SDK FOR SMARTBOX

PROGRAMMING GUIDE

AUTHOR : CHRISTOPHE RAOULT

VERSION 1.1

CREATION DATE : OCTOBER 2011

Table 1: Document Change History

Document	Author	Date
Preliminary document v.1.1	C. Raoult	2011-11-01

CONTENT

1. INTRODUCTION

1.1. PURPOSE

The SPACECODE SDK Programming guide explains client application requirements and ways to integrate the SPACECODE SmartBox (DSB). The SDK is written for Developers who create Client Applications while using the SmartBox (DSB) in USB connexion mode.

1.1. SCOPE

This library implements a Dynamically Linked Library (DLL) in .NET for Windows. The framework must be the .NET 2.0 SP2 or above.

The name of the high level main library is *SDK_SC_RFID_Devices.dll.* This library uses several attached sub-libraries -notably the low level APIs used to drive all SPACECODE devices.

This document describes the optimal way to structure an application that uses the SPACECODE SmartBox (DSB).

2. INSTALLATION

The SDK is provided with a sample project in C# and a BIN directory including the library.

Libraries include in the BIN folder must be added to the .NET project environment using the Microsoft command *Add Reference*.

Three sub-libraries must be added:

DataClass.dll: contains all the basic data class and structure (always required)

SDK_SC_Reader.dll: Low level library for RFID device (always required)

SDK_SC_Fingerprint.dll: Low level library for Fingerprint device –And the main library:

SDK_SC_RFID_Devices.dll: High level library for SPACECODE devices (always required) Two options exist to setting up the fingerprint reader:

First option:

Methods from the SDK must be used to enrol, load fingerprint templates and create a user profile? The developer will have to take care of storing and retrieving the fingerprint templates.

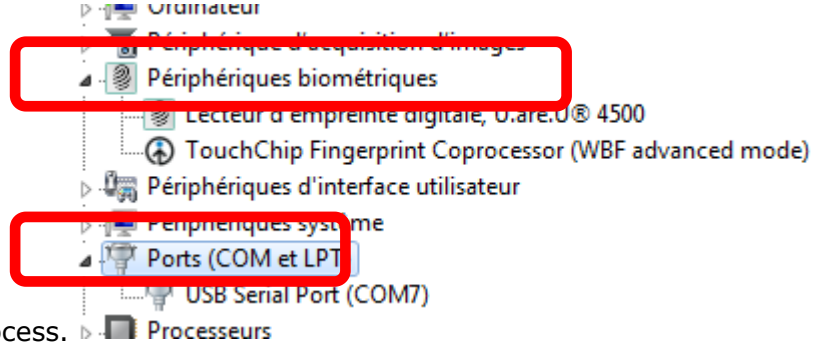
Second option:

A DemoSoft program exists to create, enrol users, create devices and manage user grants. The DemoSoft stores fingerprint templates in a SQLite database and allow an automatic load with a particular parameter when creating the device to retrieve the stored templates. In this option, the *System.Data.SQLite.dll* must be present in the folder of the application. Depending on the architecture, the x86 or x64 bits version of this particular DLL must be added.

This document describes the first option, that is the Fingerprint device management being set up by the developer without using DemoSoft.

SDK includes all mandatory drivers in the driver folder. For computer using Windows XP or not connected to internet, USB drivers are also include. The fingerprint device driver (DigitaPersona Driver) must be installed before plugging the device to the PC to prevent bad driver installation.

The Fingerprint driver is included in the FP folder. setup must be run and follow the installation process.



Drivers are correctly installed if an USB Serial port and a biometric sensor U.are.U®4500 are present in the device manager.

3. PRINCIPLE OF NOTIFICATION

All SPACECODE devices work with the same data structure. For each RFID device in use, an object is created and this object must subscribe to the notification process.

Each method request to a particular device will generate a notification event. The client application has to handle this notification to recover the device status and will be able to receive data from the device.

