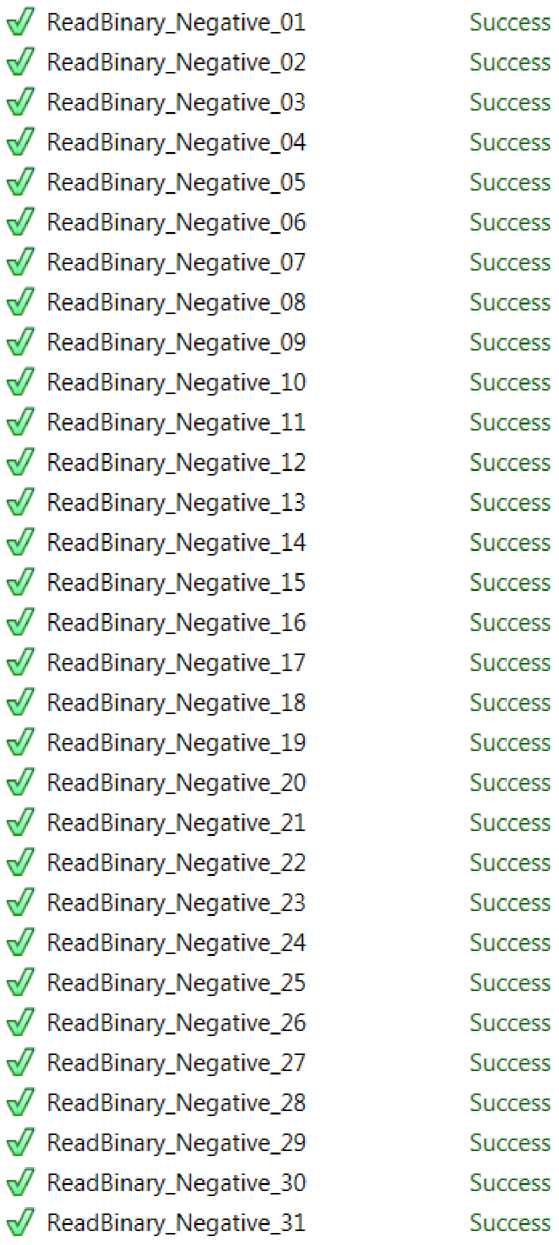
Response [return code]	→	IF (return code = OK)
		THEN UPDATE BINARY successful
		ELSE abort

Figure 4.3: Command exchange for ReadBinary, WriteBinary and UpdateBinary



```
✓ ReadBinary_Positive_01    Success
✓ ReadBinary_Positive_02    Success
✓ ReadBinary_Positive_03    Success
✓ ReadBinary_Positive_04    Success
✓ ReadBinary_Positive_05    Success
✓ ReadBinary_Positive_06    Success
✓ ReadBinary_Positive_07    Success
✓ ReadBinary_Positive_08    Success
✓ ReadBinary_Positive_09    Success
✓ ReadBinary_Positive_10    Success
✓ ReadBinary_Positive_11    Success
✓ ReadBinary_Positive_12    Success
✓ ReadBinary_Positive_13    Success
✓ ReadBinary_Positive_14    Success
✓ ReadBinary_Positive_15    Success
✓ ReadBinary_Positive_16    Success
✓ ReadBinary_Positive_17    Success
✓ ReadBinary_Positive_18    Success
✓ ReadBinary_Positive_19    Success
✓ ReadBinary_Positive_20    Success
✓ ReadBinary_Positive_21    Success
```

Figure 4.4: Result of test cases for ReadBinary



✓	ReadBinary_Negative_01	Success
✓	ReadBinary_Negative_02	Success
✓	ReadBinary_Negative_03	Success
✓	ReadBinary_Negative_04	Success
✓	ReadBinary_Negative_05	Success
✓	ReadBinary_Negative_06	Success
✓	ReadBinary_Negative_07	Success
✓	ReadBinary_Negative_08	Success
✓	ReadBinary_Negative_09	Success
✓	ReadBinary_Negative_10	Success
✓	ReadBinary_Negative_11	Success
✓	ReadBinary_Negative_12	Success
✓	ReadBinary_Negative_13	Success
✓	ReadBinary_Negative_14	Success
✓	ReadBinary_Negative_15	Success
✓	ReadBinary_Negative_16	Success
✓	ReadBinary_Negative_17	Success
✓	ReadBinary_Negative_18	Success
✓	ReadBinary_Negative_19	Success
✓	ReadBinary_Negative_20	Success
✓	ReadBinary_Negative_21	Success
✓	ReadBinary_Negative_22	Success
✓	ReadBinary_Negative_23	Success
✓	ReadBinary_Negative_24	Success
✓	ReadBinary_Negative_25	Success
✓	ReadBinary_Negative_26	Success
✓	ReadBinary_Negative_27	Success
✓	ReadBinary_Negative_28	Success
✓	ReadBinary_Negative_29	Success
✓	ReadBinary_Negative_30	Success
✓	ReadBinary_Negative_31	Success

Figure 4.5: Result of test cases for ReadBinary

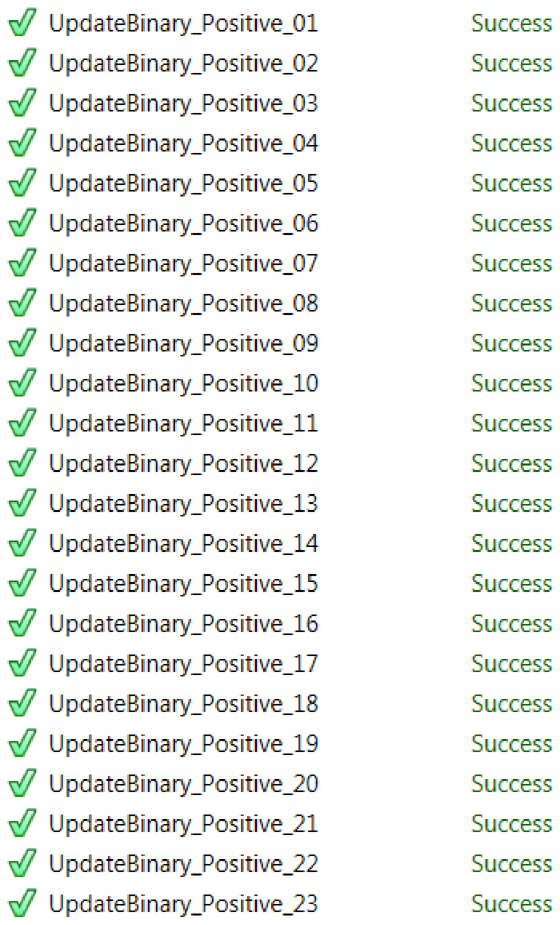
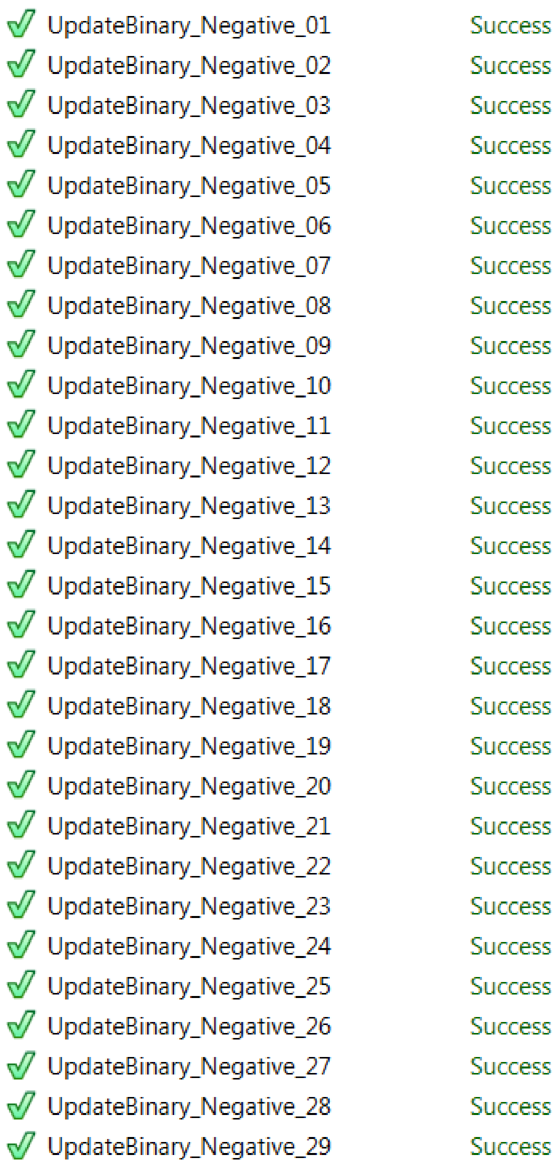

```
✓ UpdateBinary_Positive_01        Success
✓ UpdateBinary_Positive_02        Success
✓ UpdateBinary_Positive_03        Success
✓ UpdateBinary_Positive_04        Success
✓ UpdateBinary_Positive_05        Success
✓ UpdateBinary_Positive_06        Success
✓ UpdateBinary_Positive_07        Success
✓ UpdateBinary_Positive_08        Success
✓ UpdateBinary_Positive_09        Success
✓ UpdateBinary_Positive_10        Success
✓ UpdateBinary_Positive_11        Success
✓ UpdateBinary_Positive_12        Success
✓ UpdateBinary_Positive_13        Success
✓ UpdateBinary_Positive_14        Success
✓ UpdateBinary_Positive_15        Success
✓ UpdateBinary_Positive_16        Success
✓ UpdateBinary_Positive_17        Success
✓ UpdateBinary_Positive_18        Success
✓ UpdateBinary_Positive_19        Success
✓ UpdateBinary_Positive_20        Success
✓ UpdateBinary_Positive_21        Success
✓ UpdateBinary_Positive_22        Success
✓ UpdateBinary_Positive_23        Success
```

Figure 4.6: Result of test cases for UpdateBinary



```
✓ UpdateBinary_Negative_01          Success
✓ UpdateBinary_Negative_02          Success
✓ UpdateBinary_Negative_03          Success
✓ UpdateBinary_Negative_04          Success
✓ UpdateBinary_Negative_05          Success
✓ UpdateBinary_Negative_06          Success
✓ UpdateBinary_Negative_07          Success
✓ UpdateBinary_Negative_08          Success
✓ UpdateBinary_Negative_09          Success
✓ UpdateBinary_Negative_10          Success
✓ UpdateBinary_Negative_11          Success
✓ UpdateBinary_Negative_12          Success
✓ UpdateBinary_Negative_13          Success
✓ UpdateBinary_Negative_14          Success
✓ UpdateBinary_Negative_15          Success
✓ UpdateBinary_Negative_16          Success
✓ UpdateBinary_Negative_17          Success
✓ UpdateBinary_Negative_18          Success
✓ UpdateBinary_Negative_19          Success
✓ UpdateBinary_Negative_20          Success
✓ UpdateBinary_Negative_21          Success
✓ UpdateBinary_Negative_22          Success
✓ UpdateBinary_Negative_23          Success
✓ UpdateBinary_Negative_24          Success
✓ UpdateBinary_Negative_25          Success
✓ UpdateBinary_Negative_26          Success
✓ UpdateBinary_Negative_27          Success
✓ UpdateBinary_Negative_28          Success
✓ UpdateBinary_Negative_29          Success
```

Figure 4.7: Result of test cases for UpdateBinary

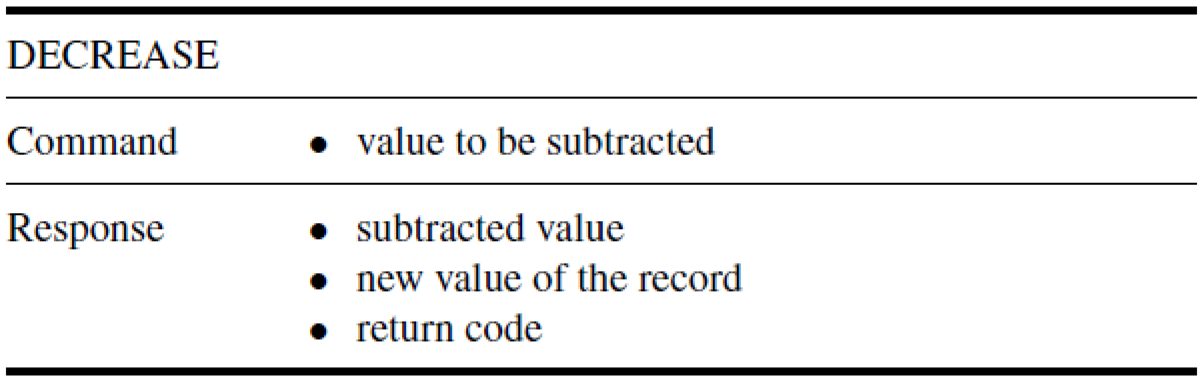
Chapter 5

File Operation Commands

5.1 Introduction

Data is stored at Elementary File. Data can be modified in two ways: a) Using Update data command, b) Using Increase/Decrease command. If application is required to have implementation of counter then use of Increase and Decrease is essential.

Decrease command will subtract mentioned value and Increase command will increase mentioned value. Functionalities of Decrease and Increase command is shown in below figure.



DECREASE	
Command	• value to be subtracted
Response	• subtracted value • new value of the record • return code

Figure 5.1: Functionality of Decrease command

Result for execution of test cases are shown at Figure 5.4 and 5.5 respectively.

INCREASE	
Command	• value to be added
Response	• added value • new value of the record • return code

Figure 5.2: Functionality of Increase command

Terminal (IFD)		Smart card (ICC)
DECREASE *Command* [value to be subtracted = 3]	$\longrightarrow$	*command processing*
	$\longleftarrow$	*Response* [subtracted value = 3 ‖ new value = 7 ‖ return code]

IF (return code = OK)
THEN DECREASE successful
ELSE DECREASE could not be executed		

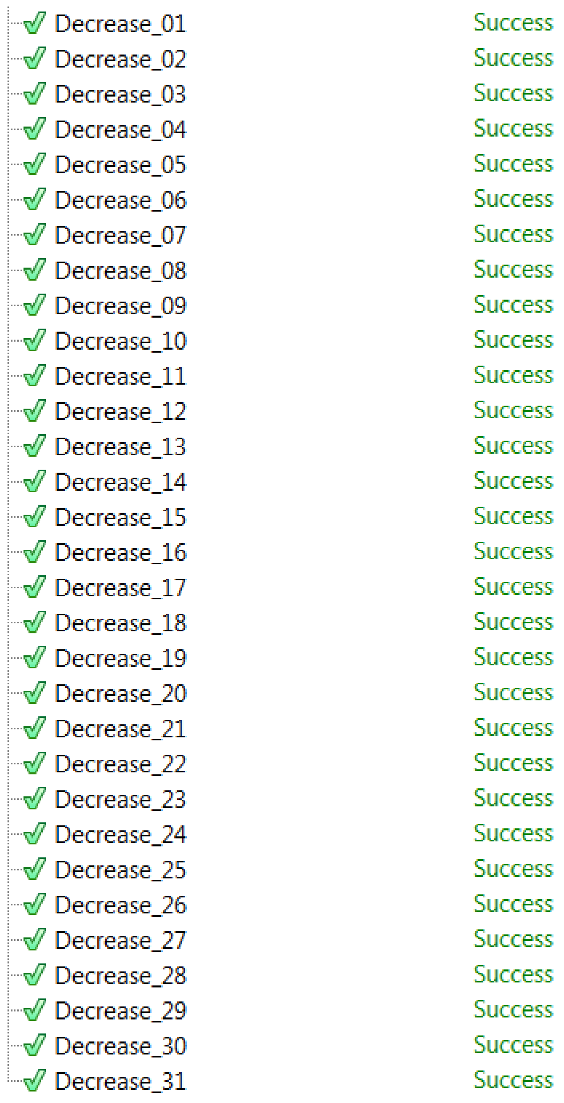
DECREASE *Command* [value to be subtracted = 2]	$\longrightarrow$	*command processing*
	$\longleftarrow$	*Response* [subtracted value = 2 ‖ new value = 5 ‖ return code]

IF (return code = OK)
THEN DECREASE successful
ELSE DECREASE could not be executed		

INCREASE *Command* [value to be added = 5]	$\longrightarrow$	*command processing*
	$\longleftarrow$	*Response* [added value = 5 ‖ new value = 10 ‖ return code]

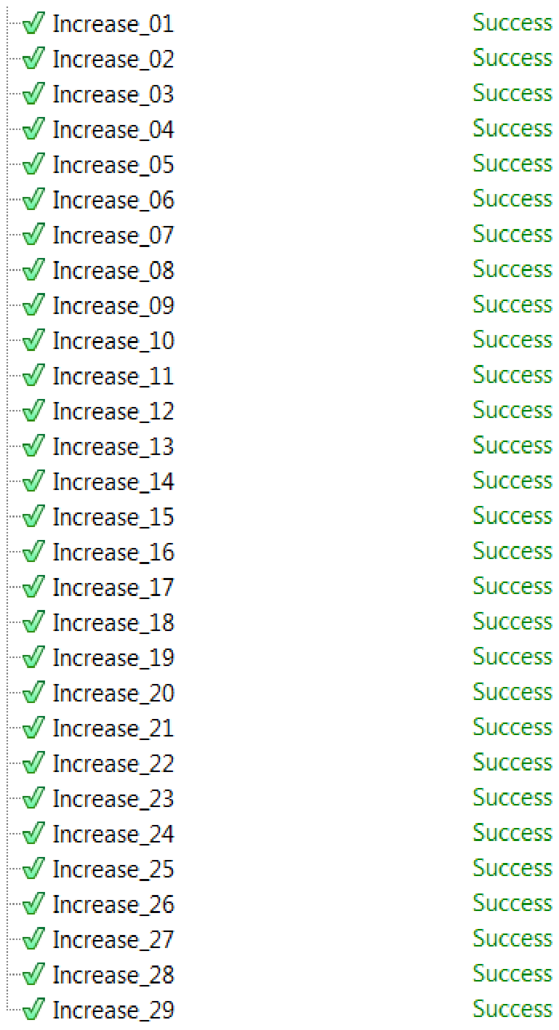
IF (return code = OK)
THEN INCREASE successful
ELSE INCREASE could not be executed		

Figure 5.3: Command sequence for Decrease and Increase commands



✓ Decrease_01	Success
✓ Decrease_02	Success
✓ Decrease_03	Success
✓ Decrease_04	Success
✓ Decrease_05	Success
✓ Decrease_06	Success
✓ Decrease_07	Success
✓ Decrease_08	Success
✓ Decrease_09	Success
✓ Decrease_10	Success
✓ Decrease_11	Success
✓ Decrease_12	Success
✓ Decrease_13	Success
✓ Decrease_14	Success
✓ Decrease_15	Success
✓ Decrease_16	Success
✓ Decrease_17	Success
✓ Decrease_18	Success
✓ Decrease_19	Success
✓ Decrease_20	Success
✓ Decrease_21	Success
✓ Decrease_22	Success
✓ Decrease_23	Success
✓ Decrease_24	Success
✓ Decrease_25	Success
✓ Decrease_26	Success
✓ Decrease_27	Success
✓ Decrease_28	Success
✓ Decrease_29	Success
✓ Decrease_30	Success
✓ Decrease_31	Success

Figure 5.4: Result for execution of test cases for Decrease command



✓ Increase_01	Success
✓ Increase_02	Success
✓ Increase_03	Success
✓ Increase_04	Success
✓ Increase_05	Success
✓ Increase_06	Success
✓ Increase_07	Success
✓ Increase_08	Success
✓ Increase_09	Success
✓ Increase_10	Success
✓ Increase_11	Success
✓ Increase_12	Success
✓ Increase_13	Success
✓ Increase_14	Success
✓ Increase_15	Success
✓ Increase_16	Success
✓ Increase_17	Success
✓ Increase_18	Success
✓ Increase_19	Success
✓ Increase_20	Success
✓ Increase_21	Success
✓ Increase_22	Success
✓ Increase_23	Success
✓ Increase_24	Success
✓ Increase_25	Success
✓ Increase_26	Success
✓ Increase_27	Success
✓ Increase_28	Success
✓ Increase_29	Success

Figure 5.5: Result for execution of test cases for Increase command

Chapter 6

File Management Commands

6.1 Introduction

Considering security aspect, several file management commands are supported by smart card operating system. Example of such commands are file creation, file deletion and file disable. If operating system support only single application then these commands often not supported. Because providing support for such commands will increase large amount of code which will increase requirement of memory size which in turn increase price of smart card.

Smart card that support multiple applications are required to have fair partition of memory and provide authorization key to create file. Since available memory will be used by all application providers. So this approach prevent one application provider from capturing entire memory.

Successful execution of authentication command followed by CREATE FILE command will lead to creation of DFs or EFs at PICC or application level at smart card. After successful creation of file, file can be selected by SELECT FILE.

Using DELETE FILE command, selected file can be removed from memory of smart card. Memory that is released by deletion of file can be used for creation of other file. Functionlity of DELETE FILE command is shown at figure 6.4.

CREATE FILE	
Command	• file type of the new file
	• IF (file type = DF) THEN [DF name of the new file]
	IF (file type = EF) THEN [
	FID of the new file
	SFI of the new file
	access conditions
	structure of the new file]
	• IF (file structure = transparent) THEN [file size]
	IF (file structure = linear fixed) OR (file structure = cyclic) THEN
	[number of records
	record length]
	IF (file structure = linear variable) THEN [
	number of records
	length of each record]
Response	• return code

Figure 6.1: Functionality of Create File command

Terminal (IFD)		Smart card (ICC)
CREATE FILE		
Command [. . .]	$\longrightarrow$	*command processing*
IF (return code = OK)	$\longleftarrow$	*Response* [return code]
THEN command successfully executed		
ELSE command failed		
UPDATE BINARY/RECORD		
Command [. . .]	$\longrightarrow$	*command processing*
IF (return code = OK)	$\longleftarrow$	*Response* [return code]
THEN command successfully executed		
ELSE command failed		

Figure 6.2: Command sequence for Create File command

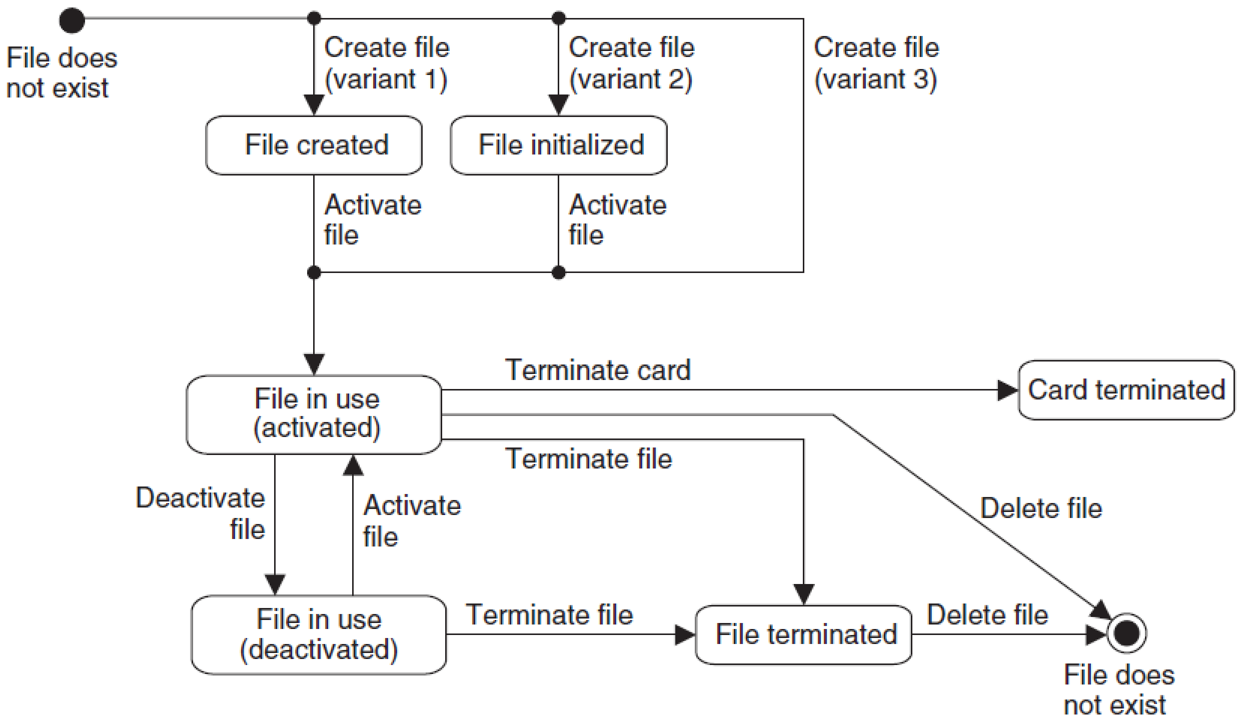
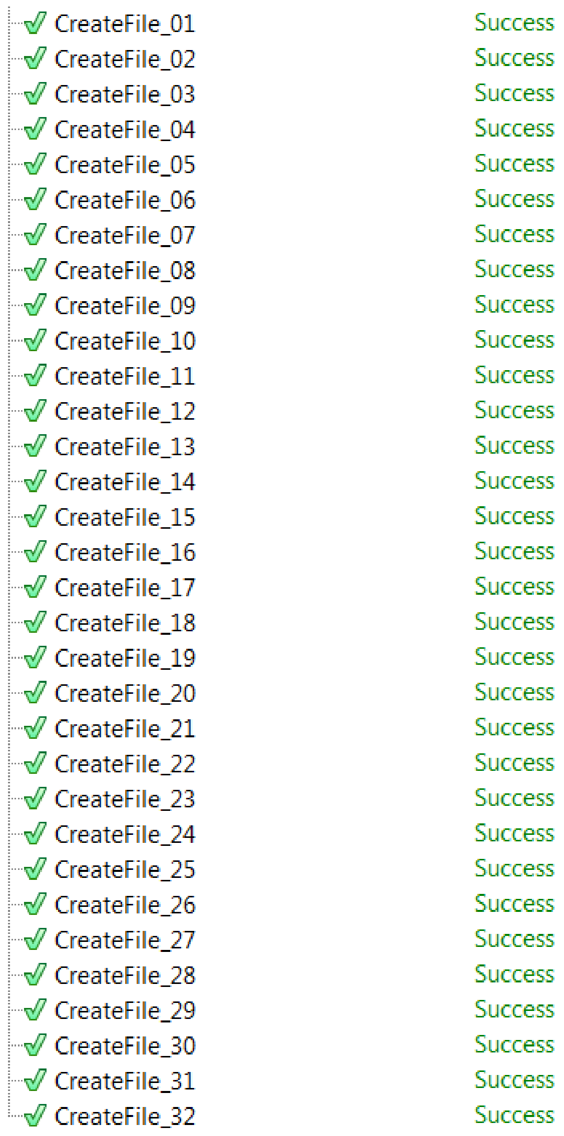


Figure 6.3: File life cycle

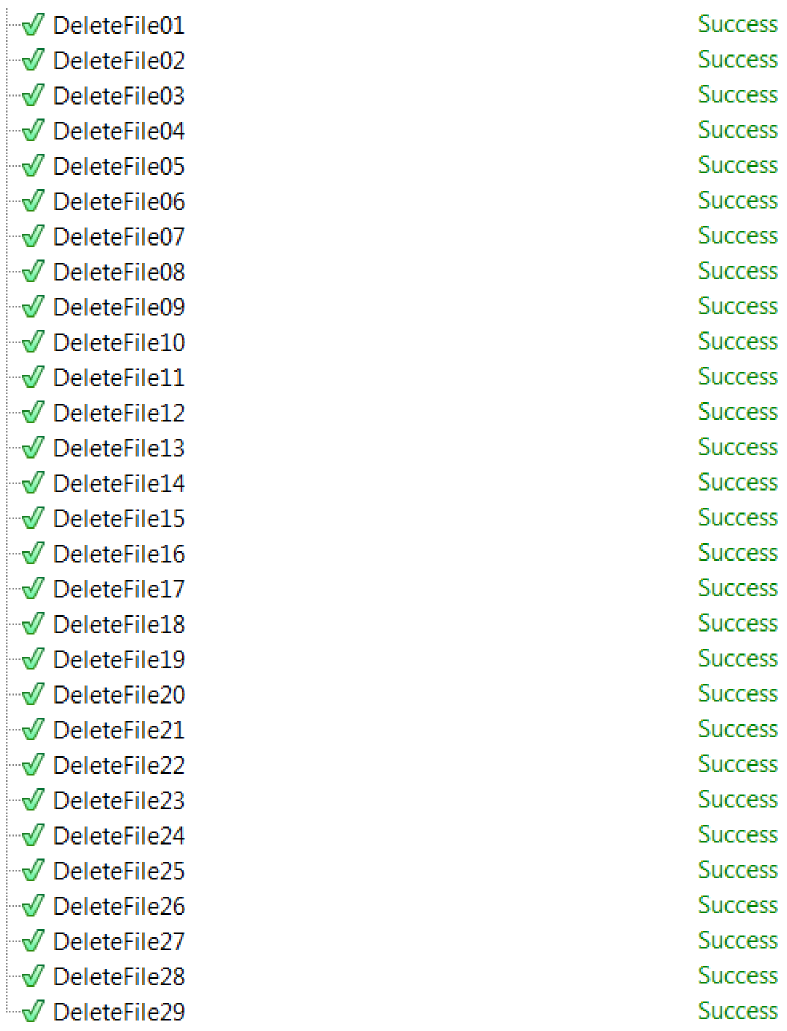
DELETE FILE	
Command	• FID *or* DF name
Response	• return code

Figure 6.4: Functionality of Delete File command



✓ CreateFile_01	Success
✓ CreateFile_02	Success
✓ CreateFile_03	Success
✓ CreateFile_04	Success
✓ CreateFile_05	Success
✓ CreateFile_06	Success
✓ CreateFile_07	Success
✓ CreateFile_08	Success
✓ CreateFile_09	Success
✓ CreateFile_10	Success
✓ CreateFile_11	Success
✓ CreateFile_12	Success
✓ CreateFile_13	Success
✓ CreateFile_14	Success
✓ CreateFile_15	Success
✓ CreateFile_16	Success
✓ CreateFile_17	Success
✓ CreateFile_18	Success
✓ CreateFile_19	Success
✓ CreateFile_20	Success
✓ CreateFile_21	Success
✓ CreateFile_22	Success
✓ CreateFile_23	Success
✓ CreateFile_24	Success
✓ CreateFile_25	Success
✓ CreateFile_26	Success
✓ CreateFile_27	Success
✓ CreateFile_28	Success
✓ CreateFile_29	Success
✓ CreateFile_30	Success
✓ CreateFile_31	Success
✓ CreateFile_32	Success

Figure 6.5: Result for execution of test cases for Create File command



✓ DeleteFile01	Success
✓ DeleteFile02	Success
✓ DeleteFile03	Success
✓ DeleteFile04	Success
✓ DeleteFile05	Success
✓ DeleteFile06	Success
✓ DeleteFile07	Success
✓ DeleteFile08	Success
✓ DeleteFile09	Success
✓ DeleteFile10	Success
✓ DeleteFile11	Success
✓ DeleteFile12	Success
✓ DeleteFile13	Success
✓ DeleteFile14	Success
✓ DeleteFile15	Success
✓ DeleteFile16	Success
✓ DeleteFile17	Success
✓ DeleteFile18	Success
✓ DeleteFile19	Success
✓ DeleteFile20	Success
✓ DeleteFile21	Success
✓ DeleteFile22	Success
✓ DeleteFile23	Success
✓ DeleteFile24	Success
✓ DeleteFile25	Success
✓ DeleteFile26	Success
✓ DeleteFile27	Success
✓ DeleteFile28	Success
✓ DeleteFile29	Success

Figure 6.6: Result for execution of test cases for Delete File command

Chapter 7

Data Transmission

7.1 Introduction

There must be agreement between two communicating party to avoid possible collision while data transmission, which specify which of the two parties will initiate transmission. For smart card and reader, reader will start communication since it works as master. Smart card works as slave.

Transmission of data starts with reset signal sent by reader to smart card. In response to this reset signal, smart card replies with 'answer to reset'(ATR). In response to ATR, PPS(protocol parameter selection) is optionally sent. PPS is used for setting parameter for following data transmission.

- Answer to Reset(ATR)
  After receiving reset signal, smart card sent out ATR(answer to reset). ATR contain information regarding data transmission rates and transmission protocol supported by the smart card.

- Protocol Parameter Selection(PPS)
  Protocol Parameter Selection is used to change value of parameter from defined in ATR. Transmission of PPS is optional.

- Transmission Protocol

  For contactless smart card, ISO/IEC 14443 standards are used. It specifies characteristics of contactless smart card with maximum distance of 10 cm from reader.

At application level, data record is known as APDU(Application protocol data unit). Command header and command body makes command APDU. Here header is mandatory and body part is optional. A response body and response trailer makes response APDU. For response APDU, only response trailer is mandatory.

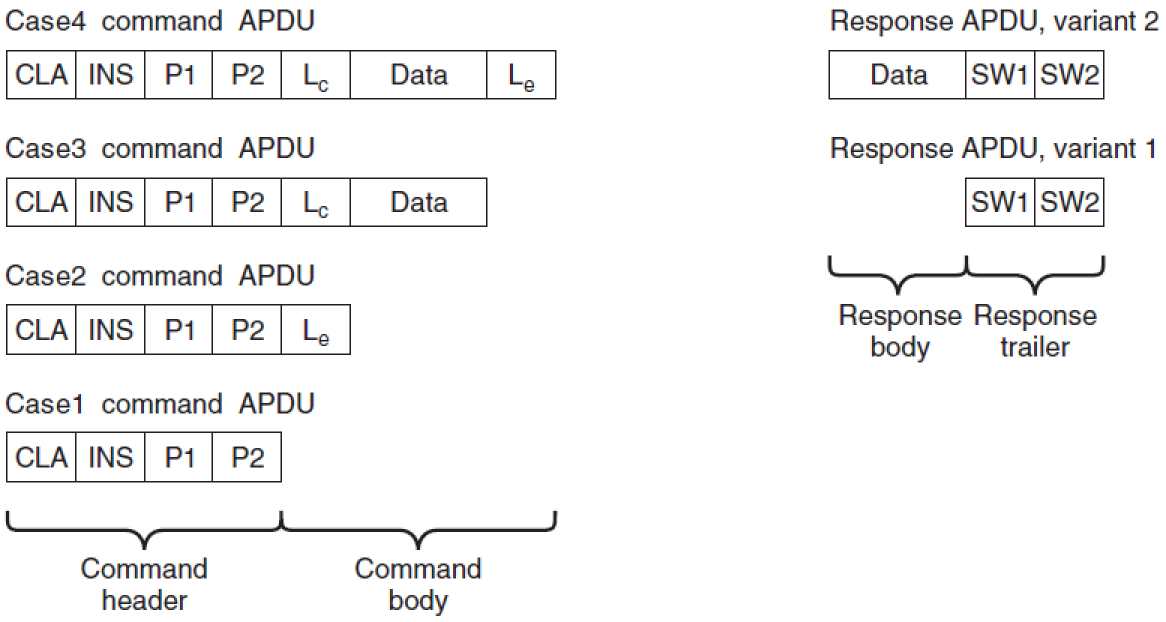


Figure 7.1: Command APDU and response APDU

Four different variants are shown at figure 7.1. Command APDU consist of four bytes: Class (CLA), Instruction (INS), Parameter 1 (P1) and Parameter 2 (P2). Class (CLA) indicate standard used to send command. Instruction(INS) indicate command code. Parameter P1 and P2 is used to provide information that is required for execution of command.

The command body at max contain three data elements. Lc indicate data length in command. Le indicate length of expected data in response.

For command APDU, four combination are allowed, out of which each combination is denoted as case. For response APDU, two combination are allowed. Variant 1 as shown at figure 7.1, is used when expected data length from smart card is zero. It menas no data is expected in response and only status word is returned, which show status of execution of command. Variant 2 is used when data is expected from smart card in response of execution of command.

Chapter 8

Conclusion and future scope

8.1 Conclusion

Implementation of commands for smart card is carried out. Implementation is followed by verification. Here commands covered are File Selection, Read Binary, Update Binary, Decrease, Increase, Create File and Delete File. It is complying partially with ISO 7816 and fully with ISO 14443. Verification is done on various platform like CSIM, FPGA and engineering sample.

8.2 Future scope

- This product can be further optimize with an intention to develop low cost smart card which required optimization at both aspect software and hardware.

Chapter 9

Gantt Chart

Gantt Chart for the project is shown in figure 9.1.

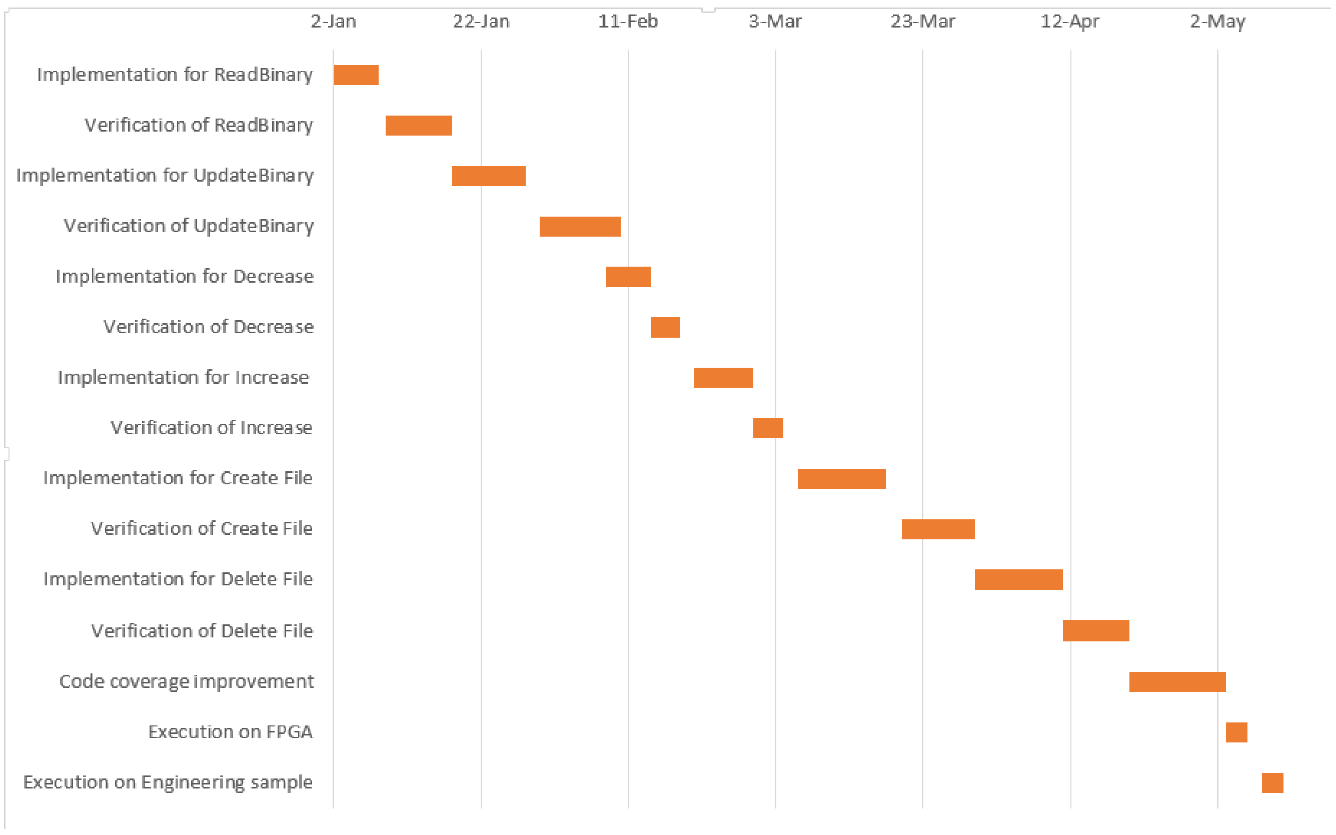


Figure 9.1: Gantt Chart

Bibliography

[1] P. Paradinas and J.-J. Vandewalle, "New directions for integrated circuit cards operating system,"

[2] "http://www.smartcardalliance.org/smart-cards-applications,"

[3] A. Tanenbaum, *Modern operating system.*

[4] W. R. Bevier, "A study in operating system verification," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 15, no. 11, pp. 1382–1396, 1989.

[5] A. M. B.-E. Mohammad R. Eletrib, Mohamed Sobh and H. M. Fahmy, "High performance java card operating system," *2014 Eighth International Conference on Software Security and Reliability*, 2014.

[6] J. A. Keane, "A method of verification in design: an operating system case study," *Hawaii International Conference on System Sciences.*

[7] W. Rankl, "Smart card application," 1997.

[8] R. F. K. P.-R. T. Bogdan Marculescu, Simon Poulding, "Tester interactivity makes a difference in search-based software testing: A controlled experiment," *Information and Software Technology*, vol. 78, pp. 66–82, 2016.

[9] W. Rankl and W. Effing, "Smart card handbook," 2015.