Security Classification:

EIDA ID Card Toolkit v2.7

Developer's Guide

Document Details

Organization	Emirates Identity Authority (EIDA)
Document Title	EIDA ID Card Toolkit developers guide for Java and .NET developers
Date	31-10-2012
Doc Name / Ref	
Classification	◉ Public ○ Internal ○ Confidential ○ Highly Confidential
Document Type	○ Policy ○ Procedure ○ Form/Template ○ Report ◉ Other

Document History

Date	Version	Author	Comments
15-8-2010	0.1		
18-8-2010	0.2		
19-8-2010	1.0		
5-12-2010	2.1		
2-8-2011	2.2		
5-11-2011	2.2.2 v0.90		Revised the contents
15-11-2011	2.2.2 v0.91		Reviewed and made the document user friendly.
16-11-2011	2.2.2 v0.92		Final review
10-01-2012	V 2.3		Update for Version 2.3
21-02-2012	V 2.3		Update to reflect the amendments to Web components
21-03-2012	0.1 for Toolkit 2.4		Released as part of Toolkit 2.4
10-04-2012	V 2.4		Final review.
2-5-2012	V 2.5		Released as part of Toolkit 2.5
27-05-2012	V 2.6		Released as part of Toolkit 2.6 to cover additional containers and update modifiable data
29-10-2012	V 2.7		Released as part of Toolkit 2.7 to cover Zero footprint web components and ID Box one MRZ scanner integration

Contents

Table of Figures

Definitions

Abbreviation	Description
API	Application Programming Interface
BIT	Biometric Information Template
DLL	Dynamic Link Library
EIDA	Emirates Identity Authority
HSM	Hardware Security Module
PIN	Personal Identification Number
SAM	Security Access Module
SDK	Software Development Kit
SM	Secure Messaging
VB	Visual Basic

1 Introduction

This document is intended as a guide for developers who can build business applications using the EIDA ID Card Toolkit SDK which provides platform to access set of UAE ID Card functions and features. .

The SDK provides Java and .NET interfaces for developing applications thus allowing organisations to create applications using C++, C#, VB and Java programming languages.

Detailed descriptions of the ID Card Toolkit functions are explained in section 5 and 6 of this document along with examples.

Note: This document is intended for Java and C# application developers. There is a separate developer's guide available for C++ application developers.

Important pre-requisites:

1) Development experience in Java, C# and/or C++, and some specificity of the languages is mandatory to develop applications using UAE ID Card Toolkit SDK.
2) Knowledge and experience in smart card field is necessary.
3) Users should receive required training from EIDA

2 Compatibility

EIDA ID Card Toolkit SDK is built around a C++ core library designed to run on Windows Operating Systems. The current version of the Toolkit is designed to work on the below operating Systems / programming languages.

Platforms: (Win32 / Win64)

- Windows XP
- Windows Vista
- Windows 2003 Server
- Windows 2008 Server
- Windows 7

Programming languages

- C/C++
- Java
- .Net languages (VB, C#, …)

IDE and Compilers:

- Microsoft Visual Studio 2005 (Or Express Edition)
- Microsoft Visual Studio 2008 (Or Express Edition)
- Eclipse 3.x

Java

JDK 1.6 or higher

.Net Framework

.Net Framework 3.5 or higher

3 Installation of Toolkit

Before starting the development, EIDA ID Card Toolkit SDK must be installed. Refer to "EIDA ID Card Toolkit Installation and Configuration Guide" document for the installation, prerequisites and installation steps.

3.1 Toolkit Components

The Toolkit setup will automatically copy some or all of the below components based on the options chosen. The below table provides the list of component names, physical file name (if applicable), and their high-level description that are part of the EIDA ID Card Toolkit package. These components will be referred in the later part of this document.

Component Name	Physical file	Description
Core Dlls	UAE_IDCardLib.dll	Core components of the Toolkit.
	Wrappers, helper Dlls	
Java API	UAE_IDCardJavaAPI.jar	Java API and core Dll wrapper library.
Secure Messaging API	SecureMessagingAPI.jar	Used with java API for local and remote secure messaging.
Public Data Parser	PublicDataParser.jar	Public data files parser and signature validator
ID Box One MRZ scanner API	IDBoxMrz.jar	Interface with ID Box One MRZ scanner
.NET API	UAE_IDCardCSharpAPI.dll	
	UAE_IDCardCSharpWrapper.dll	.NET wrapper for the Core Dll.
Public Data Parser	PublicDataParser.dll	Public data files parser and signature validator
ID Box One MRZ scanner API	IDBoxMrz.dll	Interface with ID Box One MRZ scanner
Web Components	DigitalSignatureActiveX.dll	An Activex exposes all PKI functions for web based applications.
	PublicDataActiveX.dll	An Activex exposes all other Toolkit functions for web based applications.

	EIDA_IDCard_Applet.jar	A java Applet exposes all Toolkit functions for web based applications.
Zero footprint web components	EIDA_ZF_ActiveX.CAB	A zero footprint ActiveX component that has no client dependencies. It does does not require Toolkit to be installed, and used to read public data raw files, data signing, and certificate exporting
	ZFApplet.jar	A zero footprint Java Applet component that has no client dependencies. It does not require Toolkit to be installed, and it can be used to read public data raw files, data signing, and certificate exporting
Sample Desktop Application	Java	Java Sample Application and batch file to launch the Java Sample Application.
	dotNet	.Net Sample Application
	Console	Sample C++ console application to demonstrate Toolkit kernel functions.
	sm.cfg	Secure Messaging configuration file.
Sample Web Components	Public Data ActiveX and Digital Signature ActiveX folders	These folders contain sample HTML files with the ActiveX embedded in it, and also javascript helper files.
	IDCard Applet folder	This folder contains a sample HTML file with the Java Applet embedded, and also a helper javascript files.
Web Services	Java	This folder contains java based secure messaging web service. It is a java servlet which provides a remote access to SM modules (SAM or HSM).
	dotNet	This folder contains the .NET based secure messaging web service. It is an ASPX application which provides a remote access to SM modules

		(SAM or HSM).
Website	Java	ZFDemoSite.war contains website implemented in JSP and Servlets to demonstrate Zero footprint web components and complementary server side components.
	.Net	ZFDemoSite contains website implemented in ASP.NET to demonstrate Zero footprint web components and complementary server side components.

4 Development environment

Developers can use either Eclipse or Microsoft Visual Studio development tools. This section provides detailed steps on how to create a new project (application) in Eclipse and Visual Studio.

Note: Prior to setting up the development environment, based on the target application platform either 32 bit or 64 bit version of EIDA ID Card Toolkit must be installed.

4.1 Developing Java based application using Eclipse

Install the Eclipse development tool from the below website:

http://www.eclipse.org/downloads

After successful installation of Eclipse tool, follow the below steps to create a new project to develop an application using ID Toolkit. Refer to the screen snapshots for assistance.

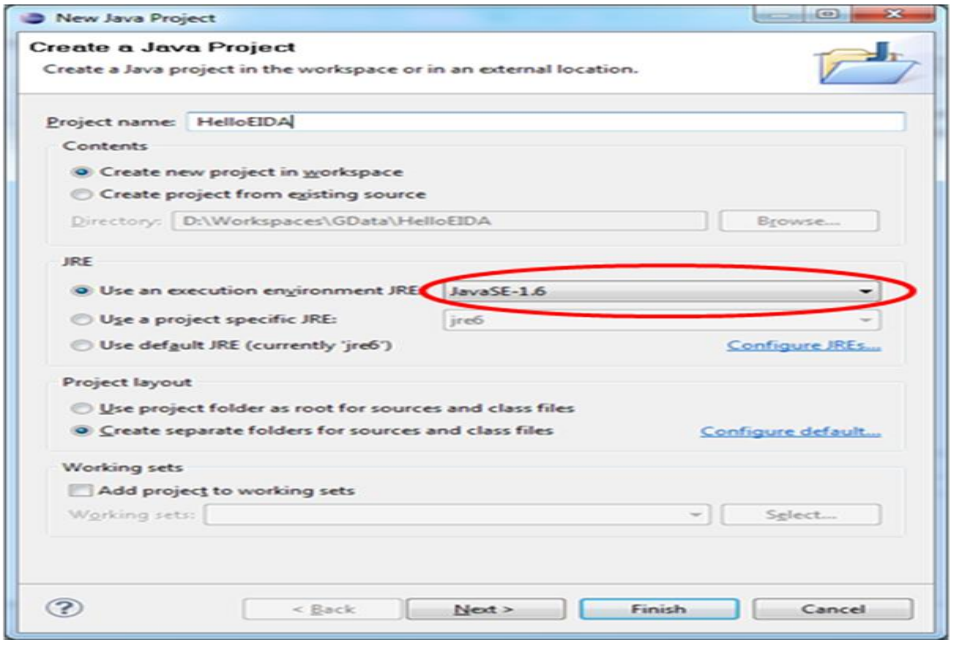
a) Run eclipse.exe

b) Select a workspace

c) Go to File -> New -> Java Project

d) Enter a project name

e) Make sure to select JRE 6 (x86) or (x64)



f) Click Finish

g) Make sure a new project is created

h) Select the created project in package explorer

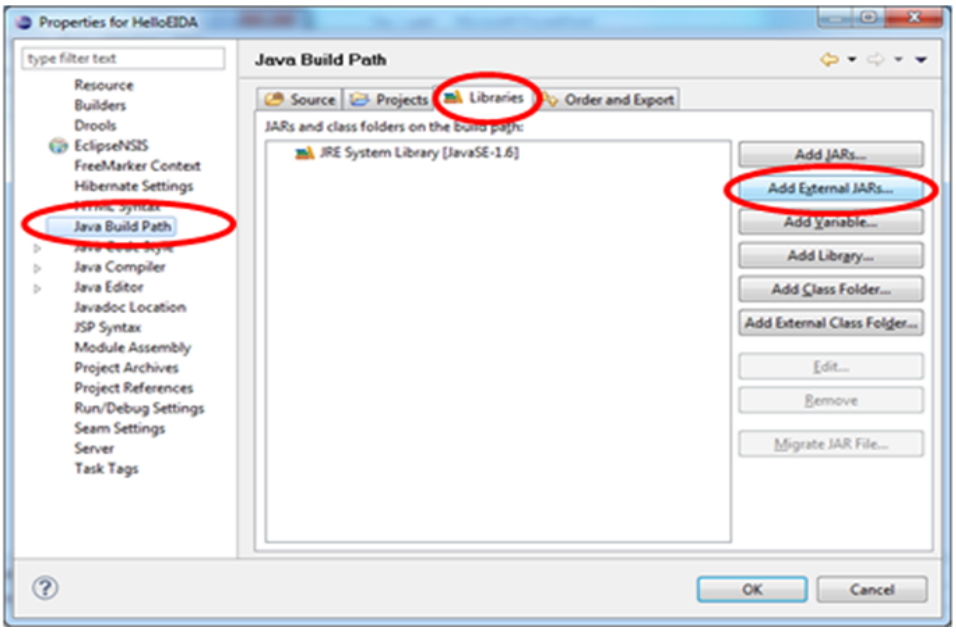
Note: If you are working on an existing project, go to point i directly.

i) Go to Project -> Properties

j) Select Java Build Path

k) Go to Libraries Tab

l) Click “Add External JARs” button

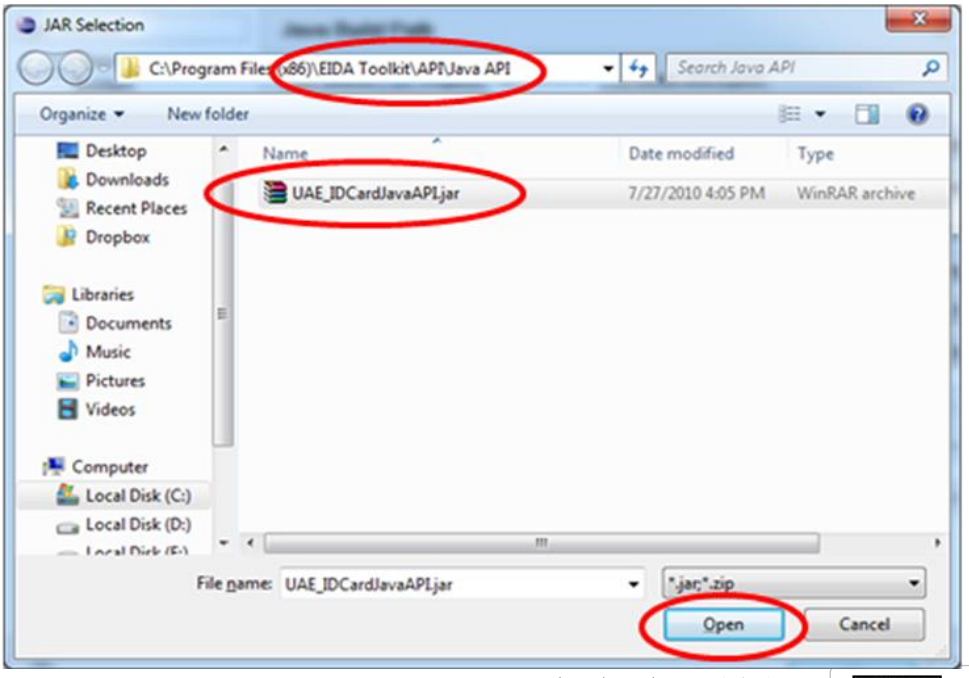


m) Navigate to the path where Toolkit SDK is installed

n) Open “API” folder

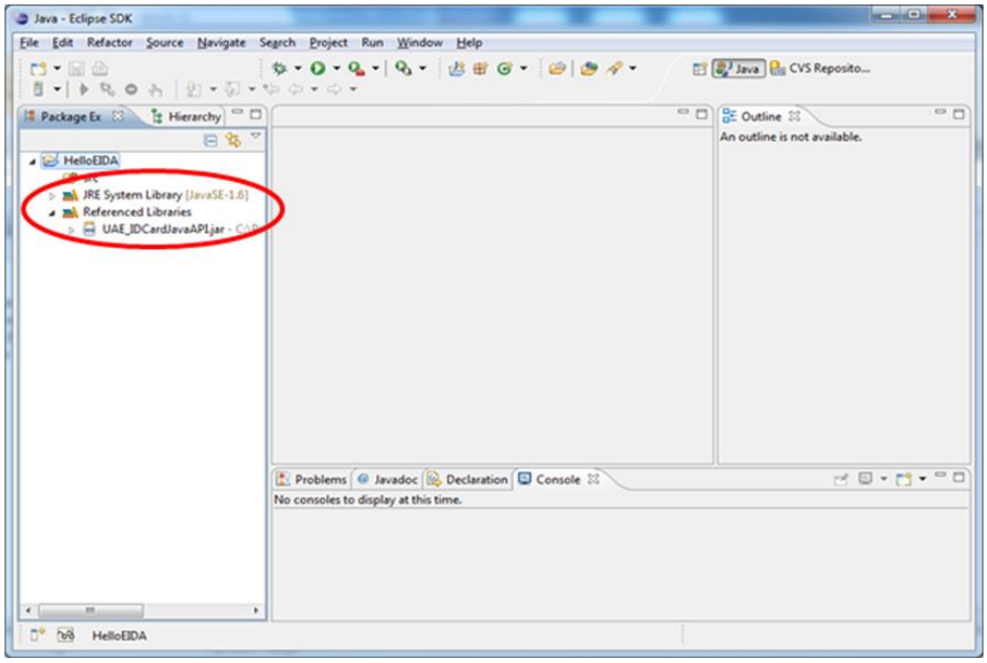
o) Open “Java API” folder

p) Select “UAE_IDCardJavaAPI.jar” file



q) Click “Open” button

r) Click “Ok” button



Repeat the previous steps to add “SecureMessagingAPI.jar” if secure messaging is needed in your application.

Refer to the section 5 for various functions available in the Toolkit to build your application and alternatively you may use the sample applications to familiarise yourself before starting the development of your own.

4.2 Developing .NET based application using Microsoft Visual Studio

Install the Microsoft Visual Studio 2005 or later.

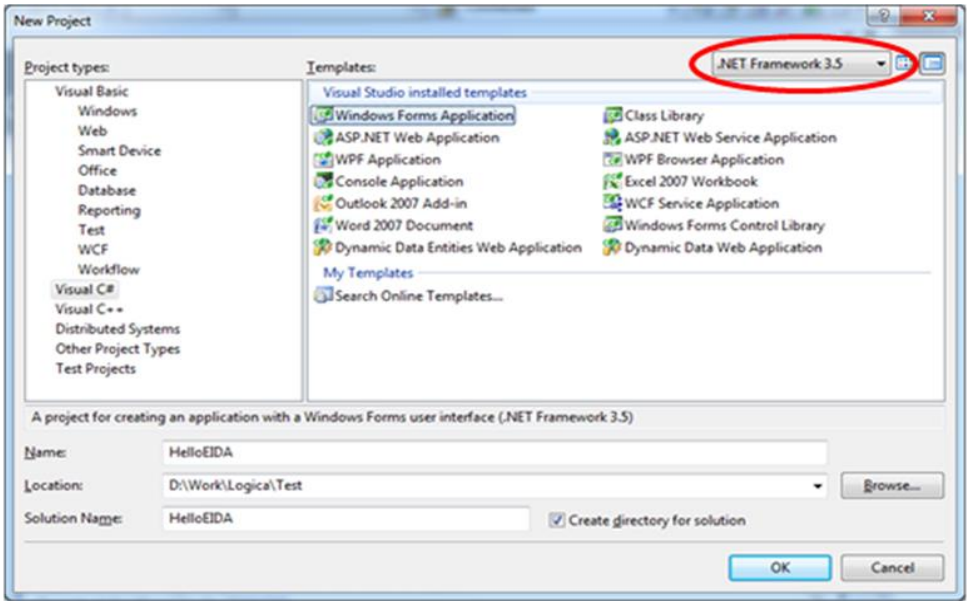
Follow the below steps to create a new project to develop an application using ID Toolkit. Refer to the sample screen snapshots for assistance.

a) Run Microsoft Visual Studio

b) Go to File -> New -> Project

c) Enter a project name

d) Select Visual C# as a project type



e) Make sure .NET Framework 3.5 is selected

f) Select the project in project explorer

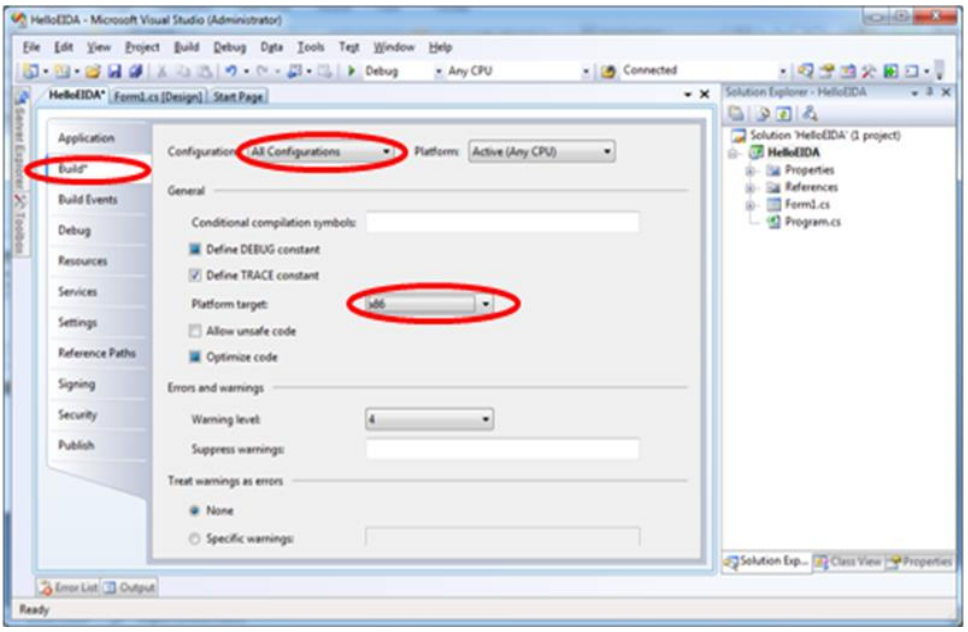
g) Go to Project -> Properties

h) Go to “Build” Tab

i) Select “All Configurations” From “Configuration” list

j) Then select “x86” or “x64” from “Platform target” list

k) Save the changes



l) Go to Project -> Add Reference…

m) Go to “Browse” tab

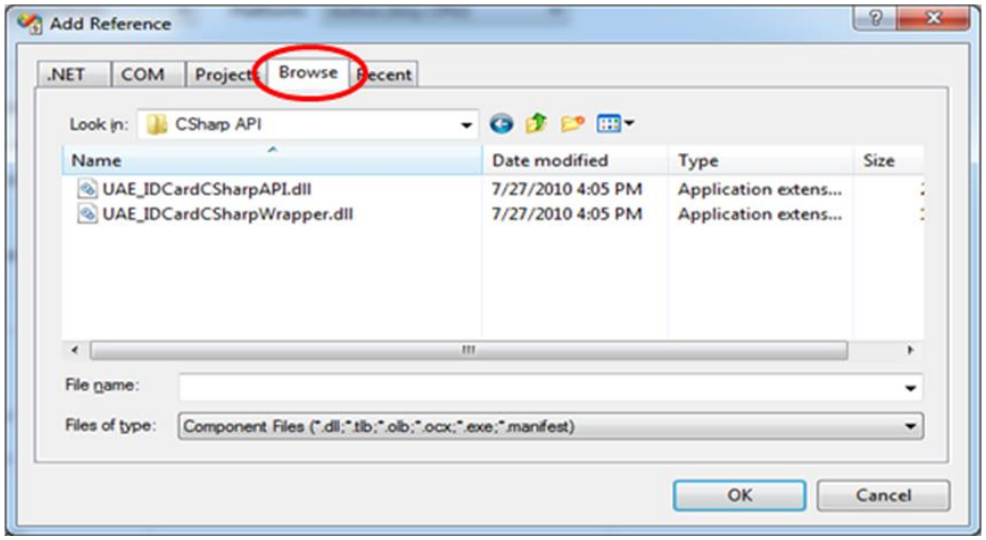
n) Navigate to the path where Toolkit SDK is installed

o) Open “API” folder

p) Open “CSharp API” folder

q) Select “UAE_IDCardCSharpAPI.dll” and “UAE_IDCardCSharpWrapper.dll”

r) Click “Ok”



Refer to the section 5 for various functions available in the Toolkit to build your application and alternatively you may use the sample applications to familiarise yourself before starting the development of your own.

5 EIDA ID Card Toolkit functions

This section lists the standard functions that are exposed by the EIDA ID Toolkit API for developing EIDA ID card based applications. Related functions are grouped and explained in the sub-sections of this chapter with example code (C# and Java).

Note for Java Developers: For detailed list of Classes / Functions that are available in the Toolkit, please refer to the Java Docs html pack.

Note for .NET Developers: For detailed list of Classes / Functions that are available in the Toolkit, please refer to the .Net help pack.

Also developers are advised to refer to Appendix B which describes the sequence of API calls a developer needs to undertake to access to the main business functions of the Toolkit.

5.1 Establishing and Closing Context

In order to support a wide range of smart card readers, EIDA ID Card Toolkit uses the PC/SC standard to communicate with any PC/SC compatible smart card readers and hence PC/SC context must be established first before using any other functions of ID Card Toolkit. The context is initialized by calling the method establishContext().

Before closing the application, it is important to invoke closeContext() which will Close the context, release the resources and clear the memory.

Note: An application should invoke the functions establishContext() and closeContext() only once.

Below examples shows how to open and close context in Java and C# respectively.

Java Example

```
import emiratesid.ae.*;
import emiratesid.ae.readersmgt.*;

// ... Some code here ...
try {
      ReaderManagement readerMgr = new ReaderManagement();
      readerMgr.establishContext();
      // do ID Card Operations
      // ……
      readerMgr.closeContext();
      }  catch (MiddlewareException ex) {
            ex.printStackTrace();
      }
```

C# Example

```
using EmiratesId.AE;
using EmiratesId.AE.ReadersMgt;

//... Some code here ...

try
   {
     ReaderManagement readerMgr = new ReaderManagement();
     readerMgr.EstablishContext();
     // ... do some ID Card operations ...
     readerMgr.CloseContext();
   }
   catch (MiddlewareException ex)
   {
    //… handle exception here …
   }
```

5.2 Discovering, Connecting and Disconnecting Readers

Once a context is established, invoking discoverReaders() function will discover all the PC/SC readers connected to the local machine. An application can then use any of the available readers to access the card functions

In case of multiple readers connected to the machine, a specific reader can be selected from the list of discovered readers by name, or by type, or by index, etc...