The notification has always the same structure

For the RFID Device, the *rfidReaderArgs* class is defined as follows:

```
private string serialNumber;
private ReaderNotify rnValue;
private string message;
```

The notification contains the serial number of the device which has fired the notification. The *rnValue* is the notification launched. The client application will have to react properly to the provided value. The message contains some info of the notification used for debugging.

3.1. NOTIFICATION LIST FOR RFID

Error notification :

RN_ReaderFailToStartScan
RN_ReaderScanTimeout
RN_ErrorDuringScan

See 5.4 for notification meaning explanation

Status notification

RN_Connected
RN_Disconnected
RN_FailedToConnect

See 5.2 for notification meaning explanation

Event notification

RN_ScanStarted
RN_ScanCompleted
RN_ScanCancelByHost
RN_TagAdded:
RN_Door_Opened
RN_Door_Closed
RN_DoorOpenTooLong

See 5.2 and 5.4 for notification meaning explanation

4. PROPERTIES

4.1. SERIALNUMBERRFID

```
string SerialNumberRFID { get; }
```

Property to retrieve the device serial number. This number is unique and allows differentiating one device among all others.

4.2. DEVICETYPEREADER

```
string DeviceTypeReader { get; }
```

Property to retrieve the device type from one of the following values:

```
public enum DeviceType
```

DT_DSB (SmartBox reader)
DT_JSC (SmartCabinet reader)
DT_MC (MedicalCabinet reader)
DT_SBR (SmartBoard reader)
DT_SAS (SmartSAS reader)
DT_Unknown

4.3. CONNECTIONSTATUS

```
ConnectionStatus ConnectionStatus { get;}
```

Property to retrieve the connection status of the device from one of the following values:

```
public enum ConnectionStatus
```

CS_Connected
CS_Disconnected
CS_InConnection
CS_Disabled

At the creation of the object, the status is *Disabled*. During object creation request, device status is *InConnection* until the result of the connection.

If the connection succeeds, device is in *Connected* status

If the connection failed, device is in *Disconnected* status, developer must rebuild the connection.

4.4. DEVICESTATUS

```
DeviceStatus DeviceStatus { get;}
```

Property to retrieve the device status from one of the following values:

```
public enum DeviceStatus
DS_NotReady,
DS_Ready,
DS_InScan,
DS_InError,
```

4.5. GET_RFID_DEVICE

```
rfidReader get_RFID_Device { get ; }
```

Property to retrieve the low level RFID object

4.6. LASTSCANRESULT

```
InventoryData LastScanResult { get; }
```

Property to retrieve the last scan result.
The value is available after having received an *RN_ScanCompleted* or *RN_ScanCancelByHost* notification:
The structure of the data is as follow:

```
public string serialNumberDevice = null;
public DateTime eventDate = DateTime.Now;
public bool bUserScan = false;
public string userFirstName = null;
public string userLastName = null;
public int nbTagAll = 0;
public int nbTagPresent = 0;
public int nbTagAdded = 0;
public int nbTagRemoved = 0;
public ArrayList listTagAll = new ArrayList();
public ArrayList listTagPresent = new ArrayList();
public ArrayList listTagAdded = new ArrayList();
public ArrayList listTagRemoved = new ArrayList();
```

- SerialNumberDevice : the serial number of the device which perform the scan
- eventDate : Date and hour of the scan
- bUserScan : if true the scan was launch with the fingerprint if false the scan comes from the scan device method.
- userFirstName : null if bUserScan to false or first name of the user.
- userLastName : null if bUserScan to false or last name of the user.
- nbTagAll : Number of tag inside the volume or area.
- nbTagPresent : number of tag that was already present from the previous scan.
- nbTagAdded : number of tag that was already added from the previous scan.
- nbTagemoved : number of tag that was already removed from the previous scan.

