Security Classification:

هيئة
الإمارات
للهوية

EMIRATES
IDENTITY
AUTHORITY

EIDA Toolkit v2.5

Developer’s guide for C++

Document Details

Organization	Emirates Identity Authority (EIDA)
Document Title	Document Name
Date	10-04-2012
Doc Name / Ref	
Classification	(•) Public ( ) Internal ( ) Confidential ( ) Highly Confidential
Document Type	( ) Policy ( ) Procedure ( ) Form/Template ( ) Report (•) Other

Document History

Date	Version	Author	Comments
24-11-2011	1.0		Initial completed version
21-01-2012	1.0 for Toolkit 2.3		Released as part of Toolkit 2.3
21-03-2012	0.1 for Toolkit 2.4		Released as part of Toolkit 2.4
10-04-2012	1.0 for Toolkit 2.4		Final version
02-05-2012	1.0 for Toolkit 2.5		Released as part of Toolkit 2.5

Table of Contents

1 Introduction 6
2 Compatibility 7
3 Toolkit installation 8
4 Development Environment 9
5 EIDA ID card Toolkit C++ Functions 11
5.1 Get Last Error Status 11
5.2 Establishing Context 11
5.3 Closing Context 12
5.4 List Readers 13
5.5 Connect 14
5.6 Disconnect 15
5.7 Read Card ATR 16
5.8 Is UAE Card 17
5.9 Is UAE Test Card 18
5.10 Is UAE SAM 19
5.11 Is Contactless Reader 19
5.12 Get Card Version 20
5.13 Get Card Serial Number 21
5.14 Get Chip Serial Number 22
5.15 Get Issuer Serial Number 23
5.16 Get Issuer reference Number 24
5.17 Get CPLC0101 24
5.18 Get CPLC9F7F 25
5.19 Read Public Data 26
5.20 Read Public Data Ex 28
5.21 Read Public Data Contactless (MRZ Fields are entered manually) 29
5.22 Read Public Data Contactless (with MRZ Reader) 31
5.23 Read Family Book Data 33
5.24 Init SM 35
5.25 Is Card Genuine 35
5.26 Is Card Genuine Ex 36
5.27 Get MOC Biometric Information Template 37

5.28 Get MOC serial number 38
5.29 Get MOC Applet state 39
5.30 Get Max Failed Match 40
5.31 Get MOC Algorithm Version 41
5.32 Capture Image 42
5.33 Capture and Convert 43
5.34 Convert Image 44
5.35 Convert BMP Image 45
5.36 Match on-Card 46
5.37 Read Fingerprints 47
5.38 Match off-Card 48
5.39 Generate Card Cryptogram 51
5.40 Verify Ciphered PIN 52
5.41 Export PKI Certificates 53
5.42 Sign Data 54
5.43 Authenticate 55
5.44 Authenticate with PIN caching 57
5.45 Unblock PIN 58
5.46 Change PIN 59
5.47 Switch to Mifare Emulation 61
5.48 Is Mifare Emulation Active 62
5.49 Load Mifare Key 63
5.50 Read Mifare Binary Data 64
5.51 Update Mifare Binary Data 65

Definitions

Abbreviation	Description
API	Application Programming Interface
BIT	Biometric Information Template
DLL	Dynamic Link Library

EIDA	Emirates Identity Authority
MOC	Match On Card
HSM	Hardware Security Module
PIN	Personal Identification Number
SAM	Security Access Module
SDK	Software Development Kit
SM	Secure Messaging
VB	Visual Basic

1 Introduction

The EIDA Toolkit core library exports the necessary C++ API's to help developers to build applications around UAE ID Card.

This document is a guide for C++ developers to build advanced applications using EIDA Toolkit core Library.

Important pre-requisites:

1) Low level skills in C++ are mandatory to build applications using C++ Core library.
2) Knowledge and experience in smart card field is necessary.
3) Quality knowledge on EIDA smartcard

For Java and C# developers, please refer to the Toolkit Developer's guide for Java and C#.

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++

IDE and Compilers

- Microsoft Visual Studio 2005 (Or Express Edition)
- Microsoft Visual Studio 2008 (Or Express Edition)

3 Toolkit installation

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.

The Toolkit setup will automatically copy the below components to the Toolkit installation folder.

Component name	Physical file	Description
C++ Core API	UAE_IDCardLib.dll	Core components of the Toolkit.
	Wrappers, helper Dlls	
Header Files	UAE_IDCardLib.h ErrorCodes.h	The header files that contain the API functions declarations and error codes.
Lib File	UAE_IDCardLib.lib	Library file that is required to integrate the UAE_IDCardLib.dll with an application.

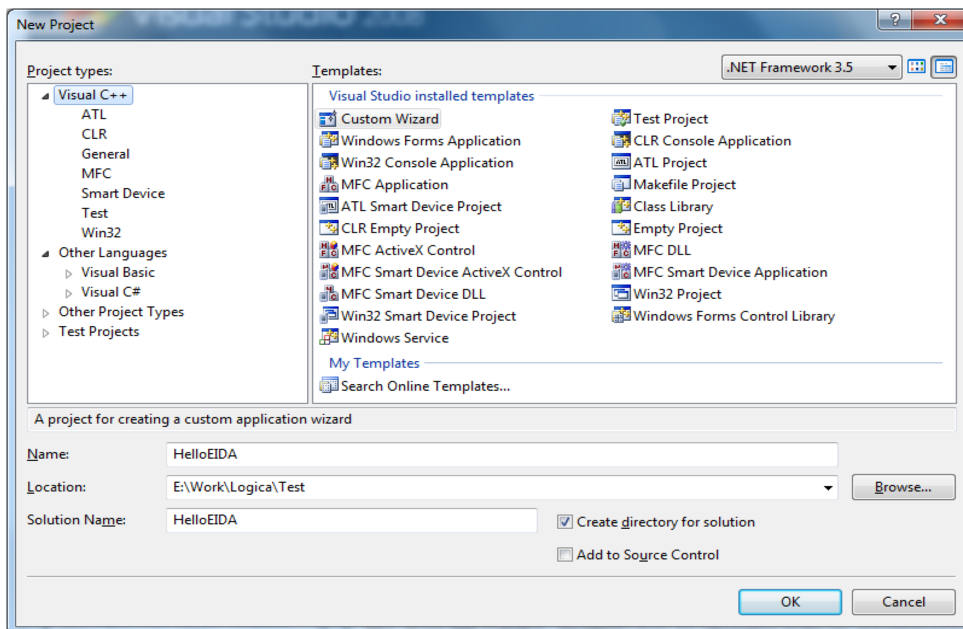
4 Development Environment

This section provides the step by step guidelines for creating VC++ project using Microsoft Visual Studio and steps to integrate the C++ APIs within the project.

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.

a) Run Microsoft Visual Studio, goto File -> New -> Project

b) Enter a project name, select Visual C++ as a project type



c) Select any of Visual C++ applications depends on the required application type

d) Select the project in project explorer

e) Go to Project -> Add Existing Item …

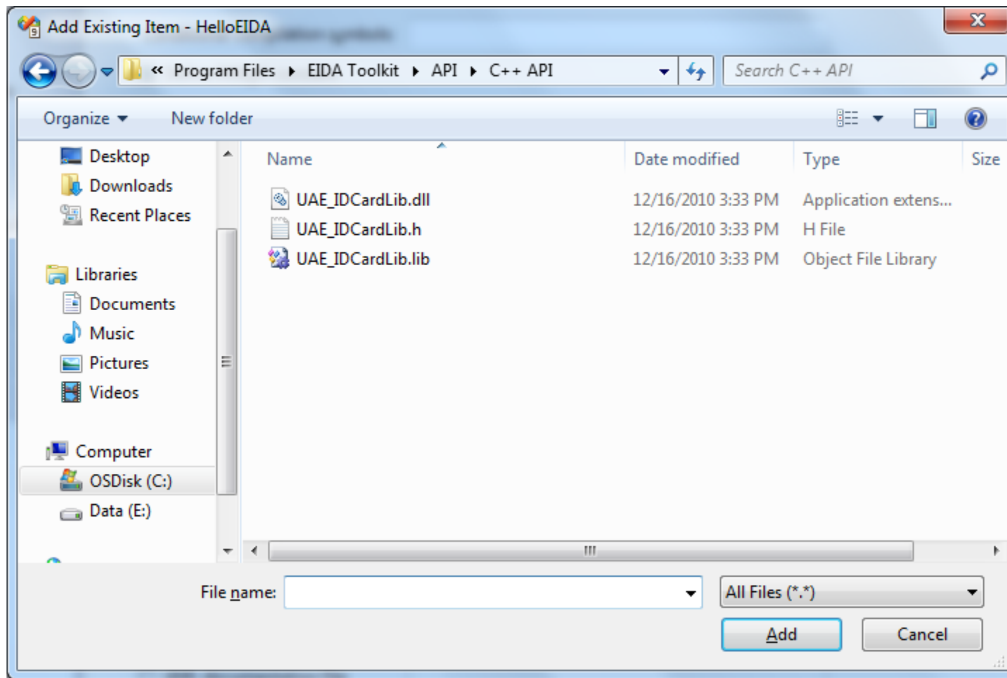
f) Navigate to the path where Toolkit SDK is installed

g) Open “API” folder

h) Open “C++ API” folder

i) Select “UAE_IDCardLib.h”, “ErrorCodes.h” and “UAE_IDCardLib.lib”

j) Click “Add”



Refer to the section 5 for various C++ functions available in the Toolkit to build your application(s).

5 EIDA ID card Toolkit C++ Functions

This section provides descriptions of the Toolkit Core Library functions available for application developers. Each function is described with Function signature, Pre-condition (if applicable), Description, Input parameters, Return values and Sample.

Sample codes in C++ are provided with the description of each function.

5.1 Get Last Error Status

Function Signature

```
long MW_GetLastErrorStatus();
```

Description

Read the status of the last operation performed.

Parameters

Void

Return Values

0	The last operation was successful
Other values	Last operation failed, the last error code is returned. Refer to EIDA Toolkit Troubleshooting document for detailed description of the error codes.

Sample

```
// Do ID card operations

long context = MW_GetLastErrorStatus();

if(context == 0)

     // Last operation succeeded

else

     // Last operation failed
```

5.2 Establishing Context

Function Signature

```
_ULONG MW_EstablishContext();
```

Description

This function initializes the PC/SC context which is a mandatory in order to communicate with the PC/SC compliant smartcard readers. When the communication with the card is no longer needed this context must be released using the MW_CloseContext().

Note: The application should call MW_EstablishContext() function only once.

Parameters

Void

Return Values

0	An error has occurred, call MW_GetLastErrorStatus() to get the error code returned by the kernel.
Other value	Context established successfully, handle is returned.

See also

Closing Context

Sample

```
_ULONG context = MW_EstablishContext();

// Do ID card operations
```

5.3 Closing Context

Function Signature

```
_ULONG MW_CloseContext(_ULONG Context);
```

Description

Closes the context opened with the PC/SC library and free its resources.

Note: The application should call MW_CloseContext () function only once.

Pre-conditions

Context must be opened using the function MW_EstablishContext.

Parameters

`_ULONG Context` `[int]` the context returned by the function MW_EstablishContext.

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Open Context

// Do ID card operations

_ULONG Result = MW_CloseContext(Context);
```

5.4 List Readers

Function Signature

```
LPCWSTR* MW_ListReaders(_ULONG Context, int* NumberOfReaders);
```

Description

Once the context is established, this function can be called to discover all the PC/SC readers connected to the machine.

Pre-conditions

A context must be opened using the function MW_EstablishContext

Parameters

`_ULONG Context`	[in] the established context.
`int* NumberOfReaders`	[out] The number of discovered readers.

Return Values

`LPCWSTR* Readers`	list of discovered readers
`NULL`	no reader is connected or another error occurred. Call MW_GetLastErrorStatus() to get the error code returned by the kernel.

Sample

```
// Some code here

_ULONG context = MW_EstablishContext();

int number_of_readers = 0;

LPCWSTR* readers =  MW_ListReaders(context, &number_of_readers);

// Do ID card operations

// ......

MW_CloseContext(context);
```

5.5 Connect

Function Signature

```
_ULONG MW_Connect(_ULONG Context, LPCWSTR ReaderName);
```

Description

This function establishes a connection with the card inserted in the reader defined by ReaderName parameter.

Pre-conditions

The list of connected readers should be acquired by calling the function MW_ListReaders.

Parameters

`_ULONG Context`	[in] the established context.
`LPCWSTR ReaderName`	[in] the selected reader name, it should be one of the readers returned by the function MW_ListReaders.

Return Values

0	An error has occurred, call MW_GetLastErrorStatus() to get the error code returned by the kernel.
Other value	connection with card has been established, card handle returned.

Sample

```
        // Some code here
        _ULONG context = MW_EstablishContext();
        int number_of_readers = 0;
        LPCWSTR* readers =  MW_ListReaders(context, &number_of_readers);
        LPCWSTR reader = readers[0];
        int cardHandle = MW_Connect(context,reader);
        if(cardHandle <= 0)
        {
             // An error has occured while connecting to the reader
        long errorcode = MW_GetLastErrorStatus();
        }
        else
        {
             // Do some operations with the reader
        int state = MW_Disconnect(cardHandle);
             if(state == 0)
        // disconnected
             else
                   // error has occurred
        `
```

5.6 Disconnect

Function Signature

```
long MW_Disconnect(_ULONG CardHandle);
```

Description

This function closes the connection between the Toolkit and the card.

Pre-conditions

The MW_Connect has to be called first in order to disconnect from the card.

Parameters

`_ULONG CardHandle` [in] handle to the card returned from the `MW_Connect` function

Return Values

0	Successful.

Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

Please refer to the previous example (with the “connect” function).

5.7 Read Card ATR

Function Signature

```
long MW_GetATR(_ULONG CardHandle, BYTEArray* ATR);
```

Description

This function can be used to make an initial check to see whether the inserted ID card is issued by EIDA or not. Please refer to the below HEX representation of the ATR values of EIDA cards:

V1 Card Warm Reset: "3B6A00008065A20130013D72D641"

V1 Card Warm Reset: "3B6A00008065A20131013D72D641"

V2 Card Cold Reset: "3B7A9500008065A20130013D72D641"

V2 Card Warm Reset: "3B7A9500008065A20131013D72D641"

NOTE: validating the ATR will not be enough to assure that the card is genuine. Calling the function `MW_IsCardGenuine` is necessary for this.

Pre-conditions

MW_Connect function must be called before any function that communicates with the card.

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect` function
`BYTEArray* ATR`	[out] the retuned ATR array

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
_ULONG context = MW_EstablishContext();
int number_of_readers = 0;
LPCWSTR* readers =  MW_ListReaders(context, &number_of_readers);
LPCWSTR reader = readers[0];
int cardHandle = MW_Connect(context, reader);
if(cardHandle <= 0)
      // An error has occured while connecting to the reader
else
BYTEArray *ATR = new BYTEArray();
int state = MW_GetATR(cardHandle, ATR);
if(state == 0)
      // no errors
// ......
MW_CloseContext(context);
```

5.8 Is UAE Card

Function Signature

```
long MW_IsUAECard(BYTEArray * ATR);
```

Description

This function checks if card is a production (live) EIDA card by comparing the card ATR provided as parameter with the list of configured UAE card ATRs, please refer to SM.CFG configuration specified in the document “EIDA_Toolkit_Install_and_Configuration_Guide”.

Pre-conditions

Read the card ATR using the function `MW_GetATR`.

Parameters

`BYTEArray* ATR` [in] the card ATR array

Return Values

0	the card is UAE live Card.
Other values	an error has occurred, please refer to EIDA Toolkit Troubleshooting document.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
BYTEArray *ATR = new BYTEArray();
int state = MW_GetATR(cardHandle, ATR);
if(state == 0){
        state = MW_IsUAECard(ATR)
        if(state == 0)
        // process the card version
        Else
        // error occured
}
```

5.9 Is UAE Test Card

Function Signature

```
long MW_IsUAETestCard (BYTEArray * ATR);
```

Description

This function checks if card is a test EIDA ID card by comparing the card ATR provided as parameter with the list of configured Test card ATRs, please refer to SM.CFG configuration specified in the document “EIDA_Toolkit_Install_and_Configuration_Guide”.

Pre-conditions

Read the card ATR using the function `MW_GetATR`.

Parameters

`BYTEArray* ATR`	[in] the card ATR array

Return Values

0	the card is UAE test Card.
Other value	an error has occurred, please refer to EIDA Toolkit

Troubleshooting document for details.

Sample

Please refer to the same sequence specified with the function Is UAE Card

5.10 Is UAE SAM

Function Signature

```
long MW_IsUAESAM(BYTEArray * ATR);
```

Description

This function checks if the inserted SAM card is a production (live) EIDA SAM by comparing the card ATR provided as parameter with the list of configured SAM ATRs. Refer to SM.CFG configuration specified in "EIDA_Toolkit_Install_and_Configuration_Guide".

Pre-conditions

Read the SAM card ATR using the function `MW_GetATR`.

Parameters

`BYTEArray* ATR` [in] the card ATR array

Return Values

0	the card is UAE SAM Card.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

Please refer to the same sequence specified with the function Is UAE Card.

5.11 Is Contactless Reader

Function Signature

```
long MW_IsContactlessReader(_ULONG CardHandle);
```

Description

This function checks whether the reader communicating with the specified card handle is contactless reader or not.

Pre-conditions

MW_Connect function must be called as CardHandle is required for the function **`MW_IsContactlessReader`**.

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect`Function

Return Values

1	The connected reader referenced by the CardHandle is contactless.
0	The connected reader referenced by the CardHandle is NOT contactless.

Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
long type = MW_IsContactlessReader(cardHandle);
if (type == 1)
// reader is contactless
else if(type == 0)
// reader is not contactless
else
// error occured
```

5.12 Get Card Version

Function Signature

```
long MW_GetCardVersion(_ULONG CardHandle, int* CardVersion);
```

Description

This function returns the version of EIDA ID card issued by EIDA. Currently there are two versions of cards are issued by EIDA (v1 and V2).

Pre-conditions

MW_Connect function must be called before any function the communicates with the card.

Parameters

ULONG CardHandle	[in] handle to the card returned from MW_Connect Function
int* CardVersion	[out] a pointer integer variable, the function set the integer variable to 1 for V1 cards and 2 for V2 cards.

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
int *CardVersion;
long state = MW_GetCardVersion(cardHandle, CardVersion);
if (state == 0)
// process the card version
Else
// error occured
```

5.13 Get Card Serial Number

Function Signature

```
long MW_GetCardSerialNumber(_ULONG CardHandle, BYTEArray* CSN);
```

Description

This function reads the card serial number (CSN) from card, CSN is used an input to the secure messaging functions.

Pre-conditions

MW_Connect function must be called before any function communicates with the card.

Parameters

ULONG CardHandle	[in] handle to the card returned from MW_Connect Function

`BYTEArray* CSN`	[out] byte array to hold the card serial number

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
BYTEArray* CSN = new BYTEArray();
int state = MW_GetCardSerialNumber(cardHandle, CSN);
```

5.14 Get Chip Serial Number

Function Signature

```
long MW_GetChipSerialNumber(_ULONG CardHandle, BYTEArray* ChipSN);
```

Description

This function reads the chip serial number from card.

Pre-conditions

MW_Connect function must be called before any function the communicates with the card.

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect`Function
`BYTEArray* ChipSN`	[out] byte array to hold the chip serial number

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
BYTEArray* ChipSerialNumber = new BYTEArray();
Int state=MW_GetChipSerialNumber(cardHandle, ChipSerialNumber);
```

5.15 Get Issuer Serial Number

Function Signature

```
long MW_GetIssuerSerialNumber(_ULONG CardHandle, BYTEArray* IssuerSN);
```

Description

This function reads the issuer serial number from card.

Pre-conditions

MW_Connect function must be called before any function which communicates with the card.

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect`Function
`BYTEArray* IssuerSN`	[out] byte array to hold the issuer serial number

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
BYTEArray* IssuerSerialNumber = new BYTEArray();
int state=MW_GetIssuerReferenceNumber(cardHandle, IssuerSerialNumber);
```

5.16 Get Issuer reference Number

Function Signature

```
long MW_GetIssuerReferenceNumber(_ULONG CardHandle, BYTEArray*
IssuerReferenceNumber);
```

Pre-conditions

MW_Connect function must be called before any function that communicates with the card.

Description

Reads the issuer reference number from card.

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect`Function
`BYTEArray* IssuerReferenceNumber`	[out] byte array to hold the issuer reference number

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
BYTEArray* IssuerReferenceNumber = new BYTEArray();
Int state= MW_GetIssuerReferenceNumber(cardHandle ,
IssuerReferenceNumber);
```

5.17 Get CPLC0101

Function Signature

```
long MW_GetCPLC0101(_ULONG CardHandle, BYTEArray* CPLC0101);
```

Description

This functions reads the card production life cycle (CPLC) information from the card tag

0101.

Pre-conditions

MW_Connect function must be called before any function that communicates with the card.

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect`Function
`BYTEArray* CPLC0101`	[out] byte array to hold CPLC

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
BYTEArray* CPLC0101 = new BYTEArray();
Int state= MW_GetCPLC0101(cardHandle , CPLC0101);
```

5.18 Get CPLC9F7F

Function Signature

```
long MW_GetCPLC9F7F(_ULONG CardHandle, BYTEArray* CPLC9F7F);
```

Description

This function reads the card production life cycle (CPLC) information from the card.

Pre-conditions

MW_Connect function must be called before any function that communicates with the card.

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect`Function

`BYTEArray* CPLC9F7F` [out] byte array to hold CPLC

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
BYTEArray* CPLC9F7F = new BYTEArray();
Int state= MW_GetCPLC9F7F(cardHandle , CPLC9F7F);
```

5.19 Read Public Data

Function Signature

```
Long MW_ReadPublicData(_ULONG CardHandle,
CardHolderPublicData* PublicData, BOOL ReadPhotography, BOOL
ReadNonModifiableData, BOOL ReadModifiableData, BOOL
SignatureValidation);
```

Description

This function fills an object of type `CardHolderPublicData` Class carrying the card holder public data. 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.

Additionally the date fields are represented in 4 bytes and it should be decoded. Refer to the example on how to decode.

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 last 4 boolean parameters when it is called.

NOTE: the value of the following data fields are represented in codes, so it needs to be mapped to the description of that codes to have readable information, please contact EIDA for the latest description of each codes.

- Sex
- Occupation
- Marital Status

- Sponsor Type
- Residency Type
- Nationality

Pre-conditions

MW_Connect function must be called first.

Parameters

`_ULONG CardHandle`	[in] a handle to the card to retrieve the public data from
`CardHolderPublicData* PublicData`	[out] the instance to hold the card holder's public data
`BOOL ReadPhotography`	[in] a flag to define whether to read the photography or not
`BOOL ReadNonModifiableData`	[in] a flag to define whether to read the non modifiable data section or not
`BOOL ReadModifiableData`	[in] a flag to define whether to read the modifiable data section or not
`BOOL SignatureValidation`	[in] a flag to define whether to validate the public data signature. Refer to the Additional note below.

Additional note for SignatureValidation flag:

If the flag is set to true, Toolkit will verify the signature using the data signing certificates located in the folder location which is configured in sm.cfg file. Please refer to Appendix A of the Java/.NET developers guide document 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. If the `SignatureValidation` flag is set to true and if the corresponding signing certificate to the card couldn't be found then certificate not found error will be returned.

Note to EIDA: The Toolkit setup will copy all the signing certificates which were issued till the releasing version 2.5 of the Toolkit. Contact EIDA for any new certificates.

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
CardHolderPublicData *PublicData = new CardHolderPublicData();
Int state= MW_ReadPublicData (cardHandle , PublicData);
//Formating date a dd/MM/yyyy
CString strDateOfBirthDate =  ByteToHex(PublicData.DateOfBirth.Value[3])
+ "/" + ByteToHex(PublicData.DateOfBirth.Value[2]) + "/" +
ByteToHex(PublicData.DateOfBirth.Value[0]) +
ByteToHex(PublicData.DateOfBirth.Value[1]);
```

5.20 Read Public Data Ex

Function Signature

```
Long MW_ReadPublicDataEx(_ULONG CardHandle,
CardHolderPublicData* PublicData, BOOL ReadPhotography, BOOL
ReadNonModifiableData, BOOL ReadModifiableData, BOOL
SignatureValidation, BOOL ReadV2Fields, BOOL ReadSignatureImage,
BOOL ReadAddress);
```

Pre-conditions

MW_Connect function must be called first.

Description

This function is an extension of Read Public Data function to support additional public data fields implemented on V2 EIDA ID Card Applet and Address data container.

Note that the last Boolean parameter (**ReadV2Fields**) shall be set to TRUE only when using V2 cards in order to enable reading the new data sets. If **ReadV2Fields** is set to TRUE while using a V1 card this function will return E_V2Data_NOT_AVAILABLE_IN_V1 error

ReadSignatureImage and **ReadAddess** fields are not checked if the flag **ReadV2Fields** passed with FALSE value.

Parameters

`_ULONG CardHandle`	[in] a handle to the card to retrieve the public data from
`CardHolderPublicDataEx* PublicDataEx`	[out] the instance to hold the card holder's public data
`BOOL ReadPhotography`	[in] a flag to define whether to read the photography or not

`BOOL ReadNonModifiableData`	[in] a flag to define whether to read the non modifiable data section or not
`BOOL ReadModifiableData`	[in] a flag to define whether to read the modifiable data section or not
`BOOL SignatureValidation`	[in] a flag to define whether to validate the public data signature
`BOOL ReadV2Fields`	[in] a flag to define whether to read the V2 additional data sets
`BOOL ReadSignatureImage`	[in] a flag to define whether to read the card holder signature image in V2 cards.
`BOOL ReadAddress`	[in] a flag to define whether to read Home and Work address fields in V2 cards.

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
CardHolderPublicDataEx *PublicDataEx = new CardHolderPublicDataEx();
Int state= MW_ReadPublicDataEx (cardHandle , PublicDataEx, True, True,
True, True, True, True, True);
//Formating date a dd/MM/yyyy
CString strDateOfBirthDate =
ByteToHex(PublicDataEx.DateOfBirth.Value[3]) + "/" +
ByteToHex(PublicDataEx.DateOfBirth.Value[2]) + "/" +
ByteToHex(PublicDataEx.DateOfBirth.Value[0]) +
ByteToHex(PublicDataEx.DateOfBirth.Value[1]);
```

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

Function Signature

```
long MW_ReadPublicDataContactless(_ULONG CardHandle,BYTEArray
*CardNumber,BYTEArray* DateOfBirth,BYTEArray* ExpiryDate,
CardHolderPublicDataEx* PublicData, BOOL ReadPhotography, BOOL
ReadNonModifiableData, BOOL ReadModifiableData, BOOL SignatureValidation);
```

Description

Reading the public data from EIDA card is protected by Basic Access Control (BAC), MW_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, the 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 MW_ReadPublicDataEx function.

NOTE:

1. The value of the following data fields that returned within the CardHolderPublicData class are represented in codes, so it needs to be mapped to the description of that codes to have readable information, please contact EIDA for the latest description of each codes.
    - Sex
    - Occupation
    - Marital Status
    - Sponsor Type
    - Residency Type
    - Nationality
2. MW_ReadPublicDataContactless works only with contactless PC\SC readers.
3. MW_ReadPublicDataContactless works only V2 Cards.

Pre-conditions

MW_Connect function must be called first.

Parameters

`_ULONG CardHandle` [in] a handle to the card to retrieve the

	public data from
`BYTEArray *CardNumber`	[in] card number in binary format
`BYTEArray *DateOfBirth`	[in] card holder date of birth in binary format
`BYTEArray *ExpiryDate`	[in] card expiry date in binary format
`CardHolderPublicDataEx* PublicData`	[out] the instance to hold the card holder's public data
`BOOL ReadPhotography`	[in] a flag to define whether to read the photography or not
`BOOL ReadNonModifiableData`	[in] a flag to define whether to read the non modifiable data section or not
`BOOL ReadModifiableData`	[in] a flag to define whether to read the modifiable data section or not
`BOOL SignatureValidation`	[in] a flag to define whether to validate the public data signature

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
CardHolderPublicDataEx *PublicData = new CardHolderPublicDataEx();
Int state= MW_ReadPublicDataContactless(cardHandle , cardNumber,
dateOfBirth, expiryDate, PublicData, TRUE, TRUE, TRUE, TRUE);
//Formating date a dd/MM/yyyy
CString strDateOfBirthDate =  ByteToHex(PublicData.DateOfBirth.Value[3])
+ "/" + ByteToHex(PublicData.DateOfBirth.Value[2]) + "/" +
ByteToHex(PublicData.DateOfBirth.Value[0]) +
ByteToHex(PublicData.DateOfBirth.Value[1]);
```

5.22 Read Public Data Contactless (with MRZ Reader)

Function Signature

```
long MW_ReadPublicDataContactlessWithMRZData (_ULONG CardHandle,BYTEArray
*MRZData, CardHolderPublicDataEx* PublicData, BOOL ReadPhotography, BOOL
```

```
ReadNonModifiableData, BOOL ReadModifiableData, BOOL SignatureValidation);
```

Description

Reading the public data from EIDA card is protected by Basic Access Control (BAC), MW_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 MW_ReadPublicDataContactlessWithMRZData

Once BAC keys are diversified, the 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 MW_ReadPublicDataEx function.

NOTE:

1. The value of the following data fields that returned within the CardHolderPublicDataEx class are represented in codes, so it needs to be mapped to the description of that codes to have readable information, please contact EIDA for the latest description of each codes.
    - Sex
    - Occupation
    - Marital Status
    - Sponsor Type
    - Residency Type
    - Nationality
2. MW_ReadPublicDataContactlessWithMRZData works only with contactless PC\SC readers.
3. MW_ReadPublicDataContactlessWithMRZData works only V2 Cards

Pre-conditions

MW_Connect function must be called first.

Parameters

`_ULONG CardHandle`	[in] a handle to the card to retrieve the public data from
`BYTEArray *MRZData`	[in] MRZ data
`CardHolderPublicDataEx * PublicData`	[out] the instance to hold the card holder's public data
`BOOL ReadPhotography`	[in] a flag to define whether to read the photography or not
`BOOL ReadNonModifiableData`	[in] a flag to define whether to read the non modifiable data section or not
`BOOL ReadModifiableData`	[in] a flag to define whether to read the modifiable data section or not
`BOOL SignatureValidation`	[in] a flag to define whether to validate the public data signature

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Some code here
CardHolderPublicData *PublicData = new CardHolderPublicData();
Int state= MW_ReadPublicDataContactlessWithMRZData(cardHandle , mrzData,
PublicData, TRUE, TRUE, TRUE, TRUE);
//Formating date a dd/MM/yyyy
CString strDateOfBirthDate =  ByteToHex(PublicData.DateOfBirth.Value[3])
+ "/" + ByteToHex(PublicData.DateOfBirth.Value[2]) + "/" +
ByteToHex(PublicData.DateOfBirth.Value[0]) +
ByteToHex(PublicData.DateOfBirth.Value[1]);
```

5.23 Read Family Book Data

Function Signature

```
long MW_ReadFamilyBookData(_ULONG CardHandle,FamilyBookData * FamilyBook)
```

Pre-conditions

- MW_Connect function must be called first.
- MW_Init function must be called before calling MW_IsCardGenuine in order to initialize the secure messaging (SM) modules.

Description

This function fills an object of type `FamilyBookData` Class carrying the card holder family book application. 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.

Note : the family book application is supported only on V2 cards hence ,using this function with a V1 card will return E_V2Data_NOT_AVAILABLE_IN_V1 error.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`FamilyBookData* familyBookData`	[out] the instance to hold the card holders family book data

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details

Sample

```
// Some code here
FamilyBookData *familyBookData = new FamilyBookApplication();
Int state= MW_FamilyBookData (cardHandle , familyBookData);
CString strDateOfBirthDate =  ByteToHex(familyBookData
->Child1.DateOfBirth.Value[3]) + "/" + ByteToHex(familyBookData
->Child1.DateOfBirth.Value[2]) + "/" + ByteToHex(familyBookData
->Child1.DateOfBirth.Value[1]) + ByteToHex(familyBookData
->Child1.DateOfBirth.Value[0]
```

5.24 Init SM

Function Signature

```
Long MW_Init();
```

Description

This function loads the secure messaging modules configurations from the sm.cfg file, that file should be configures as described in appendix A of the Java/.NET developers guide document.

Parameters

Void

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for error details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
int state = MW_Init();
if(state == 0)
{
      // initialization successful
}
else
      // error has occurred
```

5.25 Is Card Genuine

Function Signature

```
long MW_IsCardGenuine(_ULONG CardHandle);
```

Description

This function verifies an ID card is genuine using local secure messaging module defined in the sm.cfg to be used with the ID Applet.

Pre-conditions

MW_Init function must be called before calling MW_IsCardGenuine in order to initialize the secure messaging (SM) modules.

Parameters

_ULONG CardHandle	[in] handle to the card returned from the MW_Connect function

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
int state = MW_Init();
if(state == 0)
{
	int state = MW_IsCardGenuine (Handle);
	if(state == 0)
		// card Is Genuine
	else
		// error has occurred
}
else
	// error has occurred
```

5.26 Is Card Genuine Ex

Function Signature

```
long MW_IsCardGenuineEx(_ULONG CardHandle);
```

Description

This function extends the “is_card_genuine” function 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.

Pre-conditions

MW_Init function must be called before calling MW_IsCardGenuineEx in order to

initialize the SM modules.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function

Return Values

0	the card is valid UAE ID card
Other value	an error has occurred, or the card is not valid UAE ID card, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
int state = MW_Init();
if(state == 0)
{
      int state = MW_IsCardGenuineEx (Handle);
      if(state == 0)
            // card Is Genuine
      else
            // error has occurred
}
else
      // error has occurred
```

5.27 Get MOC Biometric Information Template

Function Signature

```
long MW_GetMOCBiometricInformationTemplate(_ULONG CardHandle,
BIT* Bit1, BIT* Bit2);
```

Description

This function populates two instances of the BIT Class which specifies the finger's indices and reference data qualifier of the two fingerprints stored on the card. This information is required for doing biometrics matching on/off card in order to decide which fingerprint should be captured and send to the matching function and which reference to be used in the case of the match on card.

Pre-conditions

MW_Connect function must be called before any function that communicates with the card.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BIT* Bit1`	[out] an instance to hold the biometric information template for the first fingerprint
`BIT* Bit2`	[out] an instance to hold the biometric information template for the second fingerprint

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

Please refer to the sample provided with the match on card function.

5.28 Get MOC serial number

Function Signature

```
long MW_GetMOCSerialNumber(_ULONG CardHandle, BYTEArray* MOCSerialNumber);
```

Description

Reads the match on card (MOC) applet's serial number from the card, this serial number will be used as an input to MOC match function.

Pre-conditions

MW_Connect function must be called before any function the communicates with the card.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BYTEArray* MOCSerialNumber`	[out] a byte array to hold the MOC serial number

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)

BYTEArray MOCSerialNumber;
      Long state = MW_GetMOCSerialNumber(handle,&MOCSerialNumber);

      if(state == 0)
            // Function Succeeded, process serial number returned
      else
            // error has occurred
```

5.29 Get MOC Applet state

Function Signature

```
Long MW_GetMOCAppletState(_ULONG CardHandle, BYTEArray* MOCAppletState);
```

Description

This function reads the MOC applet state which is a single byte indicating the state code.

Pre-conditions

MW_Connect function must be called before any function that communicates with the card.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BYTEArray* MOCAppletState`	[out] byte array to hold the MOC applet state

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit

Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)

BYTEArray MOCAppletState;
      Long state = MW_GetMOCAppletState(handle,&MOCAppletState);

      if(state == 0)
            // Function Succeeded, process serial number returned
      else
            // error has occurred
```

5.30 Get Max Failed Match

Function Signature

```
long MW_GetMaxFailedMatch(_ULONG CardHandle, BYTEArray* MaxFailedMatch);
```

Pre-conditions

MW_Connect function must be called before any function which requires communication with the card.

Description

Get the maximum number of remaining failed trials when executing matching on card represented in one byte.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BYTEArray* MaxFailedMatch`	[out] a byte array to with a single byte represents the remaining trials.

Return Values

0	Successful.

Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)

BYTEArray MaxFailedMatch;
      Long state = MW_GetMaxFailedMatch(handle,& MaxFailedMatch);

      if(state == 0)
            // Function Succeeded, process number of reaming failure
            returned
      else
            // error has occurred
```

5.31 Get MOC Algorithm Version

Function Signature

```
long MW_GetMOCAlgorithmVersion(_ULONG CardHandle, BYTEArray*
AlgorithmVersion);
```

Description

This function reads the version of the algorithm used in the on card biometric matching operations.

Pre-conditions

MW_Connect function must be called before any function which requires communicates with the card.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BYTEArray* AlgorithmVersion`	[out] a byte array represents the algorithm version.

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)

BYTEArray AlgorithmVersion;
      Long state = MW_GetMOCAlgorithmVersion(handle,&AlgorithmVersion);

      if(state == 0)
            // Function Succeeded, process Algorithm version returned
      else
            // error has occurred
```

5.32 Capture Image

Function Signature

```
long MW_CaptureImage(int SensorID, FTP_Image* Image);
```

Description

This function captures the fingerprint image from the fingerprint sensor and checks 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 offer an Off-the-shelf integration with Sagem MSO 1350 sensor, so if this sensor is used then a developer can directly call this function with passing -1 as SensorID.

The Toolkit design gives the flexibility to use other sensors as well through implementing standardised interface specified in appendix C of the Java/.NET developers guide. Refer to the document for further details.

So all what the developer needs to do in order to use a sensor other than Sagem MSO 1350 is the following:

1. Implement the interface specified in Appendix C of the Java/.NET developers guide document as a C++ DLL

2. Change the configuration in the file “sensors.cfg” as specified in the Appendix C.
3. Call the MW_CaptureImage as below.
    a. Set the SensorID parameter with the corresponding id of 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 will try to connect to Sagem MSO 1350 sensor. If it fails then the capture function will return E_BIOMETRICS_NO_DEVICE error.
    b. The returned BMP image that shall be passed later to the API function MW_ConvertImage to get the template required for the matching function.

Parameters

`int SensorID`	[in] the ID to the sensor to be used in capturing the fingerprint (0 auto detect, -1 in case of SAGEM device, 1,2, ... for other devices).
`FTP_Image* Image`	[in/out] specifies the fingerprint index need to be captured, and returns the captured image in RAW format.

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

Please refer to the sample provided with the match on card function.

5.33 Capture and Convert

Function Signature

```
long MW_CaptureAndConvert(int SensorID, FTP_Template* Template1);
```

Description

If the sensor has a built-in feature which allows direct conversion of the captured image to any of the templates supported by the Toolkit matching functions (ISO_19794_CS or DINV_66400) then this function can be used. Note that this function is useful only if SAGEM MSO 1350 sensor is used or another other sensor with interface DLL consists the function Capture_Convert as specified in appendix C of the Java/.NET developers guide document.

Parameters

`int SensorID`	[in] the ID to the sensor to be used in capturing the fingerprint (always 1)
`FTP_Template* Template1`	[in\out] a pointer an instance of the structure FTP_Template, template format and fingerprint index specified as follows: - `FTP_Template.FingerIndex:` indicating the fingerprint index according to the information read by the function. `"MW_GetMOCBiometricInformationTemplate"` - `FTP_Template.FingerIndex: template` format ISO(0), DINV(1) . ISO is used for off card matching while DINV is used for on card matching.

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

Refer to the sample provided with the match on card function.

5.34 Convert Image

Function Signature

```
long MW_ConvertImage(FTP_Image* Image, FTP_Template* Template1);
```

Description

This function converts a fingerprint image into a fingerprint template to be used later in the matching process. The fingerprint image has to be in RAW format.

Parameters

`FTP_Image* Image`	[in] the image to be converted
`FTP_Template* Template1`	[in/out] an instance of the structure `FTP_Template` specifies the desired template format.

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

Please refer to the sample provided with the match on card function.

5.35 Convert BMP Image

Function Signature

```
long MW_ConvertBmpImage(FTP_Image* BmpImage, FTP_Template*
Template1);
```

Description

This function is required when other type of sensors (non SAGEM) are used to capture the fingerprint; in this case the integration with the sensor is done through separate component that returns the BMP image captured from the sensor. The function MW_ConvertBmpImage converts the BMP fingerprint image into a fingerprint template to be used later in the matching process.

Parameters

`FTP_Image* BmpImage`	[in] the BMP image to be converted
`FTP_Template* Template1`	[in/out] an instance of the structure `FTP_Template` specifies the desired template format.

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
BYTEArray BmpImage;
FTP_Template Template1;
// capture fingerprint using separate component
// allocate space for the variable BmpImage. based on the BMP buffer
length
// copy the BMP buffer to the variable BmpImage.Value and the length to
the available BmpImage.Length
long state = MW_ConvertBmpImage(&BmpImage, & Template1)
      if(state == 0)
            // Function Succeeded, use Template1 with Matching functions
      else
            // error has occurred
```

5.36 Match on-Card

Function Signature

```
long MW_MatchOnCard(_ULONG CardHandle, BIT* Bit, FTP_Template*
Template1);
```

Description

After capturing a fingerprint and converting the captured image into the appropriate template, the resulting template is sent as a parameter to the function MW_MatchOnCard along with the desired on card template reference. The template on the card is identified using the BIT object retrieved from the card using function MW_GetMOCBiometricInformationTemplate.

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

Pre-conditions

Before calling the matching process, a valid fingerprint template is needed.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BIT* Bit`	[in] bit reference to the template to match with.
`FTP_Template* Template1`	[in] Captured and converted fingerprint

Return Values

0	Matching successful
>0	Matching failed, this positive number represents the number of remaining match try before blocking the matching operation on the card.
<0	an error has occurred, please refer to EIDA Toolkit Troubleshooting document.

EMIRATES IDENTITY AUTHORITY

Sample

```
// Iniailize Connction with the card to get the card handle (Handle)
Long state;
BIT* Bit1 = new BIT;
BIT* Bit2 = new BIT;
State = MW_GetMOCBiometricInformationTemplate(handle, Bit1, Bit2);
if(state == 0)
{
      FTP_Template Template1;
      FTP_Image image;
      image.FingerIndex = Bit1.FingerIndex;//set the fingerprint index
      to the first template stored on the card

      //capture the fingerprint using the SAGEM sensor
      state = MW_CaptureImage(1, &image);
      if(state == 0)
      {
            Templae1.Format = 1;//DINV
            state = MW_ConvertImage(&image, &Template1);
            if(state == 0)
            {
                  state = MW_MatchOnCard(Handle);
                  if(state == 0)
                        //matched
                  Else
                        // no match
            }
            else
                  // error has occurred
      }
      else
            // error has occurred
}
else
      // error has occurred
```

5.37 Read Fingerprints

Function Signature

```
long MW_ReadFingerprints(_ULONG CardHandle, FTP_Template*
Tpl1, FTP_Template* Tpl2);
```

Description

This function reads the two fingerprint templates stored on the card to be used later for Off-Card matching.

Pre-conditions

A secure messaging session must be established with the card before calling this function. This can be achieved by calling the function MW_IsCardGenuine.

Parameters

`_ULONG CardHandle`	[in] a handle to the card to be used in the extraction process
`FTP_Template* Tpl1`	[out] a template instance to hold the first fingerprint template stored on the card
`FTP_Template* Tpl2`	[out] a template instance to hold the second fingerprint template stored on the card

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

Please refer to the sample provided with the match off card function.

5.38 Match off-Card

Function Signature

```
long MW_Match(FTP_Template* Tpl1, FTP_Template* Tpl2);
```

Description

EIDA toolkit has built-IN ISO matcher that can match two ISO templates and returns the matching score. This function uses the matcher to compare two templates together. If the matching score is greater than 13000, the Offcard matching is successful.

Pre-conditions

- Capture fingerprint image and convert it to a template
- Read the fingerprint from the card

Parameters

`FTP_Template* Tpl1`	[in] the first fingerprint template to match

`FTP_Template* Tpl2`	[in] the second fingerprint template to match

Return Values

<=0	an error has occurred, please refer to EIDA Toolkit Troubleshooting document
>13000	Matching Success
<13000	Matching Failed

Sample

```
// Initialize Connection with the card to get the card handle (Handle)
long state;
BIT* Bit1 = new BIT;BIT* Bit2 = new BIT;
State = MW_GetMOCBiometricInformationTemplate(handle, Bit1, Bit2);
if(state == 0)
{
	FTP_Template Template1;
	FTP_Image image;
	image.FingerIndex = Bit1.FingerIndex;//set the
	index of the first template stored on the card

	state = MW_CaptureImage( 1, &image);
	if(state == 0)
	{
		Template1.Format = 1;//DINV
		state = MW_ConvertImage(&image,&Template1);
		if(state == 0)
		{
			state = MW_Init();
			state = MW_IsCardGenuine(handle);
			if(state == 0)
			{
				FTP_Template Tpl1,  Tpl2;
				state = MW_ReadFingerprints(handle, &Tpl1,
			 &Tpl2);

				if(state == 0)
				{
					//match the captured templates with the
				first template stored on the card

					state = MW_Match(Tpl1, Template1);
					if(state >= 13000)
						//matched
					Else
						// no match
				}
			}
		}
		else
			// error has occurred
	}
	else
		// error has occurred
}
else
	// error has occurred
```

EMIRATES IDENTITY AUTHORITY

5.39 Generate Card Cryptogram

Function Signature

```
long MW_GenerateCardCryptogram(_ULONG CardHandle, BYTEArray* SM_Challenge, BYTEArray* CardCryptogram);
```

Description

This function generates the card cryptogram using the secure messaging challenge generated by calling MW_SM_GetChallenge function.

Pre-conditions

To be able to generate the card cryptogram, a handle to the card must be acquired.

Parameters

`_ULONG CardHandle`	[in] a handle to the card to be used to generate the cryptogram
`BYTEArray* SM_Challenge`	[in] Challenge generated by the SM module
`BYTEArray* CardCryptogram`	[out] a byte array to hold the cryptogram of the card after generating it

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize a Connection with the card

// in case the SAM is used as SM module, the SM_ID is SAM card
connection.

// open an SM context
Long Context = MW_SM_OpenContext(SM_ID);

// generate the SM challenge
BYTEArray* SM_Challenge = new BYTEArray();
MW_SM_GetChallenge(SM_ID, Context, SM_Challenge);

// Compute the card cryptogram
BYTEArray* CardCryptogram = new BYTEArray();
MW_GenerateCardCryptogram(CardHandle, SM_Challenge, CardCryptogram);
```

5.40 Verify Ciphered PIN

Function Signature

```
long MW_VerifyCipheredPIN(_ULONG CardHandle, BYTEArray* CipheredPin);
```

Description

This function verifies the ciphered PIN generated by the SM module.

Pre-conditions

In order to be able to verify the ciphered pin the MW_SM_GetCipheredPIN function has to be called in order to retrieve the ciphered pin locally.

Parameters

`_ULONG CardHandle`	[in] a handle to the card to be used in the verification process
`BYTEArray* CipheredPin`	[in] the ciphered pin retrieved from the function MW_SM_GetCipheredPIN

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
// Call CipheredPIN = MW_SM_GetCipheredPIN(...)
Long status = MW_VerifyCipheredPIN(Handle, CipheredPIN);
If(status == 0)
{
// Card genuine
}
else
{
// error, check the error values on the troubleshooting guide
}
```

5.41 Export PKI Certificates

Function Signature

```
long MW_ExportCertificates(_ULONG CardHandle, int CardVersion,
BYTEArray* authCert, BYTEArray* signCert);
```

Description

This function reads the authentication and singing certificates from the card.

Pre-conditions

- Card handle is acquired.
- Call the function MW_GetCardVersion to detect the card version that is required as a parameter to this function.

Parameters

Parameter	Description
`_ULONG CardHandle`	[in] a handle to the card
`int CardVersion`	[in] the card version read by the function `MW_GetCardVersion`
`BYTEArray* authCert`	[out] byte array hold the exported authentication certificate
`BYTEArray* signCert`	[out] byte array hold the exported signing certificate

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)

BYTEArray authCert;
BYTEArray signCert;
int CardVersion;
MW_GetCardVersion(handle, &CardVersion);
long state = MW_ExportCertificates(handle, CardVersion, &authCert,
&signCert);

if(state == 0)

//process the retuned certificates

Else

//error occurred
```

5.42 Sign Data

Function Signature

```
long MW_SignData(_ULONG CardHandle, BYTEArray* PIN, BYTEArray*
data, BYTEArray* pSignature);
```

Description

This functions signs the binary data with the signing certificate stored on the card using the card native APIs where PIN caching is disabled.

Pre-conditions

Card handle is acquired.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BYTEArray* PIN`	[in] the card PIN in binary format, E.g. If the card PIN is 1234 then the PIN.Value array should be

	{0x1,0x2,0x3,0x4}
`BYTEArray* data`	[in] data to be signed
`BYTEArray* pSignature`	[out] digital signature of the input data in PKCS#1 format

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
// set the card PIN (an Application should prompt the user to enter the
PIN)
      BYTE PINarray[4];
      PINarray[0]=0x1;
      PINarray[1]=0x2;
      PINarray[2]=0x3;
      PINarray[3]=0x4;
      BYTEArray PIN;
      PIN.Value = PINarray;
      PIN.Length = 4;
      // set the data to be signed
      BYTEArray data;
      data.Length = 8;
      data.Value = (LPBYTE)malloc(8);
      memset(data.Value,0,8);
      // define the signature variable where signature will be stored
      BYTEArray *pSignature = new BYTEArray();
      // call the sign function
      long state = MW_SignData(handle, &PIN, &data, pSignature);
```

5.43 Authenticate

Function Signature

```
long MW_Authenticate(_ULONG CardHandle, BYTEArray* PIN,
BYTEArray* data, BYTEArray* pSignature);
```

Description

This function signs the binary data with the authentication certificate stored on the card., This function is used to sign random challenge sent from authentication server for authentication purpose. The user will be prompted to fill the PIN with every function call.

Pre-conditions

Card handle acquired.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BYTEArray* PIN`	[in] the card PIN in binary format, E.g. If the card PIN is 1234 then the PIN.Value array should be {0x1,0x2,0x3,0x4}
`BYTEArray* data`	[in] data to be signed
`BYTEArray* pSignature`	[out] digital signature of the input data in PKCS#1 format

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
// set the card PIN (an Application should prompt the user to enter the
PIN)
      BYTE PINarray[4];
      PINarray[0]=0x1;
      PINarray[1]=0x2;
      PINarray[2]=0x3;
      PINarray[3]=0x4;
      BYTEArray PIN;
      PIN.Value = PINarray;
      PIN.Length = 4;
      // set the data to be signed
      BYTEArray data;
      data.Length = 8;
      data.Value = (LPBYTE)malloc(8);
      memset(data.Value,0,8);
      // define the signature variable where signature will be stored
      BYTEArray *pSignature = new BYTEArray();
      // call the sign function
      long state = MW_Authenticate(handle, &PIN, &data, pSignature);

      if(state == 0 )

      // process the PKCS#1 signature returned in the pSignature
      Else
      //An Error occured
```

5.44 Authenticate with PIN caching

Function Signature

```
long MW_Authenticate_PinCached (_ULONG CardHandle, BYTEArray* data, BYTEArray* pSignature);
```

Description

This function performs as the function `MW_Authenticate`. However, it caches the PIN so that the user will be prompt to fill the PIN only once then the PIN will be cached for the subsequent calls.

The function will display a dialog box to prompt the user for entering the PIN when it called for the first time.

Pre-conditions

Card handle acquired.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BYTEArray* data`	[in] data to be signed
`BYTEArray* pSignature`	[out] digital signature of the input data in PKCS#1 format

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
// set the data to be signed
BYTEArray data;
data.Length = 8;
data.Value = (LPBYTE)malloc(8);
memset(data.Value,0,8);
// define the signature variable where signature will be stored
BYTEArray *pSignature = new BYTEArray();
// call the sign function
long state = MW_Authenticate_PinCached(handle, &data, pSignature);
if(state == 0 )
// process the PKCS#1 signature returned in the pSignature
Else
//An Error occured
```

5.45 Unblock PIN

Function Signature

```
long MW_UnBlockPIN(_ULONG CardHandle,BYTEArray* PIN);
```

Description

This function unblocks the card PIN code and sets the newly passed PIN in the card.

This function requires access to SM module locally therefore, the SM.CFG file needs to be configured as specified in the document "EIDA_Toolkit_Install_and_Configuration_Guide".

Pre-conditions

- Initialise the secure messaging module(s) using the function `MW_Init`
- Card handle acquired

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BYTEArray* PIN`	[in] the new card PIN in binary format, E.g. If the card PIN is 1234 then the PIN.Value array should be {0x1,0x2,0x3,0x4}

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// initialize secure messaging module accoring to SM.cfg configuration
file
long state  = MW_Init();
//Initialize connection with the card to get the card handle (Handle)
_ULONG handle = MW_Connect(m_Context, strReader);
// set the card PIN (an Application should prompt the user to enter the
PIN)
BYTE PINarray[4];
PINarray[0]=0x1;
PINarray[1]=0x2;
PINarray[2]=0x3;
PINarray[3]=0x4;
BYTEArray PIN;
PIN.Value = PINarray;
PIN.Length = 4;
// the PIN Unblock function
state =  MW_UnBlockPIN(handle, &PIN);occurred
If(state == 0 )
//PIN reset succeeded
Else
//PIN reset failed
```

5.46 Change PIN

Function Signature

```
long MW_ChangePIN(_ULONG CardHandle, BYTEArray* OldPIN,BYTEArray* NewPIN);
```

Description

This function changes the old card PIN with the new one passed in the function call.

Pre-conditions

Card handle acquired.

Parameters

`_ULONG CardHandle`	[in] handle to the card returned from the `MW_Connect` function
`BYTEArray* OldPIN`	[in] the old card PIN in binary format, E.g. If the card PIN is 1234 then the PIN.Value array should be {0x1,0x2,0x3,0x4}
`BYTEArray* NewPIN`	[in] the New card PIN in binary format, E.g. If the card PIN is 1234 then the PIN.Value array should be {0x1,0x2,0x3,0x4}

EMIRATES IDENTITY AUTHORITY

Return Values

0	Successful.
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
 //Initialize connection with the card to get the card handle (Handle)
_ULONG handle = MW_Connect(m_Context, strReader);
// set the old card PIN (an Application should prompt the user to enter the PIN)
BYTE OldPINarray[4];
OldPINarray[0]=0x1;
OldPINarray[1]=0x2;
OldPINarray[2]=0x3;
OldPINarray[3]=0x4;
BYTEArray OldPIN;
OldPIN.Value = OldPINarray;
OldPIN.Length = 4;
// set the old card PIN (an Application should prompt the user to enter the PIN)
BYTE NewPINarray[4];
NewPINarray[0]=0x1;
NewPINarray[1]=0x1;
NewPINarray[2]=0x1;
NewPINarray[3]=0x1;
BYTEArray NewPIN;
NewPIN.Value = NewPINarray;
NewPIN.Length = 4;
// Call the PIN change function
state =  MW_ChangePIN(handle, &OldPIN,&NewPIN);
If(state == 0 )
//PIN change succeeded
Else
//PIN change failed
```

5.47 Switch to Mifare Emulation

Function Signature

```
ULONG MW_MF1_SwitchToMifareEmulation(int ReaderID, _ULONG Context,_ULONG
CardHandle,LPCWSTR ReaderName);
```

Description

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.

Note:

- Cyrrently 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

MW_MF1_SwitchToMifareEmulation API switches the reader connection to Mifare Emulation application on the card, the actual implementation of this API comes in different Plugin for each reader as mention in the above paragraph.

Pre-conditions

MW_Connect function must be called to a contactless reader as CardHandle is required

Parameters

`int ReaderID`	[in] reader plug-in ID as configured in mifare.cfg or 0 to try the configured plug-ins one by one in order
`ULONG Context`	[in] a valid PCSC context returned from `MW_EstablishContext` function

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect` function
`LPCWSTR ReaderName`	[in] name of the reader to switch

Return Values

0	Switch to MIFARE failed
Positive value	new handle to MIFARE reader
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
_ULONG mifareHandle = MW_MF1_SwitchToMifareEmulation(0,
Context,cardHandle,readerName);
if (mifareHandle > 0)
// mifare emulation is active
else
// error occured
```

5.48 Is Mifare Emulation Active

Function Signature

```
long MW_MF1_IsMifareEmulationActive(_ULONG CardHandle);
```

Description

This API checks if the a specific reader is already switched to Mifare Emulation, if it returns true, then Mifare functions can be used directly otherwise MW_MF1_SwitchToMifareEmulation must be executed first.

Pre-conditions

MW_Connect function must be called as CardHandle is required

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect`Function

Return Values

1 The reader is in Mifare emulation mode
0 The reader is NOT in Mifare emulation mode

Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
long active = MW_ MF1_IsMifareEmulationActive(cardHandle);
if (active == 1)
// mifare emulation is active
else if(active == 0)
// mifare is NOT active
else
// error occured
```

5.49 Load Mifare Key

Function Signature

```
long MW_MF1_LoadKey(_ULONG CardHandle,BYTE KeyNr, BYTEArray *Key);
```

DescriptionMifare 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 loads the key to the reader memory via a standard PC/SC command..

Pre-conditions

Reader should be in Mifare emulation mode

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect` or `MW_MF1_SwitchToMifareEmulation` functions
`BYTE KeyNr`	[in] Key number to load value between 0 and 31
`BYTEArray *Key`	[in] 6 bytes holding key value

Return Values

0	Key loaded successfully
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
long status = MW_MF1_LoadKey(cardHandle, KeyNr, Key);
if (status == 0)
// key loaded
else
// error occured
```

5.50 Read Mifare Binary Data

Function Signature

```
long MW_MF1_ReadBinary(_ULONG CardHandle,BYTE BlockNr, BYTE KeyNr,
BYTE KeyType, BYTEArray *Data);
```

Description

This API allows reading the 16 bytes binary data from any Mifare data blocks.

Pre-conditions

Reader must be in Mifare emulation mode.

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect` or `MW_MF1_SwitchToMifareEmulation` functions
`BYTE BlockNr`	[in] Block number to read from, value between 0 and 63
`BYTE KeyNr`	[in] Key number, value between 0 and 31
`BYTE KeyType`	[in] Key type, value 0x60 or 0x61
`BYTEArray *Data`	[out] 16 bytes holding data block read from the card

Return Values

0	read successfully
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
BYTEArray* Data = new BYTEArray();
long status = MW_MF1_ReadBinary(cardHandle, 0, 0, 0x60, Data)
if (status == 0)
// data read successfully
else
// error occured
```

5.51 Update Mifare Binary Data

Function Signature

```
long MW_MF1_UpdateBinary(_ULONG CardHandle,BYTE BlockNr, BYTE KeyNr, BYTE KeyType, BYTEArray *Data);
```

Description

This API allows Updating the 16 bytes binary data to any Mifare data blocks.

Pre-conditions

Reader must be in Mifare emulation mode

Parameters

`ULONG CardHandle`	[in] handle to the card returned from `MW_Connect` or `MW_MF1_SwitchToMifareEmulation` functions
`BYTE BlockNr`	[in] Block number to write to, value between 0 and 63
`BYTE KeyNr`	[in] Key number, value between 0 and 31
`BYTE KeyType`	[in] Key type, value 0x60 or 0x61
`BYTEArray *Data`	[in] 16 bytes holding data to be written to card.

Return Values

0	updated successfully
Other value	an error has occurred, please refer to EIDA Toolkit Troubleshooting document for details.

Sample

```
// Initialize connection with the card to get the card handle (Handle)
long status = MW_MF1_UpdateBinary(cardHandle, 0, 0, 0x60, Data)
if (status == 0)
// data updated successfully
else
// error occured
```