ReaderManagement class provides the below functions for selecting readers.

- **SelectReaderByName**

This function selects a specific reader with the given name.

- **SelectReaderByATR**

  This function selects the first available reader that has ID card with the given ATR

- **SelectTestIDCardReader**

  This function selects the first available reader with Test ID card inserted in the reader.

  Card is identified by its ATR; the Toolkit is configured with a list of possible EIDA Test card ATRs. Please refer to the sm.CFG configuration file in Appendix A for configuration details.

- **SelectIDCardReader**

  This function selects the first available reader with Live ID card inserted.

- Card is identified by its ATR; the toolkit is configured with a list of possible EIDA Live card ATRs. Please refer to the sm.CFG configuration file in Appendix A for configuration details.**SelectIDCardContactlessReader**

  This function selects the first available contactless reader with Live ID card in range.

Card is identified by its ATR; the toolkit is configured with a list of possible EIDA Live card ATRs. Please refer to the sm.CFG configuration file in Appendix A for configuration details.

Note: if any of the select functions above didn't find a reader with the required criteria it returns a null PCSCReader object.

The below examples provide the usage of those functions in Java and C# respectively.

Java Example

```
readerMgr.discoverReaders();
PCSCReader[] readers = readerMgr.getReaders();
// select a reader by name code goes here
// ...
PCSCReader selectedReader =
readerMgr.selectReaderByName(selectedReaderName);
//Other select methods may be called...
selectedReader.isConnected();
boolean isUAE = ATRSetting.Is_UAE_Card(selectedReader.getATR());
```

C# Example

```
readerMgr.DiscoverReaders();
PCSCReader[] readers = readerMgr.readers;
// select a reader by name code goes here
// ...
PCSCReader selectedReader =
readerMgr.SelectReaderByName(selectedReaderName);
//Other select methods may be called...
selectedReader.IsConnected();
bool isUAE = ATRSetting.Is_UAE_Card(selectedReader.ATR);
```

5.3 Load SM Configuration

In order to use EIDA secure messaging in local mode, the function IDCardWrapper.LoadConfiguration shall be called to load the secure messaging modules configurations from the sm.cfg file. Sample configuration of is described in appendix A.

The below examples provide the usage of those functions in Java and C# respectively.

Java Example

```
UAEIDCardLibJavaWrapper.LoadConfiguration();
```

C# Example

```
IDCardWrapper.LoadConfiguration();
```

5.4 Establishing connection with the card

Scenario 1: If the EIDA ID card was inserted before selecting the reader then the Toolkit will automatically establish the connection with the card and hence the reader will be in a connected state (the function IsConnected will return TRUE if the ID card was inserted already).

Scenario 2*:* If the EIDA ID card was not inserted before selecting the reader, then the Connect function must be called to establish the connection with card whenever the card is inserted, the Connect function takes the Context parameter as input.

Note that the PC/SC context was opened before using the ReaderManagement class, the value of this parameter can be acquired from the property “Context” available in the ReaderManagement class.

Tip: The function Connect can be used to detect the card insertion, where Connect function is called then IsConnected function should be called to detect if the card is connected.

With Contactless readers support for V2 cards, there is a need to differentiate between contactless and normal PCSC reader, therefore the function IsContactless() exists in PCSC reader class.

Below examples, provide the usage of the above functions in Java and C# respectively.

Java Example

```
PCSCReader selectedReader =
readerMgr.selectReaderByName(selectedReaderName);
boolean isCardConnected = selectedReader.isConnected();
boolean isContactless;
if(isCardConnected)
  isContactless = selectedReader.isContactless();

if(!isCardConnected)
  //use the context opened before using the ReaderManagement object
  selectedReader.Connect(readerMgr. Context);
```

C# Example

```
PCSCReader selectedReader =
readerMgr.SelectReaderByName(selectedReaderName);
selectedReader.IsConnected();
bool IsCardConnected = selectedReader.IsConnected();
bool isContactless;
if(isCardConnected)
  isContactless = selectedReader.IsContactless();

if(!IsCardConnected)
//use the context opened before using the ReaderManagement object
selectedReader.Connect(readerMgr. Context);
```

5.5 Reading Card Related Information

Once PCSCReader object is acquired in a CONNECTED state and with right type, application can extract the EIDA ID Card related information such as Card Serial Number, and Chip Serial Number.

The retrieved information will be in binary format, using the format conversion functions of the Toolkit. Developers can convert data from binary format to string representation. Refer to Utils class for a sample conversion implementation.

The below examples provide the usage of both the functions in Java and C# respectively.

Java Example

```
import emiratesid.ae.readersmgt.*;
import emiratesid.ae.utils.*;

CardInfo cardInfo = reader.getCardInfo();
try {
      char[] csn = cardInfo.getCardSerialNumber();
      char[] chipSN = cardInfo.getChipSerialNumber();
      char[] cplc0101 = cardInfo.getCPLC0101();
      char[] cplc9f7f = cardInfo.getCPLC9F7F();
      char[] isn = cardInfo.getIssuerSerialNumber();
      char[] irn = cardInfo.getIssuerReferenceNumber();
      char[] mocSN = cardInfo.getMOCSerialNumber();
      char[] mocAppState = cardInfo.getMOCAppletState();
      char[] mocAlgVer = cardInfo.getMOCAlgorithmVersion();
      long maxFailed = cardInfo.getMaxFailedMatch();
      int cardVersion = cardInfo. getCardVersion();

      String csnHex = Utils.CharArrayToHex(csn);
} catch (MiddlewareException e) {
      e.printStackTrace();
}
```

C# Example

```
using EmiratesId.AE.ReadersMgt;
using EmiratesId.AE.Utils;

CardInfo cardInfo = reader.GetCardInfo();
try {
byte[] csn = cardInfo.GetCardSerialNumber();
byte[] chipSN = cardInfo.GetChipSerialNumber();
byte[] cplc0101 = cardInfo.GetCPLC0101();
byte[] cplc9f7f = cardInfo.GetCPLC9F7F();
byte[] isn = cardInfo.GetIssuerSerialNumber();
byte[] irn = cardInfo.GetIssuerReferenceNumber();
byte[] mocSN = cardInfo.GetMOCSerialNumber();
byte[] mocAppState = cardInfo.GetMOCAppletState();
byte[] mocAlgVer = cardInfo.GetMOCAlgorithmVersion();
byte[] maxFailed = cardInfo.GetMaxFailedMatch();
int cardVersion = cardInfo. GetCardVersion();

String csnHex = Utils. ByteArrayToHex(csn);
} catch (MiddlewareException e) {
      // … Handle exceptions here
}
```

5.6 Reading Card Holder Public Data

Using a connected PCSCReader object, application can read the Card holder public data from the card. The retrieved information will be in binary format, and hence it should be converted to String format before using them. Refer to 'Utils' Class for sample conversion implementation.

Some of the data retrieved are Code which corresponds to a description, for example Card holder occupation is stored on the card as occupation code. To map the code into actual occupation, refer to the occupations (lookups) table published by EIDA. It is the developer's responsibility to map the relevant code to the corresponding description.

Note: Codes lookups should be acquired regularly from EIDA

In order to optimize the performance of reading public data from the card, below flags (parameters) are available to read specific set of data instead of reading all the public data which is huge:

- **Refresh flag:** set to true to read the data from the card and also to enforce the refreshing of the cached data, set it to false to read the already cached data (from memory).
- **Photography flag** : set to true to read the photography, false otherwise
- **Non-modifiable data flag:** set to true to read the non-modifiable data area on the ID card, false otherwise.
- **Modifiable data flag:** set to true to read the modifiable data on the ID card, false otherwise.
- **Validate signature flag:** set it to true to validate the signatures in public data files on the ID card, false otherwise.

  The toolkit verifies the signature using the data signing certificates located in the folder location configured in sm.cfg file. Please refer to Appendix A for more details on how to configure the signing certificates folder location. EIDA has issued multiple signing certificates therefore all of them must exist in the configured folder location otherwise if the signing certificate corresponding to the card couldn't be found while "SignatureValidation" flag set to true, an error returned if the certificate is not found. Please Note that the toolkit setup is copying all the signing certificates tell the date of releasing version 2.5 of the toolkit, EIDA will deploy new certificates in future by distributing a dedicated toolkit service packs .

A successful call to this function populates an instance of the class CardHolderPublicData with the read public data fields and then these fields can be retrieved by calling member function dedicated for each data filed in this Class, The list of data fields in this class are listed in a table in section 6.5.3

Below examples provide the usage of both the functions in Java and C# respectively.

Java Example

```
import emiratesid.ae.publicdata.*;

// ... some code here ...

try {
PublicDataFacade publicDataFacade = reader.getPublicDataFacade();
CardHolderPublicData publicData = publicDataFacade.readPublicData(true,
true, true, true, true);

char[] fullNameBin = publicData.getFullName();
String fullName = Utils.CharArrayToUTF8String(fullNameBin);
char[] sexBin = publicData.getSex();
String sex = Utils.CharArrayToUTF8String(sexBin);
char[] issueDateBin = publicData.getIssueDate();
String issueDate = Utils.CharArrayToStringDate(issueDateBin);
char[] photography = publicData.getPhotography();

// use publicData.getXXX as needed
//...

} catch (MiddlewareException e) {
      // ... Handle exceptions here
}
```

C# Example

```
using EmiratesId.AE.PublicData;

try {
PublicDataFacade publicDataFacade = reader.GetPublicDataFacade();
CardHolderPublicData publicData = publicDataFacade.ReadPublicData(true,
true, true, true, true);

byte[] fullNameBin = publicData.GetFullName();
String fullName = Utils.CharArrayToUTF8String(fullNameBin);
byte[] sexBin = publicData.GetSex();
String sex = Utils.CharArrayToUTF8String(sexBin);
byte[] issueDateBin = publicData.GetIssueDate();
String issueDate = Utils.CharArrayToStringDate(issueDateBin);
byte[] photography = publicData.GetPhotography();

// use publicData.getXXX as needed
//…

} catch (MiddlewareException e) {
      // … Handle exceptions here
}
```

5.7 Reading Card Holder Public Data Extended

Read Card Holder Public Data feature is extended to support reading additional public data fields added in V2 cards such as address, passport information, Company name, Qualification, Field of Study, etc...This function expects the same parameters of the function “readPublicData” and additionally the below parameters:

ReadV2Fields: set it to true to read V2 data specified above(if the flag is set to true while the card inserted is V1 cards, the function throws an error)

ReadSignatureImage: only if the above parameter is true, this parameter determines to read the signature image or not.

ReadAddress: only if the **ReadV2Fields** parameter is true, this parameter determines to read the Home and Work address fields or not.

A successful call to this function populates an instance of the class CardHolderPublicDataEx with the read public data fields and then these fields can be retrieved by calling member function dedicated for each data filed in this class. The list of data fields in this class can be identified from the table in section 6.5.4.

Below examples provide the usage of both the functions in Java and C# respectively.

Java Example

```
import emiratesid.ae.publicdata.*;

try {
PublicDataFacade publicDataFacade = reader.getPublicDataFacade();
CardHolderPublicDataEx publicDataEx =
publicDataFacade.readPublicDataEx(true, true, true, true, true,true,
true, true);

char[] fullNameBin = publicDataEx.getFullName();
String fullName = Utils.CharArrayToUTF8String(fullNameBin);
char[] sexBin = publicDataEx.getSex();
String sex = Utils.CharArrayToUTF8String(sexBin);
char[] issueDateBin = publicData.getIssueDate();
String issueDate = Utils.CharArrayToStringDate(issueDateBin);
char[] photography = publicDataEx.getPhotography();

char[] FieldofStudyEnglishBin=publicDataEx.getFieldofStudyEnglish();
String FieldofStudyEnglish= Utils.CharArrayToUTF8String
(FieldofStudyEnglishBin);

char[] FieldofStudyArabicBin=publicDataEx.getFieldofStudyArabic();
String FieldofStudyArabic= Utils.CharArrayToUTF8String
(FieldofStudArabicBin);


// use publicDataEx.getXXX as needed
//...

} catch (MiddlewareException e) {
      // ... Handle exceptions here
}
```

C# Example

```
using EmiratesId.AE.PublicData;

try {
PublicDataFacade publicDataFacade = reader.GetPublicDataFacade();
CardHolderPublicDataEx publicDataEx =
publicDataFacade.ReadPublicDataEx(true, true, true, true, true,true,
true, true);

byte[] fullNameBin = publicDataEx.GetFullName();
String fullName = Utils.ByteArrayToUTF8String(fullNameBin);
byte[] sexBin = publicDataEx.GetSex();
String sex = Utils. ByteArrayToUTF8String(sexBin);
byte[] issueDateBin = publicDataEx.GetIssueDate();
String issueDate = Utils.ByteArrayToStringDate(issueDateBin);
byte[] photography = publicDataEx.GetPhotography();
byte[] FieldofStudyEnglishBin=publicDataEx.getFieldofStudyEnglish();
String FieldofStudyEnglish= Utils.ByteArrayToUTF8String
(FieldofStudyEnglishBin);

byte[] FieldofStudyArabicBin=publicDataEx.getFieldofStudyArabic();
String FieldofStudyArabic= Utils.ByteArrayToUTF8String
(FieldofStudArabicBin);


// use publicDataEx.getXXX as needed
//…

} catch (MiddlewareException e) {
      // … Handle exceptions here
}
```

5.8 Read Public Data Contactless (MRZ Fields are entered manually)

Reading the public data from EIDA card is protected by Basic Access Control (BAC). ReadPublicDataContactless generates MRZ input required for deriving the BAC keys. MRZ input is based on the CardNumber, DateOfBirth, and ExpiryDate that are accepted as parameters to this function.

Once BAC keys are diversified, Toolkit establishes secure messaging with the card to read the data files and then populates the CardHolderPublicDataEx Class with the card holder public data.

The text fields in this class are encoded in UTF8 format. If required, the conversion should be carried out to convert it to a proper encoding before use, date fields are represented in 4 bytes and it should be decoded properly. Refer to the example on how to decode it.

In order to optimise reading public data performance as reading data from the smart card is known to be slow, this function allows reading only specific sets of the data based on combination of the same set of flags used for the ReadPublicDataEx function.

A successful call of this function populates an instance of the class CardHolderPublicDataEx with the read public data fields and then these fields can be retrieved by calling member

function dedicated for each data filed in this class, the list of data fields in this class can be identified from the table in section 6.5.4

Note:

1. MW_ReadPublicDataContactless works only with contactless PC\SC readers.
2. MW_ReadPublicDataContactless works only with V2 Cards

Below examples provide the usage of both the functions in Java and C# respectively.

Java Example

```
import emiratesid.ae.publicdata.*;

// ... some code here...

try {
PublicDataFacade publicDataFacade = reader.getPublicDataFacade();

// user was prompted to enter MRZ fields Manually
char[] cn = // card number
char[] birthDate = // date of birth
char[] expiryDate = // card expiry date
CardHolderPublicDataEx publicData =
publicDataFacade.readPublicDataContactless(cn, birthDate, expiryDate,
true, true, true, true, true, true, true, true);

char[] fullNameBin = publicData.getFullName();
String fullName = Utils.CharArrayToUTF8String(fullNameBin);
char[] sexBin = publicData.getSex();
String sex = Utils.CharArrayToUTF8String(sexBin);
char[] issueDateBin = publicData.getIssueDate();
String issueDate = Utils.CharArrayToStringDate(issueDateBin);
char[] photography = publicData.getPhotography();

// use publicData.getXXX as needed
//...

} catch (MiddlewareException e) {
      // ... Handle exceptions here
}
```

C# Example

```
using EmiratesId.AE.PublicData;

try {
PublicDataFacade publicDataFacade = reader.GetPublicDataFacade();

// user was prompted to enter MRZ fields Manually
byte[] cn = // card number
byte[] birthDate = // date of birth
byte[] expiryDate = // card expiry date
CardHolderPublicDataEx publicData =
publicDataFacade.ReadPublicDataContactless(cn, birthDate, expiryDate,
true, true, true, true, true, true, true, true);
byte[] fullNameBin = publicData.GetFullName();
String fullName = Utils.CharArrayToUTF8String(fullNameBin);
byte[] sexBin = publicData.GetSex();
String sex = Utils.CharArrayToUTF8String(sexBin);
byte[] issueDateBin = publicData.GetIssueDate();
String issueDate = Utils.CharArrayToStringDate(issueDateBin);
byte[] photography = publicData.GetPhotography();

// use publicData.getXXX as needed
//…

} catch (MiddlewareException e) {
      // … Handle exceptions here
}
```

5.9 Read Public Data Contactless (with MRZ Reader)

Reading the public data from EIDA card is protected by Basic Access Control (BAC). ReadPublicDataContactlessWithMRZData expects the MRZ lines read by MRZ reader as an input that is used to diversify BAC keys.

Note: new line and cartridge return characters must be removed from the MRZ text returned from an MRZ reader before passing it to the function ReadPublicDataContactlessWithMRZData

Once BAC keys are diversified, Toolkit establishes secure messaging with the card to read the data files .then populate the CardHolderPublicDataEx Class with the card holder public data.

The text fields in this class are encoded in UTF8. If required, the conversion should be carried need to convert it to a proper encoding before use, date fields are represented in 4 bytes and it should be decoded. Refer to the example on how to decode it.

In order to optimise reading public data performance as reading data from the smart card is known to be slow, this function allows to read only specific sets of the data based on combination of the same set of flags used for the ReadPublicDataEx function.

A successful call to this function populates an instance of the class CardHolderPublicDataEx with the read public data fields and then these fields can be retrieved by calling member function dedicated for each data filed in this class, the list of data fields in this class can be identified from the table in section 6.5.4

Note:

1. MW_ReadPublicDataContactlessWithMRZData works only with contactless PC\SC readers.
2. MW_ReadPublicDataContactlessWithMRZData works only V2 Cards

Below examples provide the usage of both the functions in Java and C# respectively.

Java Example

```
import emiratesid.ae.publicdata.*;

// ... some code here...

try {
PublicDataFacade publicDataFacade = reader.getPublicDataFacade();

// reading with MRZ data
char[] mrzData = // read MRZ data here...
CardHolderPublicDataEx publicData = publicDataFacade.
readPublicDataContactlessWithMRZData(mrzData, true, true, true, true,
true, true, true, true);

char[] fullNameBin = publicData.getFullName();
String fullName = Utils.CharArrayToUTF8String(fullNameBin);
char[] sexBin = publicData.getSex();
String sex = Utils.CharArrayToUTF8String(sexBin);
char[] issueDateBin = publicData.getIssueDate();
String issueDate = Utils.CharArrayToStringDate(issueDateBin);
char[] photography = publicData.getPhotography();

// use publicData.getXXX as needed
//...

} catch (MiddlewareException e) {
      // ... Handle exceptions here
}
```

C# Example

```
using EmiratesId.AE.PublicData;

try {
PublicDataFacade publicDataFacade = reader.GetPublicDataFacade();

// reading with MRZ data
byte[] mrzData = // read MRZ data here...
CardHolderPublicDataEx publicData = publicDataFacade.
ReadPublicDataContactlessWithMRZData(mrzData, true, true, true, true,
true, true, true, true);
byte[] fullNameBin = publicData.GetFullName();
String fullName = Utils.CharArrayToUTF8String(fullNameBin);
byte[] sexBin = publicData.GetSex();
String sex = Utils.CharArrayToUTF8String(sexBin);
byte[] issueDateBin = publicData.GetIssueDate();
String issueDate = Utils.CharArrayToStringDate(issueDateBin);
byte[] photography = publicData.GetPhotography();

// use publicData.getXXX as needed
//…

} catch (MiddlewareException e) {
      // … Handle exceptions here
}
```

5.10 Reading Family Book Data

Using a connected PCSCReader object, application can read the Card holder family book data from the card. The retrieved information will be in binary format, and hence it should be converted to String format before using them. Refer to ‘Utils’ Class for sample conversion implementation.

In order to read family book data, Toolkit must be authenticated with the ID applet via dedicated cryptography sequence that requires access to EIDA secure messaging module from the card requires secure messaging

Read family book data function can be executed locally if EIDA secure messaging module (SAM smart card or HSM) attached to the PC locally. This function can also be executed with Software SAM in case test cards used. Please refer to Appendix A for more details on configuring EIDA secure messaging.

Family book container exists only on V2 cards hence executing this function with V1 cards will return an error.

In order to optimize the performance of reading family book data a refresh flag is used to read the already cached data (from memory).

- **Refresh flag:** set to true to read the data from the card and ,set to false to will read cached data

Java Example

```
import emiratesid.ae.publicdata.*;

try {
// make sure that UAEIDCardLibJavaWrapper.LoadConfiguration() is called
// before executing the following code, and sm.cfg is configured

FamilyBookDataFacade familyBookDataFacade =
reader.getFamilyBookDataFacade();
FamilyBookData familyBookData =
familyBookDataFacade.readFamilyBookData(true);

char[] firstNameBin =
familyBookData.getChild1().getChildFirstNameArabic();
String firstName = Utils.CharArrayToUTF8String(firstNameBin);
char[] sexBin = familyBookData.getChild1().getSex();
String sex = Utils.CharArrayToUTF8String(sexBin);
char[] DateOfBirthBin = publicData.getIssueDate();
String DateOfBirth = Utils.CharArrayToStringDate(issueDateBin);

// use familyBookData.getXXX as needed
//...

} catch (MiddlewareException e) {
      // ... Handle exceptions here
}
```

C# Example

```
using EmiratesId.AE.PublicData;

try {
// make sure that IDCardWrapper.LoadConfiguration() is called
// before executing the following code, and sm.cfg is configured

FamilyBookDataFacade FamilyBookDataFacade =
reader.GetFamilyBookDataFacade();
FamilyBookData FamilyBookData =
familyBookDataFacade.ReadFamilyBookData(true);

//…
byte[] firstNameBin =
familyBookData.getChild1().getChildFirstNameArabic();
String firstName = Utils.ByteArrayToUTF8String(firstNameBin);
byte[] sexBin = familyBookData.getChild1().getSex();
String sex = Utils.ByteArrayToUTF8String(sexBin);
byte[] DateOfBirthBin = publicData.getIssueDate();
String DateOfBirth = Utils.ByteArrayToStringDate(issueDateBin);

// use familyBookData.getXXX as needed
//...


} catch (MiddlewareException e) {
      // … Handle exceptions here
}
```

5.11 Checking Card Genuine

This function is used to verify that the presented ID card is authentic. Verifying an ID card is authentic and the card is issued by EIDA involves cryptographic challenge response sequence where the Toolkit sends a random challenge to the card and hence the card applies internal cryptographic operations and provides with response corresponds to the challenge. The Toolkit then verifies the card response and makes sure that the keys and cryptographic algorithm used by the card is genuine, and therefore the card is authentic issued by EIDA. In order to carry out this function EIDA secure messaging (SM) module.

5.11.1 Verifying Card Genuine in local mode

Check card genuine function can be executed locally if EIDA secure messaging module (SAM smart card or HSM) attached to the PC locally, Alternatively Check Card genuine function can be executed remotely using EIDA's remote Secure Messaging web service as explained in the following section.

This section provides the sample code on how to utilize the local SAM / HSM in order to check the card is genuine. Next section describes the remote execution of this function.

In order to invoke isCardGenuine function in the local mode, the sm.cfg configuration file shall be configured as specified in Appendix A according to the availability of secure messaging modules (HSM, SAM or multiple SAM, or Software SAM). Additionally this function requires a connected PCSCReader object in order to conduct the communication with the card.

Below examples, provide the usage of the functions in Java and C# respectively.

Java Example

```
import emiratesid.ae.publicdata.*;

// ... Some code here...

try {
// make sure that UAEIDCardLibJavaWrapper.LoadConfiguration() is called
// before executing the following code, and sm.cfg is configured
PublicDataFacade publicDataFacade = reader.getPublicDataFacade();
boolean isGenuine = publicDataFacade.isCardGenuine();

} catch (MiddlewareException e) {
      // ... Handle exceptions here...
}
```

C# Example

```
using EmiratesId.AE.PublicData;

// ... Some code here ...

try {
PublicDataFacade publicDataFacade = reader.GetPublicDataFacade();
bool isGenuine = publicDataFacade .IsCardGenuine();

} catch (MiddlewareException e) {
      // ... Handle exceptions here...
}
```

5.11.2 Verifying Card Genuine in remote mode

In case if EIDA secure messaging module is not available locally or in case of having EIDA secure messaging module in a central place, EIDA secure messaging web service can be used to avail the secure messaging remotely to multiple clients over HTTP/S protocol.

At client side the Toolkit offers a client implementation where consuming the secure messaging web service is done. In case of using secure messaging web service, the developer needs to use the secure messaging API (web service Client) for executing any cryptography based function with the card.

For using secure messaging API in java, the SecureMessagingAPI.jar must be added the build path or the classpath before using the following code.

The web service client is implemented in two classes called “JSONSecureMessagingClient” and “SecureMessagingActions”. In order to utilize the web service, both classes must be used as below:

1. an instance of **JSONSecureMessagingClient** class should be created by passing the URL of the secure messaging web service parameter to the class constructor:
2. an instance of **SecureMessagingAction** class should be created by passing Reader parameter to the class constructor. Reader is an i**nstance of the class P**CSCReader in a connected state.

Note: All the cryptography based functions exported by the SecureMessagingAction class are expects an instance of JSONSecureMessagingClient as a parameter.

Below examples, describe the usage of the IsCardGenuine in Java and C# respectively.

Java Example

```
import emiratesid.ae.sm.client.*;
// ... Toolkit Open context then Intialize PCSCReadercomes here

try {
// assuming that the web service URL is
// http://localhost:8080/RemoteSecureMessagingService/JsonSMServlet

JSONSecureMessagingClient client = new
JSONSecureMessagingClient
(http://localhost:8080/RemoteSecureMessagingService/JsonSMServlet);
SecureMessagingActions actions = new SecureMessagingActions (reader);

boolean isGenuine = actions.isCardGenuine(client);

} catch (MiddlewareException ex) {
      // ... Handle exceptions here...
}
```

C# Example

```
using EmiratesId.AE.SM.Client;


// ... Toolkit Open context then Intialize PCSCReadercomes here

try {
// assuming that the web service URL is
//http://localhost/RemoteSecureMessagingService/RemoteSecureMessagingSer
//vice.aspx

JSONSecureMessagingClient client = new
JSONSecureMessagingClient("http://localhost/RemoteSecureMessagingService
/RemoteSecureMessagingService.aspx");

         SecureMessagingActions actions = new
SecureMessagingActions(reader);
         bool CardIsGenuine = actions.IsCardGenuine(client);
} catch (MiddlewareException e) {
      // ... Handle exceptions here...
}
```

5.11.3 Verifying Card Genuine Extended

"IsCardGenuineEx" function extends IsCardGenuine validation utilising extended cryptography functions, it is recommended to use this function rather than 'IsCardGenuine'.

The way this function to be invoked is exactly the same way as IsCardGenuine in either local or remote modes, refer to the two previous sections for details.

5.12 Matching Off/On Card

After capturing a fingerprint and converting the captured image into the appropriate template, Toolkit provides two types of functions to match (compare) the captured fingerprint against the one retrieved from the EIDA ID Card.

Match-Off-Card

After reading (extracting) the fingerprint template stored on the card can be matched with the card holder fingerprint read from the cardholder using the fingerprint. This type of matching operation is called Match-Off-Card operation as the match operation is performed outside the card applets. This involves the below steps.

- Biometric Information Template read in order to decide which finger will be used for matching.
- The fingerprint of the owner is captured and converted to ISO template.
- Fingerprint template(s) retrieved from card
- Do Off-Card matching outside the card using the matcher function embedded in the Toolkit.

Match-On-Card

After capturing the fingerprint from the scanner, Toolkit will convert the captured fingerprint into the required template and invoke the MOC Applet located on the card. The MOC Applet will return either success or Failure. As the matching operation is done inside the card, it is called Match-On-Card (MOC). This involves the below steps.

- Biometric Information Template read in order to decide which finger will be used for matching.
- The cardholder fingerprint is captured and converted to DINV template.
- Send the captured template to the card.
- Do On-Card matching inside the card using the match on card Applet.

The following sections explain the steps specified above with code samples.

5.12.1 Reading Biometric Information Templates (BITs)

The BIT templates in the Card provide the indices of the fingerprints which are stored on the card and their reference data qualifiers. Therefore, this function should be called before the execution of a Match-on-card operation. Since there are typically two fingerprint templates stored on the card, Toolkit consists of two functions called getFirstBit and getSecondBit to return an instance of the BIT class containing the template information for each template. The following example shows how to retrieve and utilize the BITs objects and how to use the FingerIndexType class to identify finger indices and finger names. The BITs objects are used as parameter to the match on card function.

NOTE: An initialized PCSCReader object is required to read the biometrics information Templates.

Java Example

```
BiometricFacade biometricFacade = reader.getBiometricFacade();
try {
biometricFacade.readBiometricInfoTemplates();
BIT firstBit = biometricFacade.getFirstBit();
BIT secondBit = biometricFacade.getSecondBit();
int refDataQualifier = firstBit.getReferenceDataQualifier();
int fingerIndex = firstBit.getFingerIndex();

// recogizing finger name
if(fingerIndex == FingerIndexType.RIGHT_THUMB_TYPE.getIndex())
      return "Right Thumb";
} catch (MiddlewareException ex) {
      // ... Handle exceptions here
}
```

C# Example

```
BiometricFacade biometricFacade = reader.GetBiometricFacade();
try
{
      biometricFacade.ReadBiometricInfoTemplates();
      BIT firstBit = biometricFacade.FirstBit;
      BIT secondBit = biometricFacade.SecondBit;
      int refDataQualifier = firstBit.ReferenceDataQualifier;
      int fingerIndex = firstBit.FingerIndex;

// recogizing finger name
if (fingerIndex == FingerIndexType.RIGHT_THUMB_TYPE.Index)
       return "Right Thumb";
}
catch (MiddlewareException ex)
{
      // ... Handle exceptions here...
}
```

5.12.2 Capturing Fingerprints

This function captures the fingerprint image from the fingerprint sensor and checks for the quality of the captured image. If the quality of the image is good enough, then this function returns the captured image to the calling application.

As there are many fingerprint sensors in the market and each has different capturing quality, the Toolkit has been tested with following three sensors:

1. Sagem MSO 1350
2. Dermalog ZF1 and ZF1Plus
3. Futronic FS82

Currently the Toolkit provides Off-the-shelf integration with Sagem MSO 1350 sensor, so if this sensor is used then a developer can directly use an existing class called “SagemSensorDevice” which implements the fingerprint capturing functionality for that sensors, the below code gives an example of using Sagem MSO 1350 sensor.

Java Example

```
import emiratesid.ae.biometrics.*;

// ... Some code here ...

try {
      SensorDevice sensor = new SagemSensorDevice();
      // SensorDevice sensor = new DermalogSensorDevice
      FingerIndexType fingerIndex = FingerIndexType.LEFT_INDEX_TYPE;
      FTP_Image image = sensor.captureImage(fingerIndex);

      BiometricFacade bio = reader.getBiometricFacade();
      FTP_Template template = bio.convertImage(image,
      FormatType.ISO_19794_CS.getFormat());
} catch (MiddlewareException ex) {
      // ... Handle exceptions here...
}
```

C# Example

```
using EmiratesId.AE.Biometrics;

// ... Some code here ...

try {
      SensorDevice sensor = new SagemSensorDevice();
      // SensorDevice sensor = new DermalogSensorDevice
      FingerIndexType fingerIndex = FingerIndexType.LEFT_INDEX_TYPE;
      FTP_Image image = sensor.CaptureImage(fingerIndex);

      BiometricFacade bio = reader.GetBiometricFacade();
      FTP_Template template = bio.ConvertImage(image,
      FormatType.ISO_19794_CS.GetFormat());
} catch (MiddlewareException ex) {
      // ... Handle exceptions here...
}
```

The Toolkit design gives the flexibility to use other sensors as well through implementing standardised interface specified in appendix C.

Below are the steps a developer needs to do in order to use a sensor other than Sagem MSO 1350 :

1. Implement the interface specified in Appendix C as a C++ DLL
2. Change the configuration in the file “sensors.cfg” as specified in the same Appendix.

   Note: EIDA toolkit comes with sample interface implementations for two different sensors (Dermalog and Futronic sensors specified above)
3. Use the class SensorDevice defined in the toolkit API as below:

a. Create an instance of the SensorDevice class with passing the sensorId to the class constructor, the sensorId is the id corresponding to the desired sensor in the “sensors.cfg” file, if the value 0 is passed as sensorId then the toolkit will try to connect with any of the configured sensors, if all configured sensors fail to connect then toolkit try to connect to Sagem MSO 1350 sensor. If it fails then the capture function returns the error E_BIOMETRICS_NO_DEVICE.

b. Call the “CaptureImage” function, the capture function returns the fingerprint image in RAW format that will be passed later to the API function ConvertImage specified in section 5.7.3. to get the template required for the matching function.

Note: If the sensor has a build-in capabilities that allows direct conversion of the captured image to target template (ISO_19794_CS or DINV_66400) then the developer might implement the optional function “Capture_Convert” that returns a template in this case the developer shall call the function “CaptureAndConvert” defined in the class “SensorDevice” to get a template ready for the matching function.

The below code gives an example of using the SensorDevice class to capture an image from a dynamically configured fingerprint sensor.

Java Example

```
//Some Code Here
BiometricFacade biometricFacade = reader.getBiometricFacade();
// DeviceID could be one of the following values
//-1 to connect to Sagem MSO 1350 sensor
// 0 for Auto detect connected device
//1,2,… the sensor ID as specified in the Sensors.CFG file
sensor = new SensorDevice(pluginId);
ftpImage = sensor.captureImage(fingerIndex);
if(ftpImage == null || ftpImage.getPixels() == null)
return;
System.out.println("Image Captured with Quality : " +
ftpImage.getQuality());
```

C# Example :

```
//Some code here

BiometricFacade biometricFacade = reader.GetBiometricFacade();
// DeviceID could be one of the following values
//-1 to connect to Sagem MSO 1350 sensor
// 0 for Auto detect connected device
//1,2,… the sensor ID as specified in the Sensors.CFG file

sensor = new SensorDevice(DeviceID);

FtpImage = sensor.CaptureImage(fingerIndex);

if(rightFtpImage == null || rightFtpImage.Pixels == null)
  return;
Console.WriteLine(("Image Captured with Quality :"+
rightFtpImage.Quality);
```

5.12.3 Converting Fingerprint

The fingerprint matching function expects the minutiae data of the captured fingerprint while the biometric sensor returns either RAW or BMP images in case if the developer decided to use any of the other sensors specified in section 5.12.2.

Toolkit provides two convert functions to acquire an instance of the FTP_Template class that is expected by the matching function. They are convertImage and ConvertBmpImage. Developer can use any one of the functions based on their application requirement.

convertImage: function computes the fingerprint minutiae from the raw captured fingerprint image. Below sample code (both in Java and C#) provides the usage of this function.

Java Example

```
import emiratesid.ae.biometrics.*;

// ... capture fingerprint in image ...

try {
      BiometricFacade BC = new BiometricFacade();
      FormatType format Type = FormatType.DINV_66400;
      FTP_ Template template = BC.convertImage(image,
      formatType.getFormat());
} catch (MiddlewareException ex) {
      // ... Handle exceptions here...
}
```

C# Example

```
using EmiratesId.AE.Biometrics;

// ... capture fingerprint in image ...

try {
      BiometricFacade BC = new BiometricFacade();
      FormatType formatType = FormatType.DINV_66400;
      FTP_Template template = BC.ConvertImage(image, formatType.Format);
} catch (MiddlewareException ex) {
      // ... Handle exceptions here...
}
```

ConvertBmpImage: function computes the fingerprint minutiae from the BMP format of the fingerprint image captured using fingerprint sensor. Below sample code (Java and C#) provides the usage of this function.

Java Example

```
import emiratesid.ae.biometrics.*;

// ... capture fingerprint using separate component then return the BMP
image as byte array ...

try {
      BiometricFacade BC = new BiometricFacade();
      FormatType formatType = FormatType.DINV_66400;
      FTP_Template template = BC. ConvertBmpImage(BMPByteArray,
      formatType.Format);
} catch (MiddlewareException ex) {
      // ... Handle exceptions here...
}
```

C# Example

```
using EmiratesId.AE.Biometrics;

// ... capture fingerprint using separate component then return the BMP
image as byte array ...

try {
      BiometricFacade BC = new BiometricFacade()
      FormatType formatType = FormatType.DINV_66400;
      FTP_Template template = BC. ConvertBmpImage(BMPByteArray,
      formatType.Format);
} catch (MiddlewareException ex) {
      // ... Handle exceptions here...
}
```

5.12.3.1 Reading Fingerprint templates from Card (Match-Off only)

Reading fingerprint template is required only in case of Match-off-Card matching since the matching process will be conducted outside the card. To read the fingerprint template from the ID applet (that is where the fingerprints are stored in the ID card), Toolkit must be authenticated with the ID applet via dedicated cryptography sequence that requires access to EIDA secure messaging module. There are two ways to access the security module, one local and the other way is remotely. Refer to the below subsections below which provides further details.

5.12.3.2 Using EIDA SM module in local mode

If EIDA Secure messaging module is available locally then SM.CFG file need to be configures as specified in Appendix A and hence cardholder fingerprints can be read from the card using the “readFingerprints” function of the “BiometricFacade”. The code below shows a simple use of that function:

NOTE: An initialized PCSCReader object is required to read the fingerprint templates.

Java Example

```
//Establish secure messaging
BiometricFacade biometricFacade = reader.getBiometricFacade();
try {
biometricFacade.readFingerprints();
} catch (MiddlewareException ex) {
      // ... Handle exceptions here
}
```

C# Example

```
//Establish secure messaging
BiometricFacade biometricFacade = reader.GetBiometricFacade();
try
{ biometricFacade.readFingerprints();}
catch (MiddlewareException ex)
{
      // ... Handle exceptions here...
}
```

5.12.3.3 Using EIDA SM module in remote mode

In case if EIDA secure messaging module is not available locally or in case of having EIDA secure messaging module in a central place, EIDA secure messaging web service can be use to avail the secure messaging remotely to multiple clients over HTTP\S protocol.

The Toolkit does not allow reading fingerprints separately in remote mode, so a developer only need to call the match-off-card function after capture\convert fingerprint.

Calling the match-off-card function in remote mode requires Toolkit web service client to be initialized as specified in section 5.11.2 and hence call the “MatchOffCard” function of the “SecureMessagingActions” class with the following parameters:

smm: an instance of the class SecureMessagingInterface instantiated as specified in section 5.11.2.

Template: an instance of the class FTP_Template instantiated as specified in section 5.12.3.

The function returns empty string if the matching successful otherwise it through an exception

The code below shows a simple use of that function:

Java Example

```
//Intialize tookit\reader

// capture the card holder fingerprint then convert it to a template
// assuming that the web service URL is//
//"http://localhost:8080/RemoteSecureMessagingService/JsonSMServlet"
JSONSecureMessagingClient client = new
JSONSecureMessagingClient("http://localhost:8080/RemoteSecureMessagingSe
rvice/JsonSMServlet");
SecureMessagingActions actions = new SecureMessagingActions(reader);
String response = actions.matchOffCard(client, template);
```

C# Example

```
//Intialize tookit\reader

// capture the card holder fingerprint then convert it to a template
// assuming that the web service URL
is//http://localhost/RemoteSecureMessagingService/RemoteSecureMessagingS
ervice.aspx

JSONSecureMessagingClient client = new
JSONSecureMessagingClient("http://localhost/RemoteSecureMessagingService
/RemoteSecureMessagingService.aspx");

SecureMessagingActions actions = new SecureMessagingActions(reader);
String response=actions.MatchOffCard(client,template);
```

5.12.4 Off Card matching

matchOffCard function can be invoked locally or remotely based on the implementation requirement. This section describes the local mode (where EIDA secure messaging module is available locally). Refer to section 5.12.3 above for further details about this function.

After capturing a fingerprint and converting the captured image into the appropriate minutiae template, the acquired template is sent as a parameter to the function matchOffCard of the class BiometricFacade in order to perform the cardholder verification.

This function expects a template parameter which is an instance of the class FTP_Template instantiated as specified in section 5.12.3. This function returns the matching score in numerical format. Note that if the returned value is greater than 13000 means templates are matching else it is not.

NOTE: The fingerprint index of the acquired template must be one of the indices stored on the card, otherwise no successful matching can occur against one of the card templates.

Java Example

```
BiometricFacade bc = reader.getBiometricFacade();
// capture and convert a fingerprint into ISO Template
// ...
bc.readFingerprints();

int result = bc.match(bc.getFirstTemplate(), capturedTemplate);
```

C# Example

```
BiometricFacade BC = Reader.getBiometricFacade();
// capture and convert a fingerprint into ISO Template
// ...
BC.ReadFingerprints();

int result = BC.Match(BC.GetFirstTemplate(), capturedTemplate);
```

5.12.5 On Card matching

Before conducting the OnCard matching, the Toolkit must be authenticated with the MOC applet via dedicated cryptography sequence that requires access to EIDA secure messaging module locally where the SM.CFG file need to be configures as specified in Appendix A.

the following parameters should be sent to the function matchOnCard of the class BiometricFacade.

bit: an instance of BIT class where desired on card template is referenced. The BIT object should be retrieved from the card using function readBiometricInfoTemplates, please refer to section 5.12.1 above.

template: an instance of the FTP_Template class that was retrieved from the convert function, please refer to section 5.12.3.

This function returns true if it matches else it returns false.

NOTE: The fingerprint index of the acquired template must be one of the indices stored on the card, otherwise no successful matching can occur against one of the card templates.

Java Example

```
// Match On Card
// capture and convert a fingerprint into DINV Template
// ...

boolean result = biometricFacade.matchOnCard(firstBit, ftpTemplate);
```

C# Example

```
// Match On Card
// capture and convert a fingerprint into DINV Template
// ...

boolean result = biometricFacade.MatchOnCard(firstBit, ftpTemplate);
```

5.13 Signing Data with the Authentication Key

The EIDA card PKI applet is personalised with two digital certificates, one for authentication and another for signing. Signing data with authentication key is used for implementing two-factor authentication (where user knows the PIN and have the card). Toolkit provides two ways of acquiring PIN from the cardholder during authentication. They are **Authenticate** and **Authenticate_PinCached**.

Authenticate function is suitable for a scenario where the cardholder is required to enter his PIN code every time. This function will not cache the PIN. The function expects the following parameters:

szPin: string contains the card PIN acquired from the card holder

data: byte array contains the data (usually the challenge) to be signed

Authenticate_PinCached function is typically used in scenario where the cardholder is prompted for his PIN code only the first time and the function will cache the PIN and reuse it when required. The function expects the following parameter:

data: byte array contains the data (usually the challenge) to be signed

Sample codes of using the 2 authentication functions are presented below.

Authenticating cardholder without PIN Caching

Java Example

```
String userPIN = "1234"; // the user to input his card pin
PKIFacade pkiFacade = reader.getPKIFacade();
byte[] signature = pkiFacade.authenticate(userPIN, "<challenge>");
```

C# Example

```
String userPIN = "1234"; // the user to input his card pin
PKIFacade pkiFacade = reader.GetPKIFacade();
byte[] signature = pkiFacade.Authenticate(userPIN, "<challenge>");
```

Authenticating a cardholder with PIN caching

Java Example

```
PKIFacade pkiFacade = reader.getPKIFacade();
// the PIN is cached
byte[] signature = pkiFacade. Authenticate_PinCached ("<challenge>");
```

C# Example

```
PKIFacade pkiFacade = reader.getPKIFacade();
// the PIN is cached
byte[] signature = pkiFacade. Authenticate_PinCached ("<challenge>");
```

5.14 Signing Data with the Signing Key

Signing data with signing key function is used for signing data or document. Toolkit exports the function SignData from the class PKIFacade. This function expects the below parameters:

szPin: string contains the card PIN acquired from the card holder

data: byte array contains the data to be signed

Below sample code provides the usage of the function.

Java Example

```
String userPIN = "1234"; // the user to input his card pin
PKIFacade pkiFacade = reader.getPKIFacade();
byte[] signature = pkiFacade.signData(pin, "data to sign");
```

C# Example

```
string userPIN = "1234"; // the user to input his card pin
PKIFacade pkiFacade = reader.GetPKIFacade();
byte[] signature = pkiFacade.SignData(pin, "data to sign");
```

5.15 Reading PKI certificates

The PKI applet in the ID card is personalised with two digital certificates, one for authentication and another for signing. ExportCertificates function of class PKIFacade reads the certificates from card and then caches it. Developer can the call GetAuthCertificate and GetSignCertificate functions to retrieve Authentication and Signing certificates respectively.

Below sample code provides the usage of both functions.

Java Example

```
PKIFacade pkiFacade = reader.getPKIFacade();
pkiFacade. ExportCertificates();

Certificate authCertificate = pkiFacade. GetAuthCertificate ();
Certificate signCertificate = pkiFacade. GetSignCertificate ();
```

C# Example

```
PKIFacade pkiFacade = reader.GetPKIFacade();
pkiFacade. ExportCertificates();
X509Certificate authCertificate = pkiFacade. GetAuthCertificate ();
X509Certificate signCertificate = pkiFacade. GetSignCertificate ();
```

5.16 PIN management functions

EIDA ID Card Toolkit provides 3 PKI applet PIN management functions These 3 functions are exported in the PKIFacade class. This section describes all the 3 functions in detail.

5.16.1 Resetting PIN

This function unblocks the PKI PIN and sets the newly passed PIN. Before calling this **function, the Toolkit should be authenticated with card and then the new PIN is transmitted** to the card over SM module.

The PIN reset function requires access to SM module locally therefore, the SM.CFG file need to be configured as specified in Appendix A and hence the new PIN should be sent to the function ResetPIN of the class PKIFacade.

If PIN reset is successful then the function will return 0.

Java Example

```
PKIFacade pkiFacade = reader.getPKIFacade();
pkiFacade. ResetPin(NewPIN /*string contains the new PIN*/);
```

C# Example

```
PKIFacade pkiFacade = reader.getPKIFacade();
pkiFacade. ResetPin(NewPIN /*string contains the new PIN*/);
```

5.16.2 Changing PIN

This functions allows a card holder to change his old PIN, to do so, the Toolkit exports the function ChangePin of the class pkiFacade that expects the following parameters:

oldPIN: string contains the old PIN

newPIN: string contains the new PIN

If PIN reset is successful then the function will return 0

Java Example

```
PKIFacade pkiFacade = reader.getPKIFacade();
pkiFacade.ChangePin (oldPIN,newPIN);
```

C# Example

```
PKIFacade pkiFacade = reader.getPKIFacade();
pkiFacade.ChangePin (oldPIN,newPIN);
```

5.17 MIFARE Emulation

EIDA V2 cards have a Mifare emulation application, the application works exactly as Mifare 1K Classic chip with same memory segmentation and security conditions.

Toolkit provides a set of APIs that exposes Mifare application phase functions such as (Load Key, authenticate, Read Binary and Write Binary).

By default, Mifare emulation is enabled if the card has a standard contactless interface which is de-facto standard. Due to this reason, any contactless reader will establish connection with the contactless interface when the card is in the range. There will be a separate reader command to switch the connection to specific Mifare emulation application. The switch command is proprietary to each reader and it will require specific implementation for each reader. In order to overcome this challenge, a dynamic framework has been implemented in the Toolkit which allows the switch command in plugin architecture and integrates with the Toolkit at runtime using a configuration file. Please refer appendix D of the Java/.NET developers guide for configuration and plug-in implementation guidance.

Notes:

- Currently Toolkit comes with a sample plug-in that implements the Switch to Mifare Emulation command for HID OMNIKEY 5321 reader.
- The Toolkit implements the rest of Mifare commands (Load Key, Autheticate, Read Binary and Write Binary) according to PC/SC standard therefore the toolkit supports only readers complies with PC/SC Mifare commands.

5.17.1 Switch to MIFARE emulation

This API switches the reader connection to Mifare Emulation application on the card. The actual implementation of this API is based on the plugin for the specific reader as mentioned in the above paragraph.

This API is exposed through the **PCSCReader** class which represents the reader; if it is successful it returns an object of type **MifareFacade** that exposes all Mifare functions supported by the Toolkit.

SwitchToMifareEmulation API takes only one parameter which can be either Reader ID or 0. The reader ID value must be configured in mifare.cfg file (please refer to Appendix D). If the value is 0, then the Toolkit to will use the first available configured reader according the sequence where readers are configured in mifare.cfg.

Once the reader is switched to MIFARE it stays in that mode even with a newly established context or a new card is connected. Developers can use another API called" isMifareEmulationActive" to check whether the connected reader is already switched to Mifare emulation or not.

Java Example

```
MifareFacade mifareFacade = reader.switchToMifareEmulation(0);
if(mifareFacade == null) {
      // Switch failed ...
}

boolean mifareActive = reader.isMifareEmulationActive();
```

C# Example

```
MifareFacade mifareFacade = reader. SwitchToMifareEmulation(0);
if(mifareFacade == null) {
      // Switch failed ...
}

bool mifareActive = reader.IsMifareEmulationActive();
```

5.17.2 Is Mifare Emulation Active

This API checks if the a specific reader is already switched to Mifare Emulation. This function returns true if the reader is in Mifare emulation mode otherwise it returns false. If it returns true, then Mifare functions can be used directly otherwise switchToMifareEmulation must be executed first.

Please refer to the examples came in the previous section.

5.17.3 Load Key

Mifare Read\Write operations mandates authentication with secret key. There are two types of keys supported by Mifare 1k Classic, they are Type A and Type B.

The calling application must load the key corresponding to the desired data block where the read\write operations will take place. Please contact EIDA for Mifare Application authentication keys.

The LoadKey API is exposed in the MifareFacade class and it loads the key to the reader memory via a standard PC/SC command.

Java Example

```
// keyNum = 0x00; // value between 0 and 31
// keyBin = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00}; // 6 bytes key
try {
	mifareFacade.loadKey(keyNum, keyBin);
} catch (MiddlewareException e) {
	// Error loading key
}
```

C# Example

```
// keyNum = 0x00; // value between 0 and 31
// keyBin = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00}; // 6 bytes key
try {
	mifareFacade.LoadKey(keyNum, keyBin);
} catch (MiddlewareException e) {
	// Error loading key
}
```

5.17.4 Read Binary Data

This API allows reading the 16 bytes binary data from any Mifare data blocks, and it is exposed in MifareFacade class.

This API can be used only of the key is already loaded on the reader and it Expects the following parameters:

Key Number: is a value between 0 and 31

Key Type:

Type A : with the value 0x60 (Hexadecimal)
Type B : with the value 0x61

Block Number: is a value between 0 and 63 (for MIFARE 1K)

If successful, the API returns the 16 bytes read from the desired data block

Java Example

```
byte keyNum = 0;
byte blockNum = 42;
byte keyType = (byte) 0x60; // Key A
try {
	char[] dataBin = mifareFacade.readBinary(blockNum, keyNum, keyType);
	System.out.println(Utils.CharArrayToHex(dataBin));
} catch (MiddlewareException ex) {
	// Error Reading Mifare
}
```

C# Example

```
byte keyNum = 0;
byte blockNum = 42;
byte keyType = (byte) 0x60; // Key A
try {
      byte[] dataBin = mifareFacade.ReadBinary(blockNum, keyNum, keyType);
            Console.WriteLine(Utils.ByteArrayToHex(dataBin));
} catch (MiddlewareException ex) {
      // Error Reading Mifare
}
```

5.17.5 Update Binary Data

This API allows Updating the 16 bytes binary data to any Mifare data blocks, and it is exposed in MifareFacade class.

This API can be used only of the key is already loaded on the reader and it Expects the following parameters:

Key Number: is a value between 0 and 31

Key Type:

Type A: with the value 0x60 (Hexadecimal)
Type B : with the value 0x61

Block Number: is a value between 0 and 63 (for MIFARE 1K)

Data: 16 Bytes Data to be written on the target data block

Java Example

```
byte keyNum = 0;
byte blockNum = 42;
byte keyType = (byte) 0x60; // Key A
char[] dataBin = new char[] {0x00, 0x01, 0x1F, 0xFF, 0xA0, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
try {
      mifareFacade.updateBinary(blockNum, keyNum, keyType, dataBin);
} catch (MiddlewareException ex) {
      // Error updating Mifare data block
}
```

C# Example

```
byte keyNum = 0;
byte blockNum = 42;
byte keyType = (byte) 0x60; // Key A
byte[] dataBin = new byte[]{0x00, 0x01, 0x1F, 0xFF, 0xA0, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
try {
      mifareFacade.UpdateBinary(blockNum, keyNum, keyType, dataBin);
} catch (MiddlewareException ex) {
      // Error Reading Mifare
}
```

5.18 Additional Containers

EIDA Card has dedicated data containers for some business sectors which allow business / government entities to manage their business related data (under an agreement with EIDA) on their corresponding data containers. Toolkit API provides access to the below containers:

- Labour
- Health and insurance
- Social Services
- Defence
- Driving License
- Qualifications

The data containers are managed by the ID Applet.Toolkit allows Updating\Reading binary data on a container by creating data file(s) under this the container's dedicated file where the binary data will be stored. However optionally it is required to restrict the access to those data files using secret key (secure messaging) or an application PIN.

Note: the secured messaging implemented by EIDA card for the additional containers doesn't support data encryption so when data read \ write are executed over secure messaging a propitiatory MAC is calculated over the data in order to insure data integrity however the data is sent in plain text except when a key or PIN are being written on the card, in this case the key\PIN are encrypted.

The toolkit offers the necessary the container management APIs required for creating secret key, application PIN and data file in order to achieve secure read\update binary data on the data containers managed by the ID applet.

The sequence of calling the container management APIs are specified below:

1. Create Secret Key on the card using MW_[*]CreateKey

2. Create an Application PIN on the card using MW_[*]CreatePIN
3. Create a Data file on the card using MW_[*]CreateDataFile
4. Freeze the access conditions to files using MW_[*]AccessConditions
5. Update Binary Data on the created data file using MW_[*]UpdateBinary
6. Read Binary Data from the created data file using MW_[*]ReadBinary

[*] is the container name, Ex. Labour, Defence, Health, etc…

The below subsections specifies the APIs offered by the toolkit for each data container

5.18.1 Labour

The APIs managing the labour container are exposed by the class **LabourContainer** which is defined in the **emiratesid.ae.containers** package (JAVA) \ namespace (.NET).

5.18.1.1 Create Key

EIDA ID card stores the keys in a dedicated key file so the "**createKey**" API first check if the input key file ID corresponds to an existing key file on the card then it creates a key file under the Labour container (if the key file is not already created) and finally it writes the input key value (3 DES -16 bytes) in the specified key index (Key Number).

Note: after all the keys are initialized the read and update on the key file(s) should be locked using FreezeAccessCondtions.

Please refer to section 5.55 for the context in which `CreateKey` shall be called.

Parameters

`byte KeyFileID` [in] identifier of the key file, KeyFileID value should be greater then or equal 0x00 and less then or equal 0x1F skipping the values 0x10 and 0x01 as it is reserved, File IDs must be unique per each container allover the files types .

`byte KeyNumber` [in] identifier of the key to be created on the key file, KeyNumber value should be less then 0x04 so maximum 4 keys are created per each key file.

`byte [] KeyValue` [in] 16 byte array key value to be written.

Java Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      LabourContainer labour = new LabourContainer(reader);
      labour.createKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      LabourContainer labour = new LabourContainer(reader);
      labour.CreateKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.1.2 Create PIN

This API creates a PIN file under Labour application (if it's not already created) and updates it with the input byte array PIN value to initialize a PIN code used for Pin verification. PIN number on the file is specified in the input parameters.

EIDA ID card stores the application PIN in a dedicated PIN file, each containers can contain only one PIN file, the "createPin" API first check if the PIN file is created then it creates a PIN file under the Labour container (if the PIN file is not already created) and finally it writes the input PIN value in the specified PIN index (PIN Number).

Note:

- Since only one PIN file can be created under the Health and Insurance container a fixed file identifier ; 0x10 is dedicated to the PIN File.
- After all the PINs are initialized the read and update on the Pin files should be locked using FreezeAccessCondtions.

Please refer to section 5.18 for the context in which "**createPin**" shall be called.

Parameters

`byte PinNumber`	[in] identifier of the PIN to be created on the application, PinNumber values should be less then 0x08 (the PIN file application can store maximum number of 8 PIN codes)

`byte []PinValue`	[in] 8 bytes PIN value byte array

Java Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      DefenseContainer defense = new DefenseContainer(reader);
      defense.createPin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      DefenseContainer defense = new DefenseContainer(reader);
      defense.CreatePin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.1.3 Create Data File

This API creates a Data file under Labour application with the identifier specified in FileID parameter with the access conditions specified in ReadAC and UpdateAC which are instances of the FAC structure specified below, the file size will be set to the size specified in the FileSize parameter.

Please refer to section 5.18 for the context in which `createDataFile` shall be called.

Parameters

`Byte FileID`	[in] identifier of the file to be created, FileID should be greater than 0x00 and less than or equal 0x FF skipping 0x01 and 0x10 since it reserved values. File IDs must be unique per each container allover the files types .
`FAC ReadAC`	[in] a pointer to file read access condition that is an isntance of the FAC structure defined below.
`FAC UpdateAC`	[in] a pointer to file update access condition that is an isntance of the FAC structure defined below.

FAC structure

Access conditions are specified using the class **FAC** in the package/namespace **emiratesid.ae.containers** FAC contains the following members :

byte ProtectionLevel : specifies the protection level of the access condition
0 : No Protection(Secure Messaging might be configured if Key file is not 0)
1 : Protected by PIN

byte KeyFile : key file stores the key(s) used for secure messaging (0 if secure messaging is not required)

byte PinNumber: PIN number that will be used to protect the file, please note that if a PIN number is sepecified then a KeyFile must be specified be cause the PIN is always sent to the card in ciphered mode using any key in the specfied key file.

To create a data file for a specific container, initialize an object from that container class with a valid reader object and then call **CreateDataFile()** method

Java Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
      DefenseContainer defense = new DefenseContainer (reader);
      defense.createDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
      DefenseContainer defense = new DefenseContainer (reader);
      defense.CreateDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.1.4 Freeze Access Conditions

This API does the following operations on any elementary file (Key file, PIN file or Data file):

- Freeze the Access condition so that I can't be changed anymore
- Lock a specific access (Read\Update) if required so that no more Read\Update operation is allowed any more on the file

Please refer to section 5.18 for the context in which freezeAccessConditions shall be called.

Note: once the access condition is locked it can't be unlocked, use this API at the end of the personalization phase to switch to the application phase.

Parameters

`byte FileID`	[in] identifier of the file which access condition will be locked, use 0x10 as FileID to lock access condition on the PIN File
`bool lockUpdate`	[in]if this flag is true update will be locked on the selected file
`bool readUpdate`	[in] if this flag is true read will be locked on the selected file

Java Example

```
byte fileID = 0x05;
try {
      DefenseContainer defense = new DefenseContainer (reader);
      defense.freezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
      DefenseContainer defense = new DefenseContainer (reader);
defense.FreezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.1.5 Update Binary

This API updates binary data on a data file specified in FileID parameter. Depending on the data file access conditions Key and PIN references and values might be needed in the input parameters to establish secure messaging or verify PIN before updating data file.

Please refer to section 5.18 for the context in which updateData shall be called.

Parameters

`byte FileID`	[in] identifier of the file to be updated
`byte PinNumber`	[in] identifier of the PIN used for pin verification (if needed)

`byte[] PinValue` [in] 8 bytes array holding the value of the pin code .this value is NULL if PIN verification is not needed

`byte KeyFileID` [in] identifier of the key file of the key used to establish secure messaging (if needed)

`byte KeyNumber` [in] identifier of the key used to establish secure messaging (if needed)

`byte] KeyValue` [in] 16 bytes array holding the key value used to establish secure messaging. This value is NULL if secure messaging is not needed

`int offset` [in] offset from which the update data is written on the file

`byte[] UpdateData` [in] byte array holding update data to be written on the file

Java Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
      DefenseContainer defense = new DefenseContainer (reader);
      defense.updateData(fileID, pinNum, pinVal, keyFileID, keyNum, keyVal,
offset, data);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
      DefenseContainer defense = new DefenseContainer (reader);
defense.UpdateData(fileID, pinNum, pinVal, keyFileID, keyNum, keyVal,
offset, data);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.1.6 Read Binary

This API reads the binary data from data file specified by FileID parameter. Depending on the data file access conditions Key and PIN references and values might be needed to establish secure messaging or verify PIN before reading data file.

Please refer to section 5.18 for the context in which readData shall be called..

Parameters

`byte FileID` [in] identifier of the file to be read

`byte PinNumber`	[in] identifier of the PIN used for pin verification (if needed)
`byte[] PinValue`	[in] 8 bytes array holding the value of the pin code (if needed).this value is NULL if PIN verification is not needed
`byte KeyFileID`	[in] identifier of the key file of the key used to establish secure messaging (if needed)
`byte KeyNumber`	[in] identifier of the key used to establish secure messaging (if needed)
`byte KeyValue`	[in] 16 bytes array holding the key value used to establish secure messaging (if needed). This value is NULL if secure messaging is not needed
int offset	[in] offset from which data is read from the file
int size	[in] number of bytes to be read from the file
byte[] Data	[out] byte array holding data retrieved from the file

Java Example

```
byte fileID = 0x05;
try {
	DefenseContainer defense = new DefenseContainer (reader);
	byte[] data = defense.readData(fileID, pinNum, pinVal, keyFileID,
keyNum, keyVal, offset, readSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
	DefenseContainer defense = new DefenseContainer (reader);
byte[] data = defense.ReadData(fileID, pinNum, pinVal, keyFileID, keyNum,
keyVal, offset, readSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.2 Health and Insurance

The APIs managing the Health and Insurance container are exposed by the class **HealthInsuranceContainer** that is defined in the **emiratesid.ae.containers** package (JAVA) \ namespace (.NET)

5.18.2.1 Create Key

EIDA ID card stores the keys in a dedicated key file so the "**createKey**" API first check if the input key file ID corresponds to an existing key file on the card then it creates a key file under the Health and Insurance container (if the key file is not already created) and finally it writes the input key value (3 DES -16 bytes) in the specified key index (Key Number).

Note: after all the keys are initialized the read and update on the key file(s) should be locked using FreezeAccessCondtions.

Please refer to section 5.55 for the context in which `CreateKey` shall be called.

Parameters

`byte KeyFileID`	[in] identifier of the key file, KeyFileID value should be greater then or equal 0x00 and less then or equal 0x1F skipping the values 0x10 and 0x01 as it is reserved, File IDs must be unique per each container allover the files types .
`byte KeyNumber`	[in] identifier of the key to be created on the key file, KeyNumber value should be less then 0x04 so maximum 4 keys are created per each key file.
`byte [] KeyValue`	[in] 16 byte array key value to be written.

Java Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      HealthInsuranceContainer Health = new
HealthInsuranceContainer(reader);
      Health.createKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      HealthInsuranceContainer Health = new
HealthInsuranceContainer(reader);
      Health.CreateKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.2.2 Create PIN

This API creates a PIN file under Health and Insurance application (if it's not already created) and update it with the input byte array PIN value to initialize a PIN code used for Pin verification. PIN number on the file is specified in the input parameters.

EIDA ID card stores the application PIN in a dedicated PIN file, each containers can contain only one PIN file, the "createPin" API first check if the PIN file is created then it creates a PIN file under the Health and Insurance container (if the PIN file is not already created) and finally it writes the input PIN value in the specified PIN index (PIN Number).

Note:

- Since only one PIN file can be created under the Health and Insurance container a fixed file identifier ; 0x10 is dedicated to the PIN File.
- After all the PINs are initialized the read and update on the Pin files should be locked using FreezeAccessCondtions.

Please refer to section 5.18 for the context in which "**createPin**" shall be called.

Parameters

`byte PinNumber`	[in] identifier of the PIN to be created on the application, PinNumber values should be less then 0x08 (the PIN file application can store maximum number of 8 PIN codes)
`byte []PinValue`	[in] 8 bytes PIN value byte array

Java Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      HealthInsuranceContainer Health = new
HealthInsuranceContainer(reader);
      Health.createPin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      HealthInsuranceContainer Health = new
HealthInsuranceContainer(reader);
      Health.CreatePin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.2.3 Create Data File

This API creates a Data file under Health and Insurance application with the identifier specified in FileID parameter with the access conditions specified in ReadAC and UpdateAC which are instances of the FAC structure specified below, the file size will be set to the size specified in the FileSize parameter.

Please refer to section 5.18 for the context in which `createDataFile` shall be called..

Parameters

`Byte FileID`	[in] identifier of the file to be created, FileID should be greater than 0x00 and less than or equal 0x FF skipping 0x01 and 0x10 since it reserved values. File IDs must be unique per each container allover the files types .
`FAC ReadAC`	[in] a pointer to file read access condition that is an isntance of the FAC structure defined below
`FAC UpdateAC`	[in] a pointer to file update access condition that is an isntance of the FAC structure defined below

FAC structure

Access conditions are specified using the class **FAC** in the package/namespace **emiratesid.ae.containers** FAC contains the following members :

byte ProtectionLevel : specifies the protection level of the access condition
0 : No Protection(Secure Messaging might be configured if Key file is not 0)
1 : Protected by PIN

byte KeyFile : key file stores the key(s) used for secure messaging (0 if secure messaging is not required)

byte PinNumber: PIN number that will be used to protect the file, please note that if a PIN number is sepecified then a KeyFile must be specified be cause the PIN is always sent to the card in ciphered mode using any key in the specfied key file.

To create a data file for a specific container, initialize an object from that container class with a valid reader object and then call **CreateDataFile()** method

Java Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
	HealthInsuranceContainer Health = new
HealthInsuranceContainer(reader);
	Health.createDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
	// Error occurred
```

C# Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
	HealthInsuranceContainer Health= new
HealthInsuranceContainer(reader);
	Health.CreateDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.2.4 Freeze Access Conditions

This API does the following operations on any elementary file (Key file, PIN file or Data file):

- Freeze the Access condition so that I can't be changed anymore
- Lock a specific access (Read\Update) if required so that no more Read\Update operation is allowed any more on the file

Please refer to section 5.18 for the context in which freezeAccessConditions shall be called.

Note: once the access condition is locked it can't be unlocked, use this API at the end of the personalization phase to switch to the application phase

Parameters

`byte FileID` [in] identifier of the file which access condition will be locked, use 0x10 as FileID to lock access condition on the PIN File

`bool lockUpdate` [in]if this flag is true update will be locked on the selected file

`bool readUpdate` [in] if this flag is true read will be locked on the selected file

Java Example

```
byte fileID = 0x05;
try {
	HealthInsuranceContainer Health= new
HealthInsuranceContainer(reader);
	Health.freezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
	// Error occurred
```

C# Example

```
byte fileID = 0x05;
try {
	HealthInsuranceContainer Health = new HealthInsuranceContainer
(reader);
Health.FreezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.2.5 Update Binary

This API updates binary data on a data file specified in FileID parameter. Depending on the data file access conditions Key and PIN references and values might be needed in the input parameters to establish secure messaging or verify PIN before updating data file.

Please refer to section 5.18 for the context in which updateData shall be called.

Parameters

`byte FileID`	[in] identifier of the file to be updated
`byte PinNumber`	[in] identifier of the PIN used for pin verification (if needed)
`byte[] PinValue`	[in] 8 bytes array holding the value of the pin code .this value is NULL if PIN verification is not needed
`byte KeyFileID`	[in] identifier of the key file of the key used to establish secure messaging (if needed)
`byte KeyNumber`	[in] identifier of the key used to establish secure messaging (if needed)
`byte] KeyValue`	[in] 16 bytes array holding the key value used to establish secure messaging. This value is NULL if secure messaging is not needed
`int offset`	[in] offset from which the update data is written on the file

`byte[] UpdateData` [in] byte array holding update data to be written on the file

Java Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
      HealthInsuranceContainer Health = new
HealthInsuranceContainer(reader);
      Health.updateData(fileID, pinNum, pinVal, keyFileID, keyNum, keyVal,
offset, data);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
      HealthInsuranceContainer Health= new
HealthInsuranceContainer(reader);
Health.UpdateData(fileID, pinNum, pinVal, keyFileID, keyNum, keyVal,
offset, data);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.2.6 Read Binary

This API reads the binary data from data file specified by FileID parameter.depending on the data file access conditions Key and PIN references and values might be needed to establish secure messaging or verify PIN before reading data file.

Please refer to section 5.18 for the context in which readData shall be called.

Parameters

`byte FileID` [in] identifier of the file to be read

`byte PinNumber` [in] identifier of the PIN used for pin verification (if needed)

`byte[] PinValue` [in] 8 bytes array holding the value of the pin code (if needed).this value is NULL if PIN verification is not needed

`byte KeyFileID` [in] identifier of the key file of the key used to establish secure messaging (if needed)

byte KeyNumber [in] identifier of the key used to establish secure messaging (if needed)

byte KeyValue	[in] 16 bytes array holding the key value used to establish secure messaging (if needed). This value is NULL if secure messaging is not needed
int offset	[in] offset from which data is read from the file
int size	[in] number of bytes to be read from the file
byte[] Data	[out] byte array holding data retreived from the file

Java Example

```
byte fileID = 0x05;
try {
	HealthInsuranceContainer Health= new HealthInsuranceContainer
(reader);
	byte[] data = Health.readData(fileID, pinNum, pinVal, keyFileID,
keyNum, keyVal, offset, readSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
	HealthInsuranceContainer Health= new HealthInsuranceContainer
(reader);
byte[] data = Health.ReadData(fileID, pinNum, pinVal, keyFileID, keyNum,
keyVal, offset, readSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.3 Defence

The APIs managing the Defence container are exposed by the class **DefenseContainer** that is defined in the **emiratesid.ae.containers** package (JAVA) \ namespace (.NET)

5.18.3.1 Create Key

EIDA ID card stores the keys in a dedicated key file so the "**createKey**" API first check if the input key file ID corresponds to an existing key file on the card then it creates a key file under the Defence container (if the key file is not already created) and finally it writes the input key value (3 DES -16 bytes) in the specified key index (Key Number).

Note: after all the keys are initialized the read and update on the key file(s) should be locked using FreezeAccessCondtions.

Please refer to section 5.55 for the context in which `CreateKey` shall be called.

Parameters

`byte KeyFileID`	[in] identifier of the key file, KeyFileID value should be greater then or equal 0x00 and less then or equal 0x1F skipping the values 0x10 and 0x01 as it is reserved, File IDs must be unique per each container allover the files types .
`byte KeyNumber`	[in] identifier of the key to be created on the key file, KeyNumber value should be less then 0x04 so maximum 4 keys are created per each key file.
`byte [] KeyValue`	[in] 16 byte array key value to be written.

Java Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      DefenseContainer Defense = new DefenseContainer(reader);
      Defense.createKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      DefenseContainer Defense = new DefenseContainer(reader);
      Defense.CreateKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.3.2 Create PIN

This API creates a PIN file under Defence application (if it's not already created) and update it with the input byte array PIN value to initialize a PIN code used for Pin verification. PIN number on the file is specified in the input parameters.

EIDA ID card stores the application PIN in a dedicated PIN file, each containers can contain only one PIN file, the "createPin" API first check if the PIN file is created then it creates a PIN file under the Defence container (if the PIN file is not already created) and finally it writes the input PIN value in the specified PIN index (PIN Number).

Note:

- Since only one PIN file can be created under the Health and Insurance container a fixed file identifier ; 0x10 is dedicated to the PIN File.
- After all the PINs are initialized the read and update on the Pin files should be locked using FreezeAccessCondtions.

Please refer to section 5.18 for the context in which "**createPin**" shall be called.

Parameters

`byte PinNumber`	[in] identifier of the PIN to be created on the application, PinNumber values should be less then 0x08 (the PIN file application can store maximum number of 8 PIN codes)
`byte []PinValue`	[in] 8 bytes PIN value byte array

Java Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      DefenseContainer Defense = new DefenseContainer (reader);
      Defense.createPin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      DefenseContainer Defense = new DefenseContainer (reader);
      DefenseContainer.CreatePin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.3.3 Create Data File

This API creates a Data file under Defence application with the identifier specified in FileID parameter with the access conditions specified in ReadAC and UpdateAC which are instances of the FAC structure specified below, the file size will be set to the size specified in the FileSize parameter.

Please refer to section 5.18 for the context in which `createDataFile` shall be called.

Parameters

`Byte FileID`	[in] identifier of the file to be created, FileID should be greater than 0x00 and less than or equal 0x FF skipping 0x01 and

0x10 since it reserved values. File IDs must be unique per each container allover the files types .

`FAC ReadAC` [in] a pointer to file read access condition that is an isntance of the FAC structure defined below

`FAC UpdateAC` [in] a pointer to file update access condition that is an isntance of the FAC structure defined below

FAC structure

Access conditions are specified using the class **FAC** in the package/namespace **emiratesid.ae.containers** FAC contains the following members :

byte ProtectionLevel : specifies the protection level of the access condition
0 : No Protection(Secure Messaging might be configured if Key file is not 0)
1 : Protected by PIN

byte KeyFile : key file stores the key(s) used for secure messaging (0 if secure messaging is not required)

byte PinNumber: PIN number that will be used to protect the file, please note that if a PIN number is sepecified then a KeyFile must be specified be cause the PIN is always sent to the card in ciphered mode using any key in the specfied key file.

To create a data file for a specific container, initialize an object from that container class with a valid reader object and then call **CreateDataFile()** method

Java Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
      DefenseContainer Defense= new DefenseContainer (reader);
      Defense.createDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
      DefenseContainer Defense= new DefenseContainer (reader);
      Defense.CreateDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.3.4 Freeze Access Conditions

This API does the following operations on any elementary file (Key file, PIN file or Data file):

- Freeze the Access condition so that I can't be changed anymore
- Lock a specific access (Read\Update) if required so that no more Read\Update operation is allowed any more on the file

Please refer to section 5.18 for the context in which freezeAccessConditions shall be called.

Note: once the access condition is locked it can't be unlocked, use this API at the end of the personalization phase to switch to the application phase

Parameters

`byte FileID`	[in] identifier of the file which access condition will be locked, use 0x10 as FileID to lock access condition on the PIN File
`bool lockUpdate`	[in]if this flag is true update will be locked on the selected file
`bool readUpdate`	[in] if this flag is true read will be locked on the selected file

Java Example

```
byte fileID = 0x05;
try {
      DefenseContainer Defense= new DefenseContainer (reader);
      Defense.freezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
      DefenseContainer Defense = new DefenseContainer(reader);
Defense.FreezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.3.5 Update Binary

This API updates binary data on a data file specified in FileID parameter. Depending on the data file access conditions Key and PIN references and values might be needed in the input parameters to establish secure messaging or verify PIN before updating data file.

Please refer to section 5.18 for the context in which updateData shall be called.

Parameters

`byte FileID` [in] identifier of the file to be updated

`byte PinNumber` [in] identifier of the PIN used for pin verification (if needed)

`byte[] PinValue` [in] 8 bytes array holding the value of the pin code .this value is NULL if PIN verification is not needed

`byte` KeyFileID [in] identifier of the key file of the key used to establish secure messaging (if needed)

`byte KeyNumber` [in] identifier of the key used to establish secure messaging (if needed)

`byte] KeyValue` [in] 16 bytes array holding the key value used to establish secure messaging. This value is NULL if secure messaging is not needed

`int offset` [in] offset from which the update data is written on the file

`byte[] UpdateData` [in] byte array holding update data to be written on the file

Java Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
      DefenseContainer Defense= new DefenseContainer (reader);
      Defense.updateData(fileID, pinNum, pinVal, keyFileID, keyNum,
keyVal, offset, data);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
      DefenseContainer Defense= new DefenseContainer (reader);
Defense.UpdateData(fileID, pinNum, pinVal, keyFileID, keyNum, keyVal,
offset, data);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.3.6 Read Binary

This API reads the binary data from data file specified by FileID parameter. Depending on the data file access conditions Key and PIN references and values might be needed to establish secure messaging or verify PIN before reading data file.

Please refer to section 5.18 for the context in which readData shall be called.

Parameters

`byte FileID`	[in] identifier of the file to be read
`byte PinNumber`	[in] identifier of the PIN used for pin verification (if needed)
`byte[] PinValue`	[in] 8 bytes array holding the value of the pin code (if needed).this value is NULL if PIN verification is not needed
`byte KeyFileID`	[in] identifier of the key file of the key used to establish secure messaging (if needed)
byte KeyNumber	[in] identifier of the key used to establish secure messaging (if needed)
byte KeyValue	[in] 16 bytes array holding the key value used to establish secure messaging (if needed). This value is NULL if secure messaging is not needed
int offset	[in] offset from which data is read from the file
int size	[in] number of bytes to be read from the file
byte[] Data	[out] byte array holding data retreived from the file

Java Example

```
byte fileID = 0x05;
try {
	DefenseContainer Defense= new DefenseContainer (reader);
	byte[] data = Defense.readData(fileID, pinNum, pinVal, keyFileID,
keyNum, keyVal, offset, readSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
	DefenseContainer Defense= new DefenseContainer (reader);
byte[] data = Defense.ReadData(fileID, pinNum, pinVal, keyFileID, keyNum,
keyVal, offset, readSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.4 Driving License

The APIs managing the Driving License container are exposed by the class **DrivingLicenseContainer** that is defined in the **emiratesid.ae.containers** package (JAVA) \ namespace (.NET)

5.18.4.1 Create Key

EIDA ID card stores the keys in a dedicated key file so the "**createKey**" API first check if the input key file ID corresponds to an existing key file on the card then it creates a key file under the Driving License container (if the key file is not already created) and finally it writes the input key value (3 DES -16 bytes) in the specified key index (Key Number).

Note: after all the keys are initialized the read and update on the key file(s) should be locked using FreezeAccessCondtions.

Please refer to section 5.55 for the context in which `CreateKey` shall be called.

Parameters

`byte KeyFileID` [in] identifier of the key file, KeyFileID value should be greater then or equal 0x00 and less then or equal 0x1F skipping the values 0x10 and 0x01 as it is reserved, File IDs must be unique per each container allover the files types .

`byte KeyNumber` [in] identifier of the key to be created on the key file, KeyNumber value should be less then 0x04 so maximum 4 keys are created per each key file.

`byte [] KeyValue` [in] 16 byte array key value to be written.

Java Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      DrivingLicenseContainer Container = new
DrivingLicenseContainer(reader);
      Container.createKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      DrivingLicenseContainer Container = new DrivingLicenseContainer
(reader);
      Container.CreateKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.4.2 Create PIN

This API creates a PIN file under Driving License application (if it's not already created) and update it with the input byte array PIN value to initialize a PIN code used for Pin verification. PIN number on the file is specified in the input parameters.

EIDA ID card stores the application PIN in a dedicated PIN file, each containers can contain only one PIN file, the "createPin" API first check if the PIN file is created then it creates a PIN file under the Driving License container (if the PIN file is not already created) and finally it writes the input PIN value in the specified PIN index (PIN Number).

Note:

- Since only one PIN file can be created under the Health and Insurance container a fixed file identifier; 0x10 is dedicated to the PIN File.
- After all the PINs are initialized the read and update on the Pin files should be locked using FreezeAccessCondtions.

Please refer to section 5.18 for the context in which "**createPin**" shall be called.

Parameters

`byte PinNumber`	[in] identifier of the PIN to be created on the application, PinNumber values should be less then 0x08 (the PIN file application can store maximum number of 8 PIN codes)
`byte []PinValue`	[in] 8 bytes PIN value byte array

Java Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      DrivingLicenseContainer Container = new
DrivingLicenseContainer(reader);
      Container.createPin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      DrivingLicenseContainer Container = new
DrivingLicenseContainer(reader);
      Container.CreatePin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.4.3 Create Data File

This API creates a Data file under Driving License application with the identifier specified in FileID parameter with the access conditions specified in ReadAC and UpdateAC which are instances of the FAC structure specified below, the file size will be set to the size specified in the FileSize parameter.

Please refer to section 5.18 for the context in which `createDataFile` shall be called.

Parameters

`Byte FileID`	[in] identifier of the file to be created, FileID should be greater than 0x00 and less than or equal 0x FF skipping 0x01 and 0x10 since it reserved values. File IDs must be unique per each container allover the files types .
`FAC ReadAC`	[in] a pointer to file read access condition that is an isntance of the FAC structure defined below
`FAC UpdateAC`	[in] a pointer to file update access condition that is an isntance of the FAC structure defined below

FAC structure

Access conditions are specified using the class **FAC** in the package/namespace **emiratesid.ae.containers** FAC contains the following members :

byte ProtectionLevel : specifies the protection level of the access condition
0 : No Protection(Secure Messaging might be configured if Key file is not 0)
1 : Protected by PIN
byte KeyFile : key file stores the key(s) used for secure messaging (0 if secure messaging is not required)
byte PinNumber: PIN number that will be used to protect the file, please note that if a PIN number is sepecified then a KeyFile must be specified be cause the PIN is always sent to the card in ciphered mode using any key in the specfied key file.

To create a data file for a specific container, initialize an object from that container class with a valid reader object and then call **CreateDataFile()** method

Java Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
	DrivingLicenseContainer Container= new
DrivingLicenseContainer(reader);
	Container.createDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
	// Error occurred
```

C# Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
	DrivingLicenseContainer Container= new
DrivingLicenseContainer(reader);
	Container.CreateDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.4.4 Freeze Access Conditions

This API does the following operations on any elementary file (Key file, PIN file or Data file):

- Freeze the Access condition so that I can't be changed anymore

- Lock a specific access (Read\Update) if required so that no more Read\Update operation is allowed any more on the file

Please refer to section 5.18 for the context in which freezeAccessConditions shall be called.

Note: once the access condition is locked it can't be unlocked, use this API at the end of the personalization phase to switch to the application phase

Parameters

`byte FileID` [in] identifier of the file which access condition will be locked, use 0x10 as FileID to lock access condition on the PIN File

`bool lockUpdate` [in]if this flag is true update will be locked on the selected file

`bool readUpdate` [in] if this flag is true read will be locked on the selected file

Java Example

```
byte fileID = 0x05;
try {
	DrivingLicenseContainer Container = new
DrivingLicenseContainer(reader);
	Container.freezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
	DrivingLicenseContainer Container= new DrivingLicenseContainer
(reader);
Container.FreezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.4.5 Update Binary

This API updates binary data on a data file specified in FileID parameter. Depending on the data file access conditions Key and PIN references and values might be needed in the input parameters to establish secure messaging or verify PIN before updating data file.

Please refer to section 5.18 for the context in which updateData shall be called.

Parameters

`byte FileID` [in] identifier of the file to be updated

`byte PinNumber` [in] identifier of the PIN used for pin verification (if needed)

`byte[] PinValue` [in] 8 bytes array holding the value of the pin code .this value is NULL if PIN verification is not needed

`byte KeyFileID` [in] identifier of the key file of the key used to establish secure messaging (if needed)

`byte KeyNumber` [in] identifier of the key used to establish secure messaging (if needed)

`byte] KeyValue` [in] 16 bytes array holding the key value used to establish secure messaging. This value is NULL if secure messaging is not needed

`int offset` [in] offset from which the update data is written on the file

`byte[] UpdateData` [in] byte array holding update data to be written on the file

Java Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
      DrivingLicenseContainer Container = new
DrivingLicenseContainer(reader);
      Container.updateData(fileID, pinNum, pinVal, keyFileID, keyNum, keyVal,
offset, data);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
      DrivingLicenseContainer Container= new
DrivingLicenseContainer(reader);
Container.UpdateData(fileID, pinNum, pinVal, keyFileID, keyNum, keyVal,
offset, data);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.4.6 Read Binary

This API reads the binary data from data file specified by FileID parameter depending on the data file access conditions Key and PIN references and values might be needed to establish secure messaging or verify PIN before reading data file.

Please refer to section 5.18 for the context in which readData shall be called.

Parameters

`byte FileID`	[in] identifier of the file to be read
`byte PinNumber`	[in] identifier of the PIN used for pin verification (if needed)
byte[] PinValue	[in] 8 bytes array holding the value of the pin code (if needed).this value is NULL if PIN verification is not needed
byte KeyFileID	[in] identifier of the key file of the key used to establish secure messaging (if needed)
byte KeyNumber	[in] identifier of the key used to establish secure messaging (if needed)
byte KeyValue	[in] 16 bytes array holding the key value used to establish secure messaging (if needed). This value is NULL if secure messaging is not needed
int offset	[in] offset from which data is read from the file
int size	[in] number of bytes to be read from the file
byte[] Data	[out] byte array holding data retreived from the file

Java Example

```
byte fileID = 0x05;
try {
	DrivingLicenseContainer Container = new
DrivingLicenseContainer(reader);
	byte[] data = Container.readData(fileID, pinNum, pinVal, keyFileID,
keyNum, keyVal, offset, readSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
      DrivingLicenseContainer Container= new
DrivingLicenseContainer(reader);
byte[] data = Container.ReadData(fileID, pinNum, pinVal, keyFileID, keyNum,
keyVal, offset, readSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.5 Qualification

The APIs managing the Qualification container are exposed by the class **QualificationContainer** that is defined in the **emiratesid.ae.containers** package (JAVA) \ namespace (.NET).

5.18.5.1 Create Key

EIDA ID card stores the keys in a dedicated key file so the "**createKey**" API first check if the input key file ID corresponds to an existing key file on the card then it creates a key file under the Qualification container (if the key file is not already created) and finally it writes the input key value (3 DES -16 bytes) in the specified key index (Key Number).

Note: after all the keys are initialized the read and update on the key file(s) should be locked using FreezeAccessCondtions.

Please refer to section 5.55 for the context in which `CreateKey` shall be called.

Parameters

`byte KeyFileID` [in] identifier of the key file, KeyFileID value should be greater then or equal 0x00 and less then or equal 0x1F skipping the values 0x10 and 0x01 as it is reserved, File IDs must be unique per each container allover the files types .

`byte KeyNumber` [in] identifier of the key to be created on the key file, KeyNumber value should be less then 0x04 so maximum 4 keys are created per each key file.

`byte [] KeyValue` [in] 16 byte array key value to be written.

Java Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      QualificationContainer Container = new
QualificationContainer(reader);
      Container.createKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      QualificationContainer Container = new QualificationContainer
(reader);
      Container.CreateKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.5.2 Create PIN

This API creates a PIN file under Qualification application (if it's not already created) and update it with the input byte array PIN value to initialize a PIN code used for Pin verification. PIN number on the file is specified in the input parameters.

EIDA ID card stores the application PIN in a dedicated PIN file, each containers can contain only one PIN file, the "createPin" API first check if the PIN file is created then it creates a PIN file under the Qualification container (if the PIN file is not already created) and finally it writes the input PIN value in the specified PIN index (PIN Number).

Note:

- Since only one PIN file can be created under the Health and Insurance container a fixed file identifier ; 0x10 is dedicated to the PIN File.
- After all the PINs are initialized the read and update on the Pin files should be locked using FreezeAccessCondtions.

Please refer to section 5.18 for the context in which "**createPin**" shall be called.

Parameters

`byte PinNumber`	[in] identifier of the PIN to be created on the application, PinNumber values should be less then 0x08 (the PIN file application can store maximum number of 8 PIN codes)

`byte []PinValue`	[in] 8 bytes PIN value byte array

Java Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      QualificationContainer Container = new
QualificationContainer(reader);
      Container.createPin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      QualificationContainer Container = new
QualificationContainer(reader);
      Container.CreatePin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.5.3 Create Data File

This API creates a Data file under Qualification application with the identifier specified in FileID parameter with the access conditions specified in ReadAC and UpdateAC which are instances of the FAC structure specified below, the file size will be set to the size specified in the FileSize parameter.

Please refer to section 5.18 for the context in which `createDataFile` shall be called.

Parameters

`Byte FileID`	[in] identifier of the file to be created, FileID should be greater than 0x00 and less than or equal 0x FF skipping 0x01 and 0x10 since it reserved values. File IDs must be unique per each container allover the files types .
`FAC ReadAC`	[in] a pointer to file read access condition that is an isntance of the FAC structure defined below
`FAC UpdateAC`	[in] a pointer to file update access condition that is an isntance of the FAC structure defined below

FAC structure

Access conditions are specified using the class **FAC** in the package/namespace **emiratesid.ae.containers** FAC contains the following members :

byte ProtectionLevel : specifies the protection level of the access condition
0 : No Protection(Secure Messaging might be configured if Key file is not 0)
1 : Protected by PIN

byte KeyFile : key file stores the key(s) used for secure messaging (0 if secure messaging is not required)

byte PinNumber: PIN number that will be used to protect the file, please note that if a PIN number is sepecified then a KeyFile must be specified be cause the PIN is always sent to the card in ciphered mode using any key in the specfied key file.

To create a data file for a specific container, initialize an object from that container class with a valid reader object and then call **CreateDataFile()** method

Java Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
      QualificationContainer Container= new
QualificationContainer(reader);
      Container.createDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
      // Error occurred
```

C# Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
      QualificationContainer Container= new
QualificationContainer(reader);
      Container.CreateDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.5.4 Freeze Access Conditions

This API does the following operations on any elementary file (Key file, PIN file or Data file):

- Freeze the Access condition so that I can't be changed anymore

- Lock a specific access (Read\Update) if required so that no more Read\Update operation is allowed any more on the file

Please refer to section 5.18 for the context in which freezeAccessConditions shall be called.

Note: once the access condition is locked it can't be unlocked, use this API at the end of the personalization phase to switch to the application phase

Parameters

`byte FileID`	[in] identifier of the file which access condition will be locked, use 0x10 as FileID to lock access condition on the PIN File
`bool lockUpdate`	[in]if this flag is true update will be locked on the selected file
`bool readUpdate`	[in] if this flag is true read will be locked on the selected file

Java Example

```
byte fileID = 0x05;
try {
	QualificationContainer Container = new
QualificationContainer(reader);
	Container.freezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
	QualificationContainer Container= new QualificationContainer
(reader);
Container.FreezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.5.5 Update Binary

This API updates binary data on a data file specified in FileID parameter. Depending on the data file access conditions Key and PIN references and values might be needed in the input parameters to establish secure messaging or verify PIN before updating data file.

Please refer to section 5.18 for the context in which updateData shall be called.

Parameters

`byte FileID` [in] identifier of the file to be updated

`byte PinNumber` [in] identifier of the PIN used for pin verification (if needed)

`byte[] PinValue` [in] 8 bytes array holding the value of the pin code .this value is NULL if PIN verification is not needed

`byte KeyFileID` [in] identifier of the key file of the key used to establish secure messaging (if needed)

`byte KeyNumber` [in] identifier of the key used to establish secure messaging (if needed)

`byte] KeyValue` [in] 16 bytes array holding the key value used to establish secure messaging. This value is NULL if secure messaging is not needed

`int offset` [in] offset from which the update data is written on the file

`byte[] UpdateData` [in] byte array holding update data to be written on the file

Java Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
	QualificationContainer Container = new
QualificationContainer(reader);
	Container.updateData(fileID, pinNum, pinVal, keyFileID, keyNum,
keyVal, offset, data);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
	QualificationContainer Container= new
QualificationContainer(reader);
Container.UpdateData(fileID, pinNum, pinVal, keyFileID, keyNum, keyVal,
offset, data);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.5.6 Read Binary

This API reads the binary data from data file specified by FileID parameter depending on the data file access conditions Key and PIN references and values might be needed to establish secure messaging or verify PIN before reading data file.

Please refer to section 5.18 for the context in which readData shall be called.

Parameters

`byte FileID`	[in] identifier of the file to be read
`byte PinNumber`	[in] identifier of the PIN used for pin verification (if needed)
`byte[] PinValue`	[in] 8 bytes array holding the value of the pin code (if needed).this value is NULL if PIN verification is not needed
`byte KeyFileID`	[in] identifier of the key file of the key used to establish secure messaging (if needed)
byte KeyNumber	[in] identifier of the key used to establish secure messaging (if needed)
byte KeyValue	[in] 16 bytes array holding the key value used to establish secure messaging (if needed). This value is NULL if secure messaging is not needed
int offset	[in] offset from which data is read from the file
int size	[in] number of bytes to be read from the file
byte[] Data	[out] byte array holding data retreived from the file

Java Example

```
byte fileID = 0x05;
try {
	QualificationContainer Container = new
QualificationContainer(reader);
	byte[] data = Container.readData(fileID, pinNum, pinVal, keyFileID,
keyNum, keyVal, offset, readSize);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
      QualificationContainer Container= new
QualificationContainer(reader);
byte[] data = Container.ReadData(fileID, pinNum, pinVal, keyFileID,
keyNum, keyVal, offset, readSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.6 Social Services

The APIs managing the Social Services container are exposed by the class **Social SocialServicesContainer** that is defined in the **emiratesid.ae.containers** package (JAVA) \ namespace (.NET).

5.18.6.1 Create Key

EIDA ID card stores the keys in a dedicated key file so the "**createKey**" API first check if the input key file ID corresponds to an existing key file on the card then it creates a key file under the Social Services container (if the key file is not already created) and finally it writes the input key value (3 DES -16 bytes) in the specified key index (Key Number).

Note: after all the keys are initialized the read and update on the key file(s) should be locked using FreezeAccessCondtions.

Please refer to section 5.55 for the context in which `CreateKey` shall be called.

Parameters

`byte KeyFileID`	[in] identifier of the key file, KeyFileID value should be greater then or equal 0x00 and less then or equal 0x1F skipping the values 0x10 and 0x01 as it is reserved, File IDs must be unique per each container allover the files types .
`byte KeyNumber`	[in] identifier of the key to be created on the key file, KeyNumber value should be less then 0x04 so maximum 4 keys are created per each key file.
`byte [] KeyValue`	[in] 16 byte array key value to be written.

Java Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      SocialServicesContainer Container = new
SocialServicesContainer(reader);
      Container.createKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte keyFileID = 0x02;
byte keyNum = 0x01;
byte[] keyVal; // 16 bytes representing key value
try {
      SocialServicesContainer Container = new SocialServicesContainer
(reader);
      Container.CreateKey(keyFileID, keyNum, keyVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.6.2 Create PIN

This API creates a PIN file under Social Services application (if it's not already created) and update it with the input byte array PIN value to initialize a PIN code used for Pin verification. PIN number on the file is specified in the input parameters.

EIDA ID card stores the application PIN in a dedicated PIN file, each containers can contain only one PIN file, the "createPin" API first check if the PIN file is created then it creates a PIN file under the Social Services container (if the PIN file is not already created) and finally it writes the input PIN value in the specified PIN index (PIN Number).

Note:

- Since only one PIN file can be created under the Health and Insurance container a fixed file identifier; 0x10 is dedicated to the PIN File.
- After all the PINs are initialized the read and update on the Pin files should be locked using FreezeAccessCondtions.

Please refer to section 5.18 for the context in which "**createPin**" shall be called.

Parameters

`byte PinNumber`	[in] identifier of the PIN to be created on the application, PinNumber values should be less then 0x08 (the PIN file application can store maximum number of 8 PIN codes)
`byte []PinValue`	[in] 8 bytes PIN value byte array

Java Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      SocialServicesContainer Container = new
SocialServicesContainer(reader);
      Container.createPin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte pinNum = 0x01;
byte[] pinVal; // 8 bytes representing pin value
try {
      SocialServicesContainer Container = new
SocialServicesContainer(reader);
      Container.CreatePin(pinNum, pinVal);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.6.3 Create Data File

This API creates a Data file under Social Services application with the identifier specified in FileID parameter with the access conditions specified in ReadAC and UpdateAC which are instances of the FAC structure specified below, the file size will be set to the size specified in the FileSize parameter.

Please refer to section 5.18 for the context in which `createDataFile` shall be called.

Parameters

`Byte FileID`	[in] identifier of the file to be created, FileID should be greater than 0x00 and less than or equal 0x FF skipping 0x01 and 0x10 since it reserved values. File IDs must be unique per each container allover the files types .
`FAC ReadAC`	[in] a pointer to file read access condition that is an isntance of the FAC structure defined below
`FAC UpdateAC`	[in] a pointer to file update access condition that is an isntance of the FAC structure defined below

FAC structure

Access conditions are specified using the class **FAC** in the package/namespace **emiratesid.ae.containers** FAC contains the following members:

byte ProtectionLevel : specifies the protection level of the access condition
0 : No Protection(Secure Messaging might be configured if Key file is not 0)
1 : Protected by PIN

byte KeyFile : key file stores the key(s) used for secure messaging (0 if secure messaging is not required)

byte PinNumber: PIN number that will be used to protect the file, please note that if a PIN number is sepecified then a KeyFile must be specified be cause the PIN is always sent to the card in ciphered mode using any key in the specfied key file.

To create a data file for a specific container, initialize an object from that container class with a valid reader object and then call **CreateDataFile()** method

Java Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
      SocialServicesContainer Container= new
SocialServicesContainer(reader);
      Container.createDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
      // Error occurred
```

C# Example

```
byte fileID = 0x05;
FAC readAC = new FAC(); // fill readAC
FAC updateAC = new FAC(); // fill updateAC
int fileSize = 100;
try {
      SocialServicesContainer Container= new
SocialServicesContainer(reader);
      Container.CreateDataFile(fileID, readAC, updateAC, fileSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.18.6.4 Freeze Access Conditions

This API does the following operations on any elementary file (Key file, PIN file or Data file):

- Freeze the Access condition so that I can't be changed anymore

- Lock a specific access (Read\Update) if required so that no more Read\Update operation is allowed any more on the file

Please refer to section 5.18 for the context in which freezeAccessConditions shall be called.

Note: once the access condition is locked it can't be unlocked, use this API at the end of the personalization phase to switch to the application phase

Parameters

`byte FileID`	[in] identifier of the file which access condition will be locked, use 0x10 as FileID to lock access condition on the PIN File
`bool lockUpdate`	[in]if this flag is true update will be locked on the selected file
`bool readUpdate`	[in] if this flag is true read will be locked on the selected file

Java Example

```
byte fileID = 0x05;
try {
	SocialServicesContainer Container = new
SocialServicesContainer(reader);
	Container.freezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
	SocialServicesContainer Container= new SocialServicesContainer
(reader);
Container.FreezeAccessConditions(fileID, true, false);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.6.5 Update Binary

This API updates binary data on a data file specified in FileID parameter. Depending on the data file access conditions Key and PIN references and values might be needed in the input parameters to establish secure messaging or verify PIN before updating data file.

Please refer to section 5.18 for the context in which updateData shall be called.

Parameters

`byte FileID` [in] identifier of the file to be updated

`byte PinNumber` [in] identifier of the PIN used for pin verification (if needed)

`byte[] PinValue` [in] 8 bytes array holding the value of the pin code .this value is NULL if PIN verification is not needed

`byte KeyFileID` [in] identifier of the key file of the key used to establish secure messaging (if needed)

`byte KeyNumber` [in] identifier of the key used to establish secure messaging (if needed)

`byte[] KeyValue` [in] 16 bytes array holding the key value used to establish secure messaging. This value is NULL if secure messaging is not needed

`int offset` [in] offset from which the update data is written on the file

`byte[] UpdateData` [in] byte array holding update data to be written on the file

Java Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
	SocialServicesContainer Container = new
SocialServicesContainer(reader);
	Container.updateData(fileID, pinNum, pinVal, keyFileID, keyNum,
keyVal, offset, data);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

C# Example

```
byte fileID = 0x05;
byte[] data; // binary data to be written
try {
	SocialServicesContainer Container= new
SocialServicesContainer(reader);
Container.UpdateData(fileID, pinNum, pinVal, keyFileID, keyNum, keyVal,
offset, data);
} catch (MiddlewareException ex) {
	// Error occurred
}
```

5.18.6.6 Read Binary

This API reads the binary data from data file specified by FileID parameter depending on the data file access conditions Key and PIN references and values might be needed to establish secure messaging or verify PIN before reading data file.

Please refer to section 5.18 for the context in which readData shall be called.

Parameters

byte FileID	[in] identifier of the file to be read
byte PinNumber	[in] identifier of the PIN used for pin verification (if needed)
byte[] PinValue	[in] 8 bytes array holding the value of the pin code (if needed).this value is NULL if PIN verification is not needed
byte KeyFileID	[in] identifier of the key file of the key used to establish secure messaging (if needed)
byte KeyNumber	[in] identifier of the key used to establish secure messaging (if needed)
byte [] KeyValue	[in] 16 bytes array holding the key value used to establish secure messaging (if needed). This value is NULL if secure messaging is not needed
int offset	[in] offset from which data is read from the file
int size	[in] number of bytes to be read from the file
byte[] Data	[out] byte array holding data retreived from the file

Java Example

```
byte fileID = 0x05;
try {
      SocialServicesContainer Container = new
SocialServicesContainer(reader);
      byte[] data = Container.readData(fileID, pinNum, pinVal, keyFileID,
keyNum, keyVal, offset, readSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

C# Example

```
byte fileID = 0x05;
try {
      SocialServicesContainer Container= new
SocialServicesContainer(reader);
byte[] data = Container.ReadData(fileID, pinNum, pinVal, keyFileID,
keyNum, keyVal, offset, readSize);
} catch (MiddlewareException ex) {
      // Error occurred
}
```

5.19 Update Modifiable Data

Since EIDA ID cards allow change in some has some data files, hence the toolkit offers this feature through Kernel and APIs. This section refers to the files where data can be modifies are stored as “modifiable data files”.

Modifiable data files exist in three applications; ID application, Address application and Family Book application. V2 cards only have address and family book applications in addition to the ID application, however V1 cards has only ID application.

Toolkit enables the update of modifiable data by accepting the binary file at once, and updating the whole file in one step. The input file must be in TLV format containing valid data signature and data fields specified in the EIDA ID card RM.

5.19.1 Update Modifiable Public Data

This API allows updating modifiable public data as it replaces the whole modifiable public data file content with the input byte array. The Toolkit validates the input byte array TLV format and signature before writing on the card.

The input array must be in TLV format containing valid data signature and data fields specified in the EIDA ID card RM (section 4.2.5.3) the signature is generated using the EIDA signing key ,please refer to EIDA card RM (section 4.2.4) for details on signature generation.

To update public modifiable data, a call to **UpdateModifiableData()** method in **PublicDataFacade** object passing the file content in binary as described above will update modifiable data in public data application.

Java Example

```
byte[] data;
// ... prepare data to be updated ...

try {
PublicDataFacade publicDataFacade = reader.getPublicDataFacade();
publicDataFacade.updateModifiableData(data);
} catch (MiddlewareException e) {
      // ... Handle exceptions here
}
```

C# Example

```
byte[] data;
// ... prepare data to be updated...

try {
PublicDataFacade publicDataFacade = reader.GetPublicDataFacade();
publicDataFacade.UpdateModifiableData(data);
} catch (MiddlewareException e) {
      // … Handle exceptions here
}
```

5.19.2 Update Address Data

This API allows updating an address data stored in work address file or home address file. The Toolkit validates the input byte array TLV format and signature before writing on the card.

The input array must be in TLV format containing valid data signature and data fields specified in the EIDA ID card RM (section 4.2.5.11) the signature is generated using the EIDA signing key ,please refer to EIDA card RM (section 4.2.4) for details on signature generation.

The API expects the address file ID that can be one of the two static constants found in Public Data Facade

- HOME_ADDRESS_FILE_ID
- WORK_ADDRESS_FILE_ID

Java Example

```
byte[] data;
// ... prepare home address data to be updated...

try {
PublicDataFacade publicDataFacade = reader.getPublicDataFacade();
publicDataFacade.updateAddress(PublicDataFacade.HOME_ADDRESS_FILE_ID,
data);
} catch (MiddlewareException e) {
      // ... Handle exceptions here
}
```

C# Example

```
byte[] data;
// ... prepare home address data to be updated...

try {
PublicDataFacade publicDataFacade = reader.GetPublicDataFacade();
publicDataFacade.UpdateAddress(PublicDataFacade.HOME_ADDRESS_FILE_ID,
data);
} catch (MiddlewareException e) {
      // … Handle exceptions here
}
```

5.19.3 Update Family Book Data

This API allows updating the whole content of any of the family book application data files which identifier is specified in the FileID parameter with the input byte array. The Toolkit validates the input byte array TLV format and signature before writing on the card.

The Toolkit validates the input byte array format and signature and replaces the existing family book data file value with the input byte array value.

The input must be in TLV format containing valid data signature and data fields specified in the EIDA ID card RM (section 4.2.5.11) ,the signature is generated using the EIDA signing key .please refer to EIDA card RM (section 4.2.4) for details on signature generation.

Family book data can be only updated in V2 cards, to address data, a call to **UpdateFamilyBook()** method in **FamilyBookDataFacade** object passing file ID, and the file content in binary as described above will update specific family member data in family book application.

Since family book contains a family head, up to 4 wives and 20 children files, file Ids for these files are found as static constants in FamilyBookDataFacade class for example

- FAMILY_HEAD_FILE_ID

- WIFE_3_FILE_ID
- CHILD_9_FILE_ID
- …

Java Example

```
byte[] data;
// ... prepare family book data to be updated...

try {
FamilyBookDataFacade facade = reader.getFamilyBookDataFacade();
facade.updateFamilyBook(FamilyBookDataFacade.WIFE_1_FILE_ID, data);
} catch (MiddlewareException e) {
      // ... Handle exceptions here
}
```

C# Example

```
byte[] data;
// ... prepare family book data to be updated...

try {
FamilyBookDataFacade facade = reader.GetFamilyBookDataFacade();
facade.UpdateFamilyBook(FamilyBookDataFacade.WIFE_1_FILE_ID, data);
} catch (MiddlewareException e) {
      // … Handle exceptions here
}
```

6 Toolkit Web-Components

The Toolkit web components are the web facet of the EIDA ID Card Toolkit. They enable the Toolkit functionalities to build web based applications.

The EIDA ID Card Toolkit contains three web components namely PublicDataActiveX, DigitalSignatureActiveX for Microsoft Internet Explorer and EIDA_IDCard_Applet for Microsoft Internet explorer, Mozilla Firefox and Google Chrome.

DigitalSignatureActiveX has all the PKI PIN management functions while PublicDataActiveX has all the other Toolkit functions. For java, all the Toolkit functions are exported in EIDA_IDCard_Applet.

Toolkit web components requires toolkit to be installed and configured on the client machine that runs the web components, as it depend on toolkit core DLL files.

EIDA toolkit also provides other type of web components that have no dependencies, and runs without any required software on the client machine. Section 7 describes the Zero footprint web components.

This section of the document provides the detailed description on how to use web-components in building web based applications.

As part of the installation, the Toolkit comes with the “WebComponents” folder which has the below subfolders.

- “ActiveX” folder:
    - PublicDataActiveX.dll: .Net implementation for the public data ActiveX
    - DigitalSignatureActiveX.dll: .Net implementation for the public data ActiveX
- “Java Applets” folder:
    - EIDA_IDCard_Applet.jar: the java archive file that contains the applet.

Also toolkit installation provides a sample demo for each of the above components, these samples can be found in “Samples\WebComponents” folder with the below structure

- “Public Data ActiveX” folder:
    - PublicDataActiveX.html: this HTML page contains a sample on how to embed the ActiveX in a web page.
    - JS (JavaScript) folder: contains two JavaScript files:
        - eida_webcomponents.js: contains all the functions used to interact with the ActiveX like initializing, reading public data from the card
        - occupations.js: used to display the occupation of the card holder as a text instead of the occupation ID

- errors.js: contains the list of errors and its corresponding description
- fingers.js: contains the list of fingerprint indices and its description

- Images folder: contains the images used in the HTML web page.

- "Digital Signature ActiveX" folder:
  - DigitalSignatureActiveX.html: this HTML page contains a sample on how to embed the ActiveX in a web page.
  - JS (JavaScript) folder: contains two JavaScript files:
    - eida_pki_webcomponents.js: contains all the functions used to interact with the ActiveX like initializing, sign data, and export certificates
    - errors.js: used to map and display error codes that's returned from the ActiveX
    - errors.js: contains the list of errors and its corresponding description
    - fingers.js: contains the list of fingerprint indices and its description
  - Images folder: contains the images used in the HTML web page
- "IDCard Applet" folder:
  - PubData_DigitalSigning.html: this HTML page contains a sample on how to embed the Applet in a web page.
  - EIDA_IDCard_Applet.jar: the java archive file that contains the applet.
    - eida_webcomponents.js: contains all the functions exported by the applet.
    - occupations.js: used to display the occupation of the card holder as a text instead of the occupation ID,
    - errors.js: contains the list of errors and its corresponding description
    - fingers.js: contains the list of fingerprint indices and its description
  - Images folder: it is exactly the same folder as the ActiveX/Images folder

6.1 Steps to embed EIDA IDCard Applet on a web page

Include the below javascript files in the html page with the same order this order:

<script language="javascript" src="js/occupations.js"></script>

<script language="javascript" src="js/eida_webcomponents.js"></script>

Add the below code to embed the Public Data Applet in a web page.

```
<APPLET ID="PublicDataApplet" NAME="PublicDataApplet"
CODE="emiratesid.ae.webcomponents.PublicDataApplet"
ARCHIVE="PublicDataApplet.jar" WIDTH="130" HEIGHT="154" >
```

- The ID attribute refers to communicate with the Applet from javascript
- The ARCHIVE attribute is the reference to the Java archive file (JAR) that contains the public data applet.
- The CODE attribute represents the main class of the public data applet
- The WIDTH and the HEIGHT attributes are used to set the dimension of the applet. By default, the applet displays the photography of the card holder. The size of the photograph is 130 x 154 pixels. It is possible to hide the photography by setting the width and the height to zero.

6.2 Steps to embed the Public Data ActiveX on a web page

Include the below javascript files in the html page with the same order this order:

<script language="javascript" src="js/occupations.js"></script>

<script language="javascript" src="js/eida_webcomponents.js"></script>

Add the below code to embed the Public Data ActiveX in a web page.

```
<OBJECT ID="PublicDataActiveX"
 CLASSID="CLSID:A4B3BB86-4A99-3BAE-B211-DB93E8BA008B" WIDTH="130"
 HEIGHT="154" >

This browser does not support ActiveX components.
</OBJECT>
```

- The ID attribute refers to communicate with the ActiveX from Javascript
- The CLASSID attribute is the reference to the Public Data ActiveX by its GUID (Global Unique Identifier).
- The width and the height attributes are used to set the dimension of the ActiveX. Similarly to the Applet, the ActiveX displays the photography of the card holder. To hide the photography, set the width and the height to zero.

6.3 Embedding the Digital Signature ActiveX on a web page

In order to use the Applet/ActiveX functions, two of the below Java script files must be loaded on the html page in the order as specified:

<script type="text/javascript" language="javascript" src="js/errors.js"></script>

<script type="text/javascript" language="javascript" src="js/eida_pki_webcomponents.js"></script>

The code below illustrate on how to embed the Digital Signature ActiveX in a web page.

```
<OBJECT ID="DigitalSignatureWebComponent"
 CLASSID="CLSID:EDE5745C-271D-4480-BE28-AE8BC0DF90BB"          WIDTH="0"
 HEIGHT="0" >

This browser does not support ActiveX components.
</OBJECT>
```

- The ID attribute is used to communicate with the ActiveX from JavaScript
- The CLASSID attribute is the reference to the Digital Signature ActiveX by its GUID (Global Unique Identifier).

The width and the height attributes are used to set the dimension of the ActiveX. The width and height are set to zero as the ActiveX has no UI to display.

6.4 Referencing Toolkit Web Component (Applet or ActiveX)

The developer has to ensure that the variables are set to the id of the referenced component:

Variable name	Js file where it is defined	Referenced component
DigitalSignatureWebComponentName	Digital Signature ActiveX\js\eida_pki_webcomponents.js	DigitalSignatureActiveX
PublicDataWebComponentName	Public Data ActiveX\js\eida_webcomponents.js	PublicDataActiveX
EIDAWebComponentName	IDCard Applet\js\eida_webcomponents.js	EIDA_IDCard_Applet

Note: If the ID of the applet is modified for any reason, also it **must** be modified in the corresponding js file.

6.5 Communication with the Web page

Only after embedding and referencing the relevant Toolkit web components in the page as explained above, developer can invoke any of the Toolkit functions via the JavaScript functions defined in the js files referenced above.

Calling the JavaScript functions in the web page will make the web page code transparent from the Toolkit web component type, i.e. a developer might decide to replace the ActiveX with the applet at any time without changing the code in the web page.

The below sections describes the JavaScript functions corresponds to the Toolkit functions.

6.5.1 Initialize

This function initializes the web component by discovering the list of PC/SC readers connected and selects the first reader with ID card inserted.

- If successful, this function will return an empty String.
- If there is no reader containing the UAE ID Card, “Please insert your card” message will be returned.
- If an error occurred during initialization, “An error occurred when initializing the application, details ….” message will be returned.

```
//method signature
// String Initialize()
var result = Initialize();
```

6.5.2 Initialize contactless reader

This function initializes the web component by discovering the list of PC/SC readers connected and selects the first contactless reader with ID card in range. .This function behaves the same as Initialize except it only works with contactless readers

```
//method signature
// String InitializeContactless()
var result = InitializeContactless();
```

6.5.3 Reading the public data

This function reads cardholder public data from the card and caches into the Applet. In addition, this function validates the data signature implicitly.

- If successful, this function returns an empty String.
- If there is no reader containing the UAE ID Card, “Applet not initialized properly” message will be returned.
- If the read public data returns null, then “Cannot retrieve public data” is returned.
- If an error occurred during initialization, “An error occurred when reading public data, details ….” is returned.

```
//method signature
// String ReadPublicData()
ReadPublicData(refresh, readPhotography, readNonModifiableData,
readModifiableData, signatureValidation);
```

Note that the parameter flags in the above signature to enable or disable reading groups of data, and also enabling signature validation.

Upon successfully calling the ReadPublicData function, the following functions are exposed by the web-component.

Function Name	Description
GetCardSerialNumber()	Returns card serial number in hexadecimal format.
GetPhotography_DataSigned()	Returns the signed data used for data signatures validation
GetCardHolderData_SF3_DataSigned()	
GetCardHolderData_SF5_DataSigned()	
GetPhotography_Signature()	Reads signature of data signed used, also, in the signature validation process. Note that the signatures are validated on the Toolkit implicitly. Nevertheless above six functions were exposed to allow external signature validation.
GetCardHolderData_SF3_Signature()	
GetCardHolderData_SF5_Signature()	
GetIDNumber()	Returns the ID number
GetCardNumber()	Returns the card number
GetPhotography()	Returns the photography in hexadecimal
GetIDType()	Returns IL for resident and IR for citizen.
GetIssueDate()	Returns the issuing date of the date in DD/MM/YYYY.
GetExpiryDate()	Returns the expiry date in DD/MM/YYYY format.
GetArabicTitle()	Return the title in Arabic in UTF-8 encoding.
GetArabicFullName()	Returns the Arabic full name in UTF-8 encoding.

GetTitle()	Returns the title of the card holder.
GetFullName()	Returns the full name of the card holder.
GetSex()	Returns the Sex of the card holder (M: for Male, F: for Female, X: for unknown).
GetArabicNationality()	Returns the Nationality in Rrabic in UTF-8 encoding.
GetNationality()	Returns the nationality of the card holder (3 characters only).
GetDateOfBirth()	Returns the birth date of the card holder in DD/MM/YYYY format.
GetArabicMotherFirstName()	Returns the Arabic mother first name in UTF-8 format.
GetMotherFirstName()	Returns the mother first name.
GetOccupation()	Returns the code of the occupation.
GetMaritalStatus()	Returns the code of the marital status of the card holder.
GetFamilyID()	Available only for UAE citizen.
GetHusbandIDN()	Returns the IDN of the husband, for married women only
GetSponsorType()	Returns the code of the sponsor type
GetSponsorNumber()	Returns the sponsor number.
GetSponsorName():	Returns the sponsor name.
GetResidencyType()	Returns the residency type.
GetResidencyNumber()	Returns the residency number.
GetResidencyExpiryDate()	Returns the residency expiry date in DD/MM/YYYY format.
GetMOIRootCertificate()	Returns the digital signature technical certificate used for card data signature verification in hexadecimal.

6.5.4 Reading public data Extended

Read Card Holder Public Data feature is extended to support reading additional public data fields added in V2 cards such as address, passport information, Company name, Qualification, Field of Study, etc...This function expects the same parameters of the function “readPublicData” and additionally the below parameters:

ReadV2Fields: set it to true to read V2 data specified above(if the flag is set to true while the card inserted is V1 cards, the function throws an error)

ReadSignatureImage: only if the above parameter is true, this parameter determines to read the signature image or not.

ReadAddress: only if the **ReadV2Fields** parameter is true, this parameter determines to read the Home and Work address fields or not.

```
//method signature
// String ReadPublicDataEx()
ReadPublicDataEx();
```

Upon successfully calling the ReadPublicDataEx function, a web page can execute any of the following additional functions to retrieve a specific public data filed:

Function Name	Description
`GetOccupationCode();`	Returns the code of the occupation
`GetOccupationEnglish();`	Returns the occupation description of the card holder
`GetOccupationArabic();`	Returns the Arabic occupation description in UTF-8 format.
`GetOccupationField();`	Returns the occupation field of the card holder
`GetPlaceOfBirth();`	Returns the place of birth of the card holder
`GetPlaceOfBirth_Ar();`	Return the place of birth of the card holder in Arabic in UTF-8 format
`GetOccupationType();`	Returns the occupation type description of the card holder
`GetOccupationType_Ar();`	Returns the occupation type description in Arabic in UTF-8 format
`GetCompanyName();`	Returns the company name of the card holder

`GetCompanyName_Ar();`	Returns the company name of the card holder in Arabic in UTF-8 format
`GetPassportNumber();`	Returns the passport number of the card holder
`GetPassportType();`	Returns the passport type code
`GetPassportIssueDate();`	Returns the passport issue date in DD/MM/YYYY format.
`GetPassportExpiryDate();`	Returns the passport expiry date in DD/MM/YYYY format.
`GetPassportCountry();`	Returns the passport country code of the card holder.
`GetPassportCountryDesc();`	Returns the passport country description of the card holder
`GetPassportCountryDesc_Ar();`	Returns the passport country description of the card holder in Arabic in UTF-8 format
`GetSponsorUnifiedNumber();`	Returns the sponsor unified number of the card holder
`GetMotherFullName();`	Returns the mother full name of the card holder
`GetMotherFullName_Ar();`	Returns the mother full name of the card holder in Arabic in UTF-8 format
`GetDegreeDesc();`	Returns the degree description of the card holder.
`GetDegreeDesc_Ar();`	Returns the degree description of the card holder in Arabic in UTF-8 format
`GetFieldOfStudy_Code();`	Returns the field of study code of the card holder
`GetFieldOfStudy();`	Returns the fields of the study of the card holder.
`GetFieldOfStudy_Ar();`	Returns the fields of the study of the card holder in Arabic in UTF-8 format.
`GetPlaceOfStudy();`	Returns the place of study of the card holder

`GetPlaceOfStudy_Ar();`	Returns the place of study of the card holder in Arabic in UTF-8 format
`GetGraduationDate();`	Returns the graduation date of the card holder
`GetQualificationLevel();`	Return the qualification level of the card holder
`GetQualificationLevelDesc();`	Returns the qualification level description of the card holder
`GetQualificationLevelDesc_Ar();`	Returns the qualification level description of the card holder in Arabic in UTF-8 format
`GetHolderSignatureImage();`	Returns the signature image in binary format of the card holder
`GetHomeAddress_EmirateDesc();`	Returns the home address emirate description of the card holder
`GetHomeAddress_EmirateDesc_Ar();`	Returns the home address emirate description of the card holder in Arabic in UTF-8 format.
`GetHomeAddress_CityDesc();`	Returns the home address city description of the card holder
`GetHomeAddress_CityDesc_Ar();`	Returns the home address city description of the card holder in Arabic in UTF-8 format
`GetHomeAddress_Street();`	Returns the home address street of the card holder
`GetHomeAddress_Street_Ar();`	Returns the home address street of the card holder in Arabic in UTF-8 format
`GetHomeAddress_AreaDesc();`	Returns home address area description of the card holder
`GetHomeAddress_AreaDesc_Ar();`	Returns home address area description of the card holder in Arabic in UTF-8 format
`GetHomeAddress_Building();`	Returns home address building of the card holder
`GetHomeAddress_Building_Ar();`	Returns home address building of the card holder in Arabic in UTF-8

	format
`GetHomeAddress_MobilePhoneNumber();`	Returns the home address mobile phone number of the card holder
`GetHomeAddress_ResidentPhoneNumber();`	Returns the home address residence phone number of the card holder
`GetHomeAddress_Email();`	Returns the home address Email address of the card holder
`GetHomeAddress_AddressTypeCode();`	Returns the home address type code of the card holder
`GetHomeAddress_LocationCode();`	Returns the home address location code of the card holder
`GetHomeAddress_EmirateCode();`	Returns the home address emirate code of the card holder
`GetHomeAddress_CityCode();`	Returns the home address city code of the card holder
`GetHomeAddress_POBox();`	Returns the home address PO box number of the card holder
`GetHomeAddress_AreaCode();`	Returns the home address area code of the card holder
`GetHomeAddress_FlatNumber();`	Returns the home address flat number
`GetWorkAddress_EmirateDesc();`	Returns the work address emirate description of the card holder
`GetWorkAddress_EmirateDesc_Ar();`	Returns the work address emirate description of the card holder in Arabic in UTF-8 format
`GetWorkAddress_CityDesc();`	Returns the work address city description of the card holder
`GetWorkAddress_CityDesc_Ar();`	Returns the work address city description of the card holder in Arabic in UTF-8 format
`GetWorkAddress_Street();`	Returns the work address street of the card holder
`GetWorkAddress_Street_Ar();`	Returns the work address street of the card holder in Arabic in UTF-8

	format
`GetWorkAddress_AreaDesc();`	Returns the work address area description of the card holder
`GetWorkAddress_AreaDesc_Ar();`	Returns the work address area description of the card holder in Arabic in UTF-8 format
`GetWorkAddress_Building();`	Returns the work address building of the card holder
`GetWorkAddress_Building_Ar();`	Returns the work address building of the card holder in Arabic in UTF-8 format
`GetWorkAddress_MobilePhoneNumber();`	Returns the work address mobile phone number of the card holder
`GetWorkAddress_LandPhoneNumber();`	Returns the work address land phone number of the card holder
`GetWorkAddress_Email();`	Returns the work address email address of the card holder
`GetWorkAddress_AddressTypeCode();`	Returns the work address type code of the card holder
`GetWorkAddress_LocationCode();`	Returns the work address location code of the card holder
`GetWorkAddress_EmirateCode();`	Returns the work address emirate code of the card holder
`GetWorkAddress_CityCode();`	Returns the work address city code of the card holder
`GetWorkAddress_POBox();`	Returns the work address PO box number if the card holder
`GetWorkAddress_AreaCode();`	Returns the work address area code of the card holder
`GetWorkAddress_CompanyName();`	Returns the work address company name of the card holder
`GetWorkAddress_CompanyName_Ar();`	Returns the work address company name of the card holder in Arabic in UTF-8 format

6.5.5 Read Public Data Contactless (MRZ Fields are entered manually)

The toolkit web components supports reading the card holder public data in contactless mode, however reading the public data in contactless mode is protected by Basic Access Control (BAC).

The function ReadPublicDataContactless generates MRZ input required for deriving the BAC keys, MRZ input is based on the CardNumber, DateOfBirth, and ExpiryDate that are accepted as parameters to this function.

Once BAC keys are diversified, Toolkit establishes secure messaging with the card to read the public data.

This function expects the same parameters of the function “readPublicDataEx”. Upon successfully calling the this function, a web page can execute any of public data filed getter functions that are used with “readPublicDataEx” to retrieve the data fields read in contactless mode.

Note: These functions work only when the component is initialized with InitializeContactless() function.

Methods signature

```
//methods signature// String ReadPublicDataContactless()
ReadPublicDataContactless(CardNumber, DateOfBirth, ExpiryDate, refresh, readPhotography, readNonModifiableData, readModifiableData, signatureValidation);
```

6.5.6 Read Public Data Contactless (with MRZ Reader)

The toolkit web components supports reading the card holder public data in contactless mode, however reading the public data in contactless mode is protected by Basic Access Control (BAC).

The function ReadPublicDataContactlessWithMRZData expects the MRZ lines read by MRZ reader as an input that is used to diversify BAC keys.

Note: new line and cartridge return characters must be removed from the MRZ text returned from an MRZ reader before passing it to the function ReadPublicDataContactlessWithMRZData

Once BAC keys are diversified, the toolkit establishes secure messaging with the card to read the public data.

This function expects the same parameters of the function “readPublicDataEx”. Upon successfully calling the this function, a web page can execute any of public data filed getter functions that are used with “readPublicDataEx” to retrieve the data fields read in contactless mode.

Note: These functions work only when the component is initialized with InitializeContactless() function.

Methods signature

```
//methods signature
// String ReadPublicDataContactlessWithMRZData()
ReadPublicDataContactlessWithMRZData(MRZData, refresh,
readPhotography, readNonModifiableData, readModifiableData,
signatureValidation);
```

6.5.7 Checking Card Genuine

As explained in section 5.11, there are two functions are available in the Toolkit to check the card validity, they are CheckCardGenuine and CheckCardGenuineEx.

Both functions do not expect parameters however the JavaScript definition in the js file does rely on a variable called “RemoteSM_Address” where the URL of EIDA secure messaging web service is specified.

Both functions returns empty string if the card is genuine otherwise an error occurred and error code returned.

6.5.8 Biometric functions

This section provides the list of Biometric functions that are available in the Web component.

6.5.8.1 GetNumberOfAvailableFingerprints

This function corresponds to the function “`readBiometricInfoTemplates`” described in section 5.8.1. When this function is invoked, the Toolkit caches the fingerprint template information into the memory.

This function returns the number of fingerprints stored on the card

6.5.8.2 GetFingerIndex

Only after the function “`readBiometricInfoTemplates`” is invoked then the function GetFingerIndex can be invoked to retrieve the index of any of the fingerprint templates stored on the card, the function expects the following parameter:

fingerIndex: 1 for the fingerprint stored on the card, 2 for the second

The function returns the fingerprint index, then the function GetFingerIndexDisplayName can be used to retrieve description of the fingerprint index.

6.5.8.3 Capture Image

This function captures the fingerprint image from a fingerprint sensor and checks for the quality of the captured image. If the quality of the image is good enough, then this function returns the captured image to the calling application.

please refer to section **Error! Reference source not found.** for more details about using fingerpeint sensors with the toolkit.

This function expects the following parameters:

Sensor ID : An ID indicating which sensor will be used for capture, please refer to section **Error! Reference source not found.** for more details about the usage of this paramter.

fingerIndex: the index of the fingerprint to be captured

Time_Out: the timeout of waiting for capturing fingerprint from the sensor

This function returns base64 string if succeeds otherwise error code is returned.

6.5.8.4 ConvertImage

This function computes the fingerprint minutiae from the raw image captured using the above function . the function expects the following parameters:

imageBase64 : the Raw image to be converted

width : the width of the image to be converted

length: the length of the image to be converted

the function returns Base64 string if successed otherwise error code is returned

6.5.8.5 CaptureAndConvert

This function captures the fingerprint using Sagem MSO 1350 sensor and converts the **captured image to template and then returns in base64 format. This function expects the** following parameters:

fingerIndex: the index of the fingerprint to be captured

seconds_TimeOut: the timeout of waiting for capturing fingerprint from the sensor

This function returns base64 string if succeeds otherwise error code is returned.

6.5.8.6 MatchOffCard

This function corresponds to the function MatchOffCard specified in section 5.12.4 where EIDA SM module is used remotely via the web service, the function expects the following parameters:

address: the URL of the web service

fingerIndex: the index of the captured fingerprint

fingerprint: the base64 string returned from the function CaptureAndConvert or the function Convert

This function returns empty string if it the matching is successful otherwise error coder is returned.

6.5.9 PKI Functions

This section provides the list of PKI functions that are available in the Web component and with brief details.

6.5.9.1 SignData

This function corresponds to the function specified in section 5.14 where data is signed using the 'Signing certificate', the function expects the following parameters:

pin: string containing the card PIN

data: the data to be signed in HEX format

The function returns signed data in hexadecimal string format.

6.5.9.2 Authenticate

This function corresponds to the function specified in section 5.13 where data is signed using the 'authentication certificate'. This function expects the following parameters:

pin: string containing the card PIN

data: the data to be signed in HEX format

The function returns signed data in hexadecimal string format.

6.5.9.3 AuthenticateWithPinCached

This function corresponds to the function specified in section 5.13 where data is signed using the 'authentication certificate' with PIN caching. This function expects the following parameters:

data: the data to be signed in HEX format

The function returns signed data in hexadecimal string format.

6.5.9.4 ReadSignCertificate

This function returns Sign Certificate in hexadecimal string format.

6.5.9.5 ReadAuthCertificate

This function returns Auth Certificate in hexadecimal string format.

6.6 Toolkit Web components JavaScript functions

The webcomponents are completely independent from presenting the data in the User Interface. The Toolkit webcomponents returns the code of data as they are stored on the card and do not display them. Mapping the returned code and displaying card data is the responsibility of the HTML page that contains the applet or the ActiveX. This design ensures the separation of the data from the UI and allows the web-designers to modify the HTML page easily without having to modify the applet (which needs a developer and not a web designer).

The benefits of this approach are as follows:

- It is possible to send the data extracted to the server (enrolment server for example) without modifying the applet. This can be done by filling hidden inputs in HTML page then submit the page. So that the applet or the ActiveX is independent from the application that receives the card data.
- It is possible to integrate the Toolkit web components with another web component placed on the same page.
- It is possible to change the look and feel of the page completely without having to modify the applet or the ActiveX.

To facilitate displaying card data on the HTML page, the web components functions described previously are wrapped by a number of javascript functions. The names and signatures of javascript functions are the same as the names and signatures of the functions exposed from the web-components except the javascript functions that allow the interaction and displaying public data.

For example, the GetOccupation() javascript function allows displaying the occupation of the card holder in 2 steps:

- Invoke the webcomponent function GetOccupation() that returns the code of the occupation.
- Invoke GetOccupationDisplayName() function that returns the occupation in text.

7 Toolkit Zero footprint Web-Components

The zero footprint web components do not have dependencies on the Toolkit kernel or any other Toolkit components. They provide access to a limited set of Toolkit functions including Read public data and PKI functions. This section provides samples on how to use those components both in ActiveX and Java Applets.

As part of the installation, the Toolkit comes with the WebComponents folder which has the below subfolders.

- "ActiveX" Folder:
    - EIDA_ZF_ActiveX.CAB: is the ActiveX version of the zero footprint components
- "Java Applets" folder:
    - ZFApplet.jar: is the Applet version of the zero footprint components

Zero footprint components supports the below functions only.

- Read Public Data raw files: Reads all raw files from the public data area of the ID applet on EIDA ID card.
- PKI functions
    - Sign Data: signs data with the SIGN key
    - Sign Challenge: signs a challenge with AUTH key
    - Read Authentication certificate: reads the AUTH certificate from card
    - Read Signature certificate: reads the SIGN certificate from card

7.1 Steps to embed EIDA IDCard Zero footprint Applet on a web page

Add the below code to embed the Applet in a web page.

```
<applet id="ZFComponent" alt="This browser does not support Applets."
    code="emiratesid.jio.webcomponents.ZFApplet" archive="ZFApplet.jar"
    width="0" height="0">
    This browser does not support Applets.

</applet>
```

- ID attribute refers to communicate with the Applet from javascript.
- CODE attribute represents the main class of the applet
- ARCHIVE attribute is the reference to the Java archive file (JAR) that contains the applet implementations.

- WIDTH and the HEIGHT attributes are used to set the dimension of the applet. As the applet doesn't display anything, the width and the height should be set to zero.

7.2 Steps to embed the Zero Footprint ActiveX on a web page

The below code provides steps to embed the Public Data ActiveX in a web page.

```
<object id="ZFComponent" width="0" height="0"
      classid="CLSID:31E5882A-A2EA-4B39-B76D-C32B49F5317B"
      codebase="EIDA_ZF_ActiveX.CAB">
       ActiveX is not supported by this browser, please use Internet Explorer
</object>
```

- ID attribute refers to communicate with the ActiveX from Javascript.
- CLASSID attribute is the reference to the ActiveX by its GUID (Global Unique Identifier).
- WIDTH and the HEIGHT attributes are used to set the dimension of the applet. As the ActiveX doesn't display anything, the width and the height should be set to zero.

7.3 Communication with the Web page

Only after embedding and referencing the relevant Toolkit web components in the page as explained above, developer can invoke any of the Toolkit functions via JavaScript.

Calling the JavaScript functions in the web page will make the web page code transparent from the Toolkit web component type, i.e. a developer might decide to replace the ActiveX with the applet at any time without changing the code in the web page.

The below sections describes the JavaScript functions corresponds to the Toolkit functions.

7.3.1 Initialize

This function initializes the web component by discovering the list of PC/SC readers connected and selects the first reader with ID card inserted.

- If successful, this function will return an empty String.
- Otherwise an error code will be returned.

```
//method signature
// String Initialize()
var result = Initialize();
```

7.3.2 Reading the public data

This function reads cardholder public data raw files from the card and caches into the Applet / ActiveX. This function doesn't parse public data, and doesn't validate the signature on the read files.

- If successful, this function returns an empty String.
- Otherwise an error code will be returned.

```
//method signature
// String ReadPublicData(boolean, boolean, boolean, boolean,
boolean, boolean)

ReadPublicData(ReadPhotography, ReadNonModifiableData,
ReadModifiableData, ReadSingatureImage,
                  ReadAddress, ReadRootCertificate);
```

Note that the parameter flags in the above signature to enable or disable reading specific data file.

Upon successfully calling the ReadPublicData function, the following functions are exposed by the web-component.

Function Name	Description
GetEF_IDN_CN ()	Returns IDN CN file in base64 format.
GetEF_NonModifiableData ()	Returns the Non Modifiable data file in base64 format.
GetEF_ModifiableData ()	Returns the Modifiable data file in base64 format.
GetEF_Photography ()	Returns the Personal photography file in base64 format.
GetEF_HolderSignatureImage ()	Returns the card holder signature image file in base64 format.
GetEF_HomeAddressData ()	Returns the home address data file in base64 format.
GetEF_WorkAddressData ()	Returns the work address data file in base64 format.
GetEF_RootCertificate ()	Returns the root certificate file in base64 format.

In a typical web scenario, the back end business application would receive the data file as read by the Zero-footprint component. It would then have to validate the digital signature on

the file(s) and parse these files in order to process the extracted data. The Toolkit is delivered with components that fit this purpose.

Refer to Appendix E for more information about parsing public data files and validating signatures on these files.

7.3.3 PKI Functions

This section provides the list of PKI functions that are available in the zero footprint web components and with brief description of those functions.

7.3.3.1 Sign Data

This function signs data using the 'Signing certificate', the function expects the following parameters:

pin: string containing the card PIN

data: the data to be signed in base64 format

This function returns PKCS1 signature in base64 string format.

7.3.3.2 Sign Challenge

This function signs data using the 'authentication certificate'. This function expects the following parameters:

pin: string containing the card PIN

data: the data to be signed in base64 format

This function returns PKCS1 signature in base64 string format.

7.3.3.3 GetSignCertificate

This function reads SIGN Certificate and returns the certificate in base64 string format.

7.3.3.4 GetAuthCertificate

This function reads AUTH Certificate and returns the certificate in base64 string format.

7.3.4 Samples

Toolkit installation provides web application samples to demonstrate the usage of the zero footprint components in both Java, and .Net platforms.

Both samples utilises the use of the components implemented in Applet and ActiveX and demonstrates the available functions as well, including server side functions for parsing public data files and validating signatures on these files.

Sample web application also demonstrates signing data and challenges with server side signature verification using Toolkit APIs with 3 the options to validate the certificate namely OCSP, CRL and CDP.

s Samples can be found in the “Samples/Website” folder in the toolkit installation folder.

Refer to Appendix E for more information about public data parser component.

Appendix A – Secure Messaging configuration file (sm.cfg)

The configuration file sm.cfg is required if the application requires local Secure Messaging (SM).

SM_Modules section contains the configuration information for the 3 applets ID, PKI, and MOC, to choose which SM module to be used with each applet

“1” represents the Sagem SAM

“2” represents SafeNet Luna HSM

“3” represents Software SAM and it is only applicable for EIDA and not applicable for end customers of ID Toolkit.

Other values for custom SM, if any other value is present, the implementation dll should be configured, as in the sample below the value “5”

```
[SM_Modules]
####### SAGEM_SAM 1, SAFENET_LUNA_HSM 2, LOGICA_SOFTWARE_HSM=3
ID_SM_Name=1
PKI_SM_Name=5
MOC_SM_Name=2

5=Another_SM.dll

[Data Signing Certificates]
PATH=C:\Program Files\EIDA Toolkit\Libs\SigningCerts

[SAM]
PIN=0123
ATR1=3B 7F 11 00 00 80 41 00 57 4A 2D 49 44 4D 36 34 83 7F 90 00
ATR2=
ATR3=

[HSM]
PIN=1234567

[UAE Card]
NUMBER=6
ATR1=3B 6A 00 00 80 65 A2 01 30 01 3D 72 D6 41
ATR2=3B 6A 00 00 80 65 A2 01 31 01 3D 72 D6 41
ATR3=3B 7A 95 00 00 80 65 A2 01 30 01 3D 72 D6 41
ATR4=3B 7A 95 00 00 80 65 A2 01 31 01 3D 72 D6 41
ATR5=3B 8A 80 01 80 65 A2 01 31 01 3D 72 D6 41 A5
ATR6=3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A
[Test Card]
NUMBER=6
ATR1=3B 6A 00 00 80 65 A2 01 30 01 3D 72 D6 41
ATR2=3B 6A 00 00 80 65 A2 01 31 01 3D 72 D6 41
ATR3=3B 7A 95 00 00 80 65 A2 01 30 01 3D 72 D6 41
ATR4=3B 7A 95 00 00 80 65 A2 01 31 01 3D 72 D6 41
ATR5=3B 8A 80 01 80 65 A2 01 31 01 3D 72 D6 41 A5
ATR6=3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A
```

Data Signing Certificates section has PATH attribute which holds the path of the signing certificates folder. The certificates stored in this folder are used by the Toolkit kernel to verify cardholder data signatures.

SAM section is used to provide the SAM card PIN number and three different ATR values if the used SAM cards have different ATR

HSM section is configured to provide the HSM password or PIN number

UAE Card section used to provide the unique ATR of UAE live ID cards

The attribute **Number** states the number of ATRs follows in this section for the toolkit to read

Test Card section used to provide the unique ATR of UAE test ID cards

The attribute **Number** states the number of ATRs follows in this section for the toolkit to read

Appendix B – Toolkit Business Sequences

This appendix describes the sequence of API calls a developer needs to undertake in order to have access to the Toolkit main business functions.

Reading Card Holder Public Data

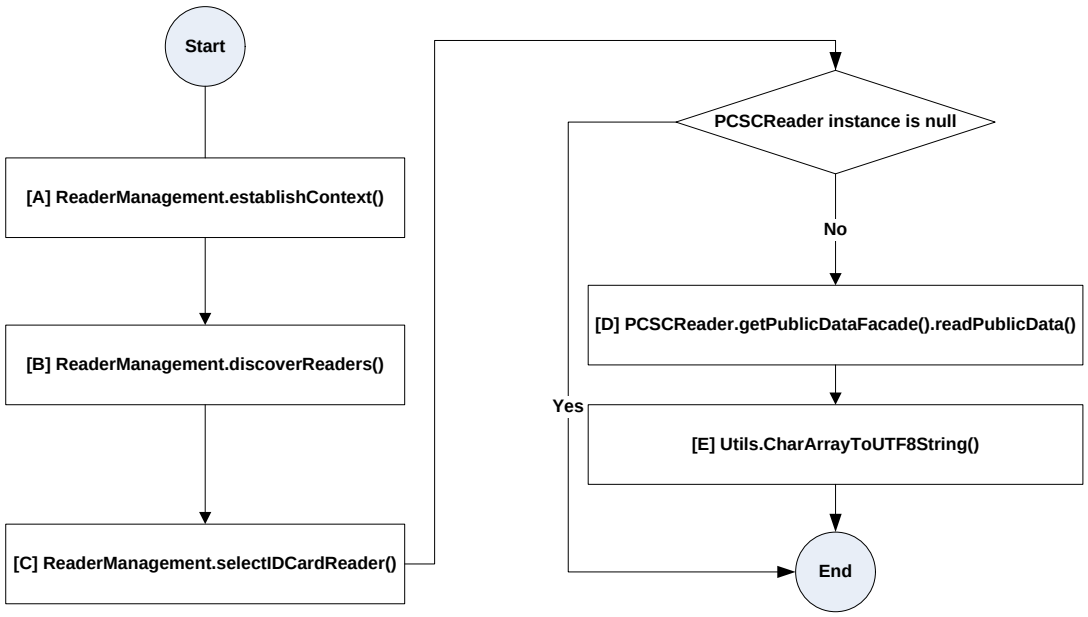


Figure 1 Read Public Data

[A] ReaderManagement.establishContext()

The very first step of each process is to establish a context with the windows PC/SC smart card library by using an instance of the ReaderManagement class.

[B] ReaderManagement.discoverReaders()

In order to access any of the cards connected to the machine, the caller application needs to discover the list of available readers using the function discoverReaders defined in the ReaderManagement class.

[C] ReaderManagement.selectIDCardReader()

If the size of the list of discovered readers is not zero means that there is at least one smart card reader connected to the machine. The function selectIDCardReader defined in ReaderManagement class allows the caller application to select a reader by name, type, or index. If the selection filter is valid, then the function will return handle to the selected reader represented in an instance of the class PCSCReader otherwise it will return null.

[D] PCSCReader.getPublicDataFacade().readPublicData()

The last step of the process executed when the reader handle retrieved from the previous step as PCSCReader object that will be used as follows:

- Retrieve an instance of the class PublicDataFacade using the function getPublicDataFacade.
- Call the function readPublicData to read public data.

NOTE: the PCSCReader instance used in this step is the same one created in step C

[E] Utils.CharArrayToUTF8String()

This function converts the binary data retrieved from the card to string representation. The CharArrayToStringDate function must be used for Date fields.

Checking Card Genuine with Secure Messaging API (Local Mode)

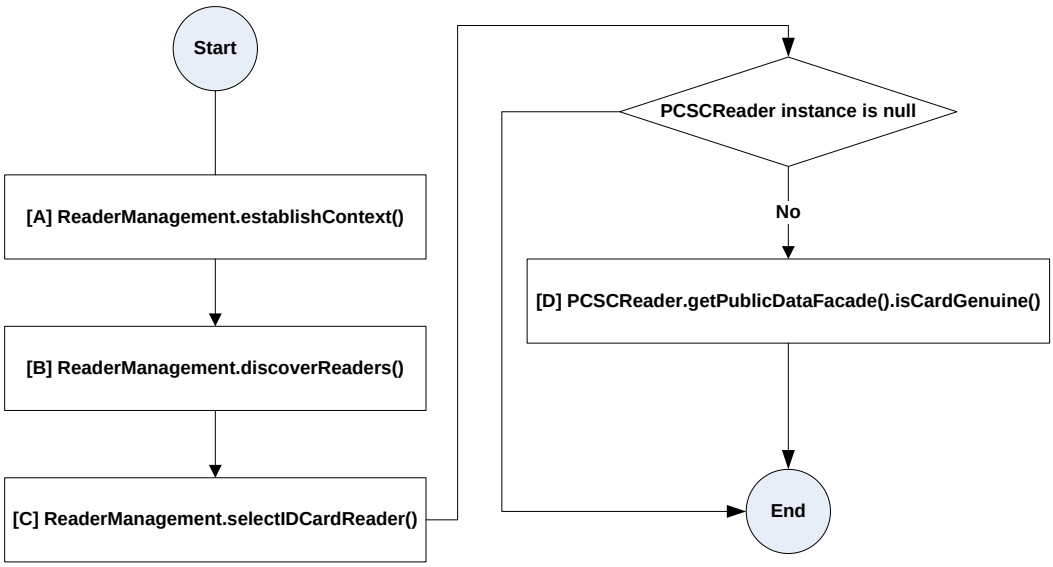


Figure 2 Check Card Genuine (Local)

Steps A, B and C are as mentioned under the read public data workflow.

[D] PCSCReader.getPublicDataFacade().isCardGenuine()
Calling the function isCardGenuine will execute a sequence of cryptographic operations between the card and the local security module to validate whether the card is a genuine card personalized by EDIA or not.

NOTE: before using this function, the configuration file sm.cfg shall be configured as specified in Appendix A according to the availability of secure messaging modules (HSM, SAM or multiple SAM, or Software SAM).

The PCSCReader instance used in this step is the same one created in step C.

Checking Card Genuine with Secure Messaging API (Remote Mode)

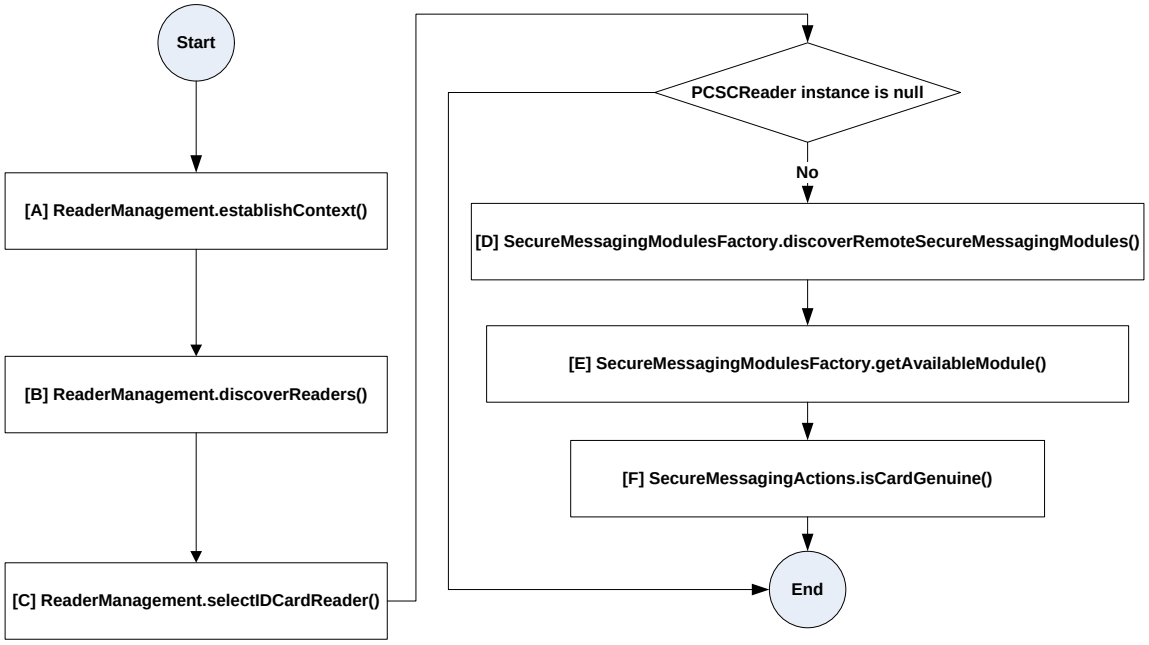


Figure 3 Check Card Genuine(Remote)

Steps A, B and C are as mentioned under the read public data workflow.

[D] SecureMessagingModulesFactory.discoverRemoteSecureMessagingModules()
The function discoverRemoteSecureMessagingModules is called to get the list of secure messaging modules configured on the server. This function takes as parameter the URL of the secure messaging server.

[E] SecureMessagingModulesFactory.getAvailableModule()
The same SecureMessagingModulesFactory instance created in the previous step is used to call the function getAvailableModule to get the available secure module of specific type.

[F] SecureMessagingActions.isCardGenuine()

1. Create a new instance of the class SecureMessagingTransaction using the returned SecureMessagingModuleInterface instance -returned in the previous step- as a parameter for SecureMessagingTransaction constructer.
2. Create a new instance of the class SecureMessagingActions using the reader instance created in step C as a parameter for SecureMessagingActions constructer.
3. Call the function isCardGenuine defined in the class SecureMessagingActions with specifying the SecureMessagingTransaction instance created at step 1 as parameter to this function.

Biometric Match-Off-Card

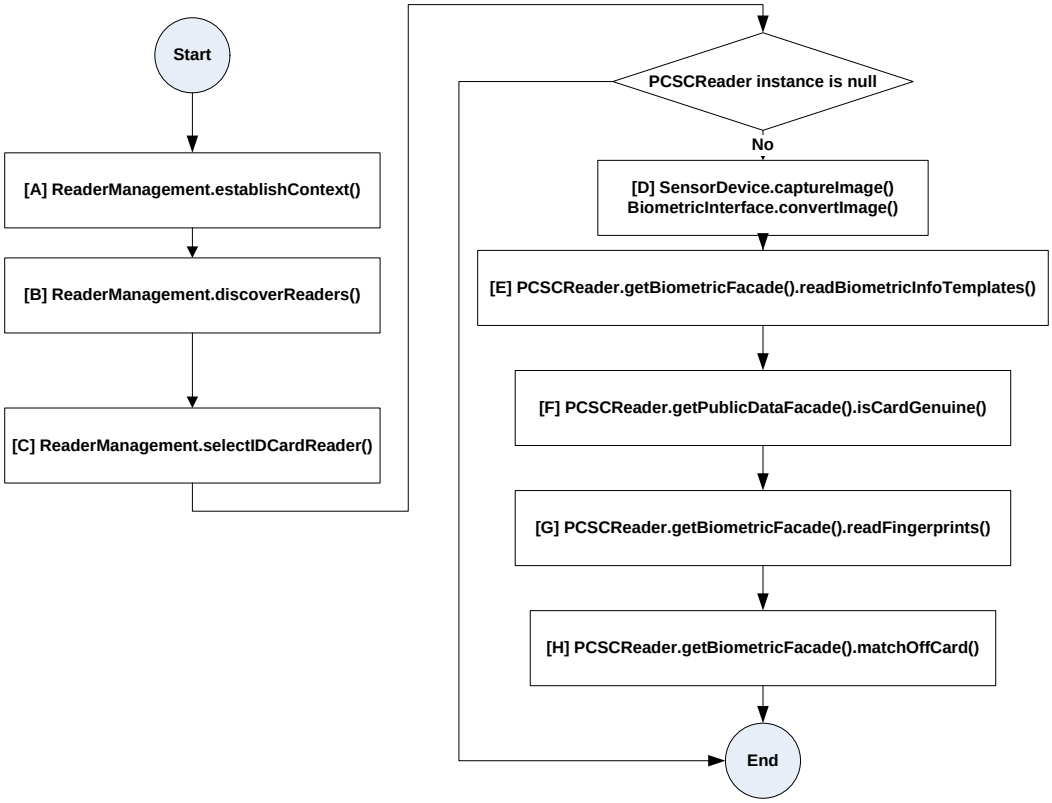


Figure 4 Match off Card

Steps A, B and C are as mentioned under the read public data workflow.

[D] SensorDevice.captureImage() and BiometricInterface.convertImage()
This function is called with the Finger Index and Template format type as parameters. If the fingerprint was captured successfully then an instance of the class FTP_Image will be returned.

The BiometricInterface.convertImage() takes FTP_Image object as parameter and creates an FTP_Template object containing the fingerprint minutiae.

[E] PCSCReader.getBiometricFacade().readBiometricInfoTemplates()
This function reads the information associated with the card fingerprints

NOTE: the PCSCReader instance used in this step is the same one created in step C

[F] isCardGenuine()
For match-off card, the fingerprint templates need to be read from the card since the matching will be executed outside the card, the card will not allow reading the templates unless there is a secure messaging session is established. The function isCardGenuine can be used to initiate secure messaging either locally or remotely.

NOTE: the PCSCReader instance used in this step is the same one created in step C

[G] PCSCReader.getBiometricFacade().readFingerprints()
readFingerprints function reads the fingerprint templates stored on the card and cache it in order to be used for Off-card matching.

NOTE: the PCSCReader instance used in this step is the same one created in step C

[H] PCSCReader.getBiometricFacade().matchOffCard()
The last step in the matching process is to call the matchOffCard function defined in the BiometricFacade by passing FTP_Template object returned from step [D] as parameter. The function will return a matching score..

NOTE: the PCSCReader instance used in this step is the same one created in step C

Sign Operation

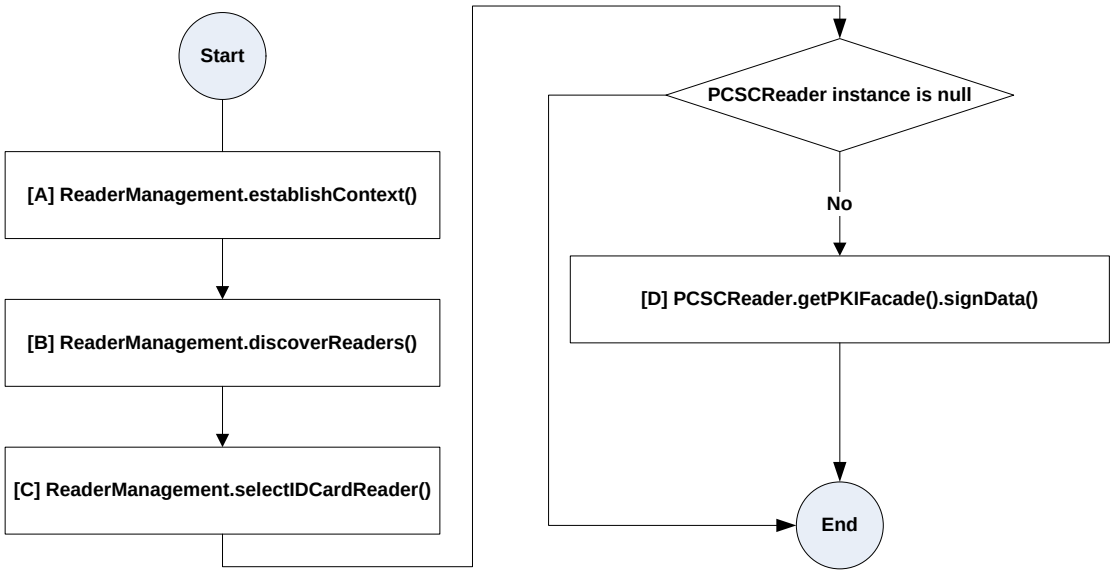


Figure 5 Sign Operation

Steps A, B and C are as mentioned under the read public data workflow.

[D] PCSCReader.getPKIFacade().signData()

The last step of the process is executed when the reader handle retrieved from the previous step as PCSCReader object:

- Retrieve an instance of the class PKIFacade using the function getPKIFacade.
- Call the function signData defined in the class PKIFacade that will sign the input data suing signing private key stored on the card.

If the function succeeds then it will return PKCS#1 signature.

Appendix C – Fingerprint Sensor Interface Specifications

In order to provide flexibility on supporting additional fingerprint sensors, flexibility in EIDA Toolkit design allows a developer to integrate other new standard sensors with minimal implementation effort. However the desired fingerprint sensor must be capable of scanning an image with the following minimum quality requirements:

- Image resolution 500 DPI
- Image width from 150 to 1000 pixels
- Image height from 166 to 1000 pixels
- Image format BMP/RAW

Integrating fingerprint sensor with the toolkit is based on the following aspects:

1. Defining a standard interface that shall be implemented as DLL for any new sensor.

2. The toolkit uses a configuration file to recognize additional sensors where the DLL name is specified with the Sensor ID.

3. The toolkit loads the DLL dynamically at runtime, using the windows API function “LoadLibrary” so the interface DLL and its dependencies has to be located either in the current application folder or in any other path that is included in the windows PATH environment variable. The deployment of the interface DLL, its dependencies and sensor driver is totally independent on the toolkit deployment as it can be deployed separately at any time then the configuration file shall be updated accordingly.

Interface Specifications

The sensor interface DLL has to implement the below specified functions with the same function prototypes:

1. Capture (Mandatory)

This function must execute the capture process with the sensor within a given timeout and return back the captured image in RAW format, if the function success it returns 0.

Function Prototype

int Capture(int TimeOut, `FTP_Image* Image`);

Parameters

TimeOut	[in] a timeout of the waiting for capture in seconds
Image	[out] a pointer to the an instance of the FTP_Image structure where the captured image will be stored in RAW format, the DLL shall allocate the memory required to store the image

Pixels, the memory will be freed later using the function FreeImageMemory.

Definition of the FTP_Imagec structure

```
typedef struct FTP_Image {
      LPBYTE Pixels;
      int Width;
      int Height;
      int FingerIndex;
      int Quality;//optional
}FTP_Image;
```

Return

0	Successful
-67	E_BIOMETRICS_SDK_INIT *Error initializing sensor SDK/DLL*
-62	E_BIOMETRICS_NO_DEVICE *sensor not found*
-65	E_BIOMETRICS_CAPTURE *Error capturing image from sensor*

2. FreeImageMemory (Mandatory)

Upon completion of processing the fingerprint, this function must be called to free any allocated memory or other resources.

Function Prototype

```
void FreeImageMemory(LPBYTE Mem);
```

Parameters

Mem	[in] a pointer to the image Pixels buffer returned within the FTP_Image structure by the function capture or a pointer to the Fingerprint buffer returned within the FTP_Template structure by the Capture_Convert function.

3. Capture_Convert (Optional)

If the sensor has a built-in functionality allows direct conversion of the captured image to any of the templates supported by the toolkit matching function (ISO_19794_CS or DINV_66400) then this function can be optionally implemented to do the capture and the conversion in one go.

Function Prototype

```
int Capture_Convert(int TimeOut, FTP_Template* Template1);
```

Parameters

TimeOut	[in] a timeout of the waiting for capture in seconds

Template1	[out] a pointer to FTP_Template structure that carries the template data, here is the definition of FTP_Template structure:

```
struct FTP_Template {
int FingerIndex;
int Format;//ISO(0), DINV(1)
BYTEArray Fingerprint;
  }FTP_Template;
```

Return

0	Successful
-67	E_BIOMETRICS_SDK_INIT *Error initializing sensor SDK/DLL*
-62	E_BIOMETRICS_NO_DEVICE *sensor not found*
-100	E_NULL_ARGUMENT *if the parameter Template1 is passed with null value*
-63	E_BIOMETRICS_CAPTURE_CONVERT *error occurred while capturing*
-64	E_BIOMETRICS_NO_TEMPLATES *sensor doesn't support template type or template can't be retrieved*

Sensor configuration file

The sensors.cfg file lists the additional sensors to be recognized by the Toolkit class SensorDevice, each sensor has a single record in the file where the sensor ID and the interface DLL name are defined.

The sensors.cfg has only one section called "Sensors" where the sensors are listed as follows:

[ID]=[DLL Name]

ID: is a number representing the sensor ID, this number shall start from 1.

DLL Name: the name of the sensor interface DLL

Sample content of the sensors.cfg file

```
[Sensors]

1=FutronicPlugin.dll

2= DermalogPlugin.dll
```

Appendix D – Switch To Mifare Emulation Interface Specifications

In order to provide flexibility on supporting additional Mifare readers, a framework has been implemented to allow a developer to integrate new reader with minimal implementation effort. The toolkit integration approach is based on the following aspects:

1. A developer need to implement the Switch to Mifare command for the desired reader in a C++ DLL.
2. The implementation must following the defined standard interface for that purpose.
3. The Toolkit uses a configuration file to recognize additional readers' plugins where the plugin DLL name is specified with the Reader ID.
4. The Toolkit loads the DLL dynamically at runtime, using the windows API function "LoadLibrary" so the interface DLL and its dependencies has to be located either in the current application folder or in a specified path that is included in the windows PATH environment variable. The deployment of the interface DLL, its dependencies and drivers of the reader are independent from the Toolkit deployment as it can be deployed separately with the required configuration changes in the configuration file.

Interface Specifications

The reader interface DLL must implement the below function with the same function prototype as specified below:

SwitchToMifareEmulation (Mandatory)

This function should execute the switch command of the reader by passing valid PCSC context, card handle and reader name and it must return back a card handle after the reader is switched to Mifare emulation successfully. The function should returns 0 in case of failure to switch, otherwise valid card handle.

Function Prototype

```
_ULONG SwitchToMifareEmulation(_ULONG Context, _ULONG CardHandle,
LPCWSTR ReaderName);
```

Parameters

Context	[in] a valid PCSC context
CardHandle	[in] a valid handle to the current connected card reader.

Return

0	Switch failed
<positive number>	the new handle to MIFARE reader.

MIFARE configuration file

The mifare.cfg file lists the additional readers to be recognized by the Toolkit, each reader has a single record in the file where the reader ID and the interface DLL name are defined.

The mifare.cfg has only one section called “Readers” where the readers are listed as follows:

[ID]=[DLL Name]

ID: is a number representing the reader ID, this number shall start from 1.

DLL Name: the name of the sensor interface DLL

Sample content of the mifare.cfg file

```
[Readers]

1=HIDMifarePlugin.dll
```

Appendix E – Public Data Parser

To parse and process the raw public data and signature read by the zero footprint components, the below complementary components (Classes) are delivered with the Toolkit.

The Public Data parser Class is implemented for both Java and .Net platforms to do public data files parsing and signature validation. Both components use the same namespace/package to hold the main class “PublicDataParser”.

- `emiratesid.ae.publicdata`: for java
- `Emiratesid.AE.PublicData`: for .Net

The PublicDataParser Class can be used to parse and validate the signature for each public data file separately and then data can be extracted from the binary file and retrieved by its corresponding GetXXX method.

The PublicDataParser Class uses EIDA data signing certificates to validate signatures in each file. EIDA Data signing certificates are kept in the folder path containing these certificates can be passed to the class constructor to be used in signature verification.

PublicDataParser class initialization example

```
PublicDataParser p1 = new PublicDataParser(IDN_CN_Base64, certsPath);

// Or

PublicDataParser p2 = new PublicDataParser(IDN_CN_Base64);
```

The first constructor expects 2 parameters

- IDN CN file in base64 string format
- Path holds the EIDA data signing certificates as a string

The second constructor expects only the IDN CN and it uses the application current directory as the path for EIDA data signing certificates.

Parsing public data files

The PublicDataParser Class has one method to parse each data file. Each method takes a base64 string holding the file and returns a Boolean value indicating the signature validation result.

The below table has a summary of parsing methods.

Method Signature (Java)	Method Signature (.Net)
`boolean parseNonModifiableData()`	`bool ParseNonModifiableData()`
`boolean parseModifiableData(String)`	`bool ParseModifiableData(String)`

boolean parsePhotography(String)	bool ParsePhotography(String)
boolean parseSignatureImage(String)	bool ParseSignatureImage(String)
boolean parseHomeAddressData(String)	bool ParseHomeAddressData(String)
boolean parseWorkAddressData(String)	bool ParseWorkAddressData(String)
boolean parseRootCertificate(String)	bool ParseRootCertificate(String)

After a successful call for any of the parsing methods above, a set of attributes is filled with the parsed data. The below tables (relevant attributes are grouped) provides those attributes.

Non-Modifiable Data File
String getIdType()
byte[] getIssueDate()
byte[] getExpiryDate()
String getArabicTitle()
String getArabicFullName()
String getTitle()
String getFullName()
String getSex()
String getArabicNationality()
String getNationality()
byte[] getDateOfBirth()
String getArabicMotherFirstName()
String getMotherFirstName()
String getArabicPlaceOfBirth()
String getPlaceOfBirth()
byte[] getCardHolderData_SF3_Signature()

Modifiable Data
String getOccupation()

String getMaritalStatus()
String getFamilyID()
String getHusbandIDN()
String getSponsorType()
String getSponsorNumber()
String getSponsorName()
String getResidencyType()
String getResidencyNumber()
byte[] getResidencyExpiryDate()
String getArabicMotherFirstName()
String getMotherFirstName()
String getOccupationType_ar()
String getOccupationType()
byte[] getOccupationField()
String getCompanyName_ar()
String getCompanyName()
String getPassportNumber()
byte[] getPassportType()
String getPassportCountry()
String getPassportCountryDesc_ar()
String getPassportCountryDesc()
byte[] getPassportIssueDate()
byte[] getPassportExpiryDate()
byte[] getQualificationLevel()
String getQualificationLevelDesc_ar()
String getQualificationLevelDesc()
String getDegreeDesc_ar()
String getDegreeDesc()
byte[] getFieldOfStudyCode()

String getFieldOfStudy_ar()
String getFieldOfStudy()
String getPlaceOfStudy_ar()
String getPlaceOfStudy()
byte[] getDateOfGraduation()
String getMotherFullName_ar()
String getMotherFullName
String getCardHolderData_SF5_Signature()

Photography
byte[] getPhotography()
byte[] getPhotographySignature()

Signature Image
byte[] getHolderSignatureImage()
byte[] getHolderSignatureImageSignature()

Root Certificate (Java)	**Root Certificate (.Net)**
Certificate getRootCertificate()	X509Certificate2 GetRootCertificate()

Home Address
byte[] getHomeAddressTypeCode()
byte[] getHomeAddressLocationCode()
byte[] getHomeAddressEmirateCode()
String getHomeAddressEmirateDesc_ar()
String getHomeAddressEmirateDesc()
byte[] getHomeAddressCityCode()
String getHomeAddressCityDesc_ar()

String getHomeAddressCityDesc()
String getHomeAddressStreet_ar()
String getHomeAddressStreet()
String getHomeAddressPOBox()
byte[] getHomeAddressAreaCode()
String getHomeAddressAreaDesc_ar()
String getHomeAddressAreaDesc()
String getHomeAddressBuildingName_ar()
String getHomeAddressBuildingName()
String getHomeAddressFlatNo()
String getHomeAddressResidentPhoneNo()
String getHomeAddressMobilePhoneNo()
String getHomeAddressEmail()
byte[] getHomeAddress_Signature()

Work Address
byte[] getWorkAddressTypeCode()
byte[] getWorkAddressLocationCode()
String getWorkAddressCompanyName_ar()
String getWorkAddressCompanyName()
byte[] getWorkAddressEmirateCode()
String getWorkAddressEmirateDesc_ar()
String getWorkAddressEmirateDesc()
byte[] getWorkAddressCityCode()
String getWorkAddressCityDesc_ar()
String getWorkAddressCityDesc()
String getWorkAddressPOBox()
String getWorkAddressStreet_ar()
String getWorkAddressStreet()

byte[] getWorkAddressAreaCode()
String getWorkAddressAreaDesc_ar()
String getWorkAddressAreaDesc()
String getWorkAddressBuildingName_ar()
String getWorkAddressBuildingName()
String getWorkAddressLandPhoneNo()
String getWorkAddressMobilePhoneNo()
String getWorkAddressEmail()
byte[] getWorkAddress_Signature()

Appendix F – ID Box One MRZ scanner

This appendix describes a standalone class/component implemented to interface with the ID Box One MRZ scanner.

Application developers can use this Class / Component in order to read the ID Card MRZ data.

The ID Box One has no SDK to read MRZ data and exposes the scanner interface only to a serial port. The below component has been developed to send the read MRZ command to the ID Box One scanner serial port.

The class `IDBoxOneScanner` is provided for the 2 platforms as the rest of Toolkit APIs: Java, and .Net.

This class is in the package/namespace `emiratesid.samples.mrz` and exposes the below methods.

Method	Description
String[] getAvailableSerialPorts()	A static method retrieves all the available serial ports names.
boolean readMRZ(String comPortName, int timeout)	Send read MRZ command to the specified serial port name, returns true if MRZ data was read successfully, timeout parameter specifies timeout in milliseconds before breaking the connection with the serial port and abort reading.
String getMrz()	Returns the MRZ data that was read by readMRZ method

Sample Code (Java)

```
import emiratesid.samples.mrz;
//...
//...
//...
IDBoxOneScanner scanner = new IDBoxOneScanner();
try {
	boolean read = scanner.readMRZ("COM6", 5000);
	if(read)
		System.out.println(scanner.getMrz());
} catch (Exception e) {
	e.printStackTrace();
}
```