- listTagAll : list of tag UID of all the tag inside the active volume or area
- listTagPresent : list of tag UID that was already present from the previous scan.
- listTagAdded : list of tag UID that was already added from the previous scan.
- listTagRemoved : list of tag UID that was already removed from the previous scan.

4.7. SETPREVIOUSSCAN

```
InventoryData setPreviousScan { set; }
```

Property to set the previous scan. Used when creating a new object to load the last scan in order to keep data consistency.

4.8. GETFPMASTER

```
FingerPrintClass get_FP_Master { get;  }
```

Property to retrieve the object of the fingerprint device.
This property is use when use the *loadFingerPrintTemplate* method to load the templates to the right fingerprint.

4.9. FPSTATUSMASTER

```
FPStatus FPStatusMaster { get ;}
```

Property to retrieve the fingerprint status from one of the following values:

```
public enum FPStatus
        FS_Ready,
        FS_Disconnected,
        FS_FingerTouch,
        FS_FingerRemove,
        FS_CaptureComplete,
        FS_UnknownUser,
```

4.10. LIGHT LUMINOSITY PROPERTIES

```
ushort LightInIdle { get; set; }
ushort LightInScan { get; set; }
ushort LightDoorOpen { get; set; }
```

These three properties stand to get and set the light luminosity value.
The three values are available when the device waits for an action, when the device is in scan and when the device's door is open.
The value can be set from 0 to 300 in relation to the desired luminosity. (0 is off and 300 is maximum luminosity)
The default values are the following:

lightInIdle = 150 – when device wait for an action - Medium luminosity
lightInScan = 0 - when device is in Scan – light switch off

lightDoorOpen = 300 – when the device' door is open – Maximum luminosity

4.11. DOOR TIMER PROPERTIES

Two properties exist to characterize the door behaviour.

```
int TimeBeforeCloseLock { get; set; }
int TimeDoorOpenTooLong { get; set; }
```

The *TimeBeforeCloseLock* defines a time limit (in seconds) for which the lock mechanism will be activated again if the door hasn't been opened

The *TimeDoorOpenTooLong* is a time limit (in seconds) for witch the door can stay opened before notification.
After this time limit the notification *RN_DoorOpenTooLong* is fired letting the developer the possibility to inform user or to generate an alert.

4.12. INTERRUPTSCANWITHFP

```
bool InterruptScanWithFP { get; set;}
```
This property allows choosing to stop a running scan process by an authorized user.
If true, a user can stop the scan running and the door will be opened.
If false, the door will remain closed and a notification *RN_ReaderNotReady* is fired.
The user will have to rescan its finger to open the door.
The default value is true.

5. METHOD

5.1. GETRFIDPLUGGEDDEVICE

Definition : `rfidPluggedInfo[] getRFIDpluggedDevice();`

Method to retrieve an array of the device plugged on the PC.

The Method returns an array `rfidPluggedInfo` defined as follow:

```
public class rfidPluggedInfo
    {
        public DeviceType deviceType;
        public string SerialRFID;
        public string portCom;
    }
```

This array can be used for the developer to select the appropriate method for creating the device.

For the SmartBox (DSB), the *Create_1FP_Device* method is used.

Example of use:

```
RFID_Device tmp = new RFID_Device();
rfidPluggedInfo[] arrayOfPluggedDevice = tmp.getRFIDpluggedDevice();
tmp.ReleaseDevice();
```

5.2. CREATE_1FP_DEVICE

These two methods create the SmartBox (DSB) device. If the serial port is known, it is better to use the second method since it will directly connect to the right device.

The first method will search on all serial ports (Bluetooth also) if the device is present. This search can take several seconds if many serial ports are present on the PC.

Definitions :

```
bool Create_1FP_Device(string serialNumberRFID,string
serialNumberFP_Master,bool bLoadTemplateFromDB);
bool Create_1FP_Device(string serialNumberRFID,string portCom,string
serialNumberFP_Master, bool bLoadTemplateFromDB);
```

These methods will create the object and attempt to connect to it.

If the Developer knows the serial port or if it was retrieved by the *getRFIDpluggedDevice* method, the serial port name can be passed as an argument to speed up the connection.

The parameter(s) are a string containing the serial number of the RFID device, a string containing the serial number of the fingerprint and a Boolean to use or not the fingerprint templates from the SQLite Database. **This parameter must be set to false.**

This function returns true if the device is created.

The result of the connection will be received by the notification process later.

For the RFID the function can generate an *RN_Connected* if succeeded or *RN_FailedToConnect* if failed.

For the fingerprint the function can generate a *RN_FingerprintConnect* if succeeded and *RN_FingerprintDisconnect* if failed.

If the connection succeeds, the device is ready, in case of failure in connection this object has to be released by the Developer and try to reconnect later by recreating this object.

After a successful connection of the fingerprint and the RFID, the device works in standalone.

Developer will have to upload the authorized fingerprint templates of each user.

After this upload, an *RN_FingerTouch* is fired if the fingerprint device is touched and an *RN_FingerGone* as soon as the finger is removed from the sensor surface.

These two notifications inform the event from the user.
When the capture is complete, two cases may arise:

- If the finger is unknown an *RN_FingerUserUnknown* is notified and the device returns in waiting mode.
- If the finger is known an *RN_AuthentificationCompleted* is notified. The message contains the first name of the user , his last name and the finger number that is recognized (see definition of template of a user)

  This later authentication automatically unlocks the lock.

If the user opens the door before the delay of the property *TimeBeforeCloseLock* the *RN_Door_Opened* notification is fired and the lock mechanism is locked. If the time limit elapsed before the door is open, the device locks the lock and goes back in waiting mode.

If the user doesn't close the door before the time defined in *TimeDoorOpenTooLong* elapsed a notification *RN_DoorOpenTooLong* is fired. This notification stands for informing the user to close the door or for generating an alarm by e-mail.

When the door is closed, the *RN_Door_Closed* is notified and the scan is launched.
After this point the process follows the same as defined in the *ScanDevice* method to notify the scan status and to recover the tag UID.
At the end of the scan method, the device will return in waiting mode.

See an example of code at the end of this document.

5.3. RELEASEDEVICE

Definition :

```
void ReleaseDevice();
```

Function to close a device. This function will disconnect the reader and the fingerprint and dispose all the objects used.

The following methods are not intended to be used in normal mode. It just to let the developer the choice to control the device if needed.

5.4. SCANDEVICE

Definition :

```
bool ScanDevice();
```

The *ScanDevice* function launches a scan for the SmartBox (DSB).
This method can be used to request an auto scan inventory without any action on the fingerprint.
The function will return true if the scan command can be launched (that is if the device is ready and connected). The function output will not tell if the scan is effectively performed. This info will be available later through the notification process.

The Developer has the choice to recover the tags inside the volume dynamically by catching the notification *RN_TagAdded* or to wait for the end of the scan process to recover the full inventory data by the property *LastScanResult*.

The method generates several notifications:

RN_ScanStarted when the inventory succeeds to start.

RN_ReaderNotReady when the reader is processing another task and cannot begin the inventory.

RN_ReaderFailToStartScan when the reader cannot start the scan for an unknown reason.

RN_ReaderScanTimeout when the time defined in the timeout property has expired before the end of the scan in progress.

RN_ErrorDuringScan when errors are present in the inventory. The data received before this notification might be corrupted bad or it can miss some tag. This notification is raised when excessive electromagnetic noise is present near the reader preventing to finish the inventory

RN_ScanCancelByHost when the Host has requested the end of the inventory in progress.

RN_ScanCompleted when inventory is finished with no issue.

RN_TagAdded when a new tag is found, the tag code is given inside the variable message of the *rfidReaderArgs* class received by the notification.

5.5. STOPSCAN

Definition :

```
bool StopScan();
```

Method to stop the inventory in progress.

The method returns true if the request is successively send.

This method generates a notification *RN_ScanCancelByHost* when the host has processed the end of the inventory in progress.

5.6. UNLOCK

```
bool UnLock();
```
Method to request to unlock the door mechanism.
The door generates two notifications *RN_Door_Opened* and *RN_Door_Closed* when an event occurs on the door

5.7. LOCK

```
bool Lock();
```
Method to request to lock the door mechanism.
The door generates two notifications *RN_Door_Opened* and *RN_Door_Closed* when event occurs on the door

5.8. SETLIGHT

```
bool SetLight(ushort power);
```

Method to set the LEDs illuminating the active volume. For instance the method can be used to lighting up when the door is open. The function takes as parameter a value between 0 to 300 to let the user choose the intensity he desires.
This function is unavailable during the scan.

The following two methods stand to create a user profile.

5.9. ENROLLUSER

Methods used to enrol a new user. The user cannot have identical first and last name.
The method will display a windows form to completely add, modify or delete finger template for a particular user.
This function will return a binary stream that contains all the information of the user.
The developer will have to store the data in an appropriate way (file, database …)

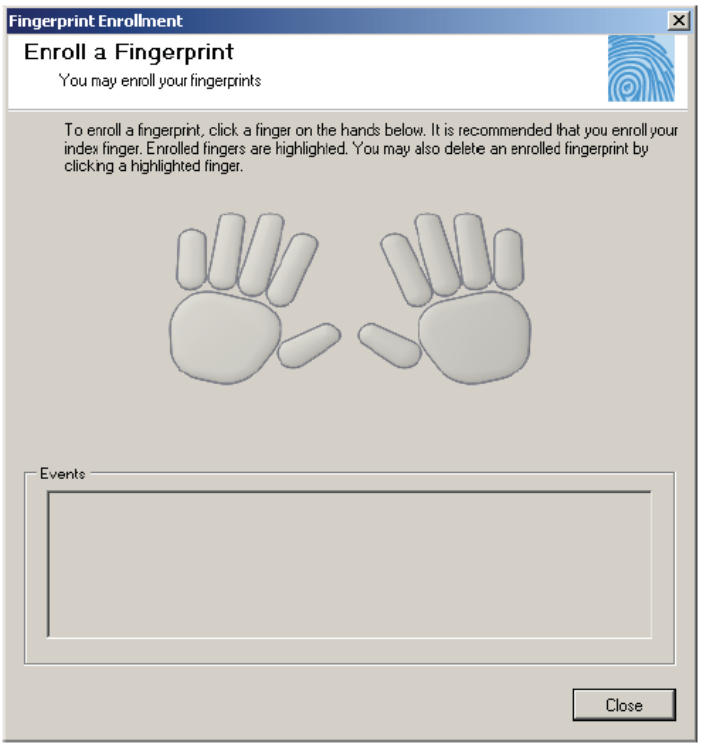
Definition :
string EnrollUser(string FPSerialNumber, string FirstName, string LastName, string template);

Parameters :

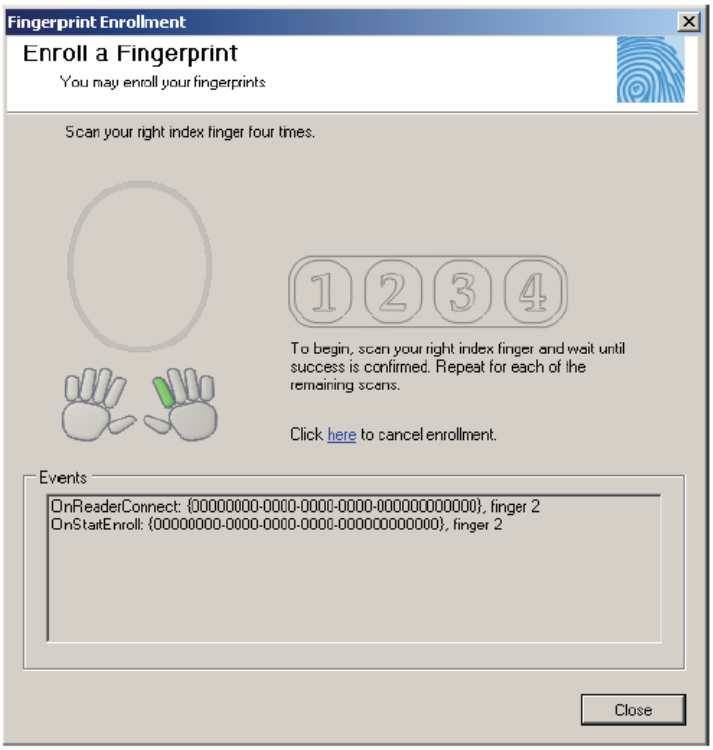
string FPSerialNumber : The serial number of the fingerprint device or null for using any fingerprint connected to the PC
string FirstName : FirstName of the user
string LastName : LastName of the user
string template : Previous template from a previous enrollment of this user if modification required. Null if a new one. ??

The output Parameter is the binary stream to store.

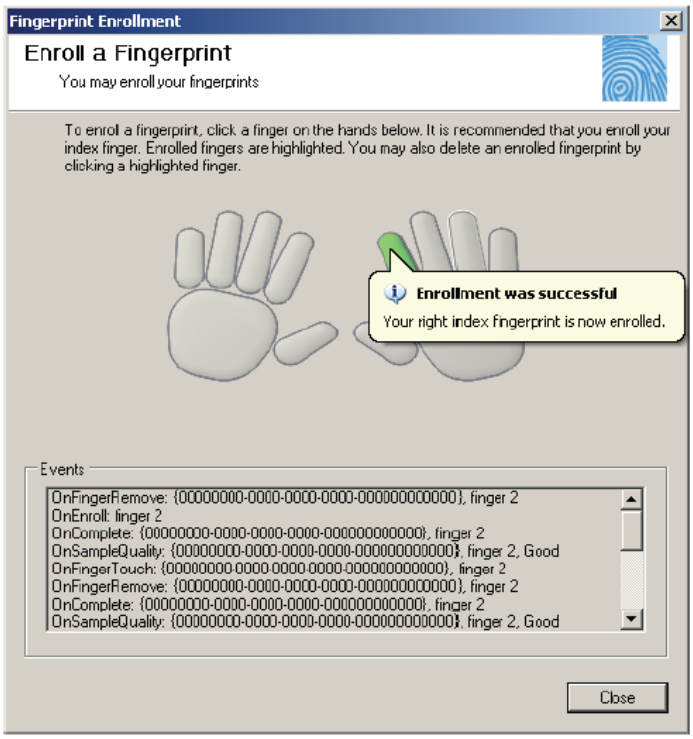
After the method is sent, the Fingerprint Enrolment dialog box appears as shown below:



Select the fingerprint to enrol. If the finger spot is green, the finger is already enrolled and a message to delete it will be displayed. If not the second Fingerprint Enrolment dialog appears



Using the fingerprint reader, present the finger you choose to register.
Repeat this step until the "Enrolment was successful" message appears.



When clicking the close button, the dialog is closed and the function returns the binary stream of the user.
This binary stream has to be saved by the developer as it is the template to provide back to the fingerprint device in order to authorize access to a given user.

5.10. LOADFPTEMPLATE

Definition :

int LoadFPTemplate(string[] templates, FingerPrintClass theFP);

Parameters :

string[] templates : array of stream from the enrol method of the entire user that has to be authorized to open a particular device.

FingerPrintClass theFP: Fingerprint object of the device that will accept the user in the array.

The function returns an integer of the number of templates successfully loaded. This value should be the number of the members in the array.

The method has to be used to load user templates after each creation of a device, or when the user array needs to be updated.

The array must always contain all the authorized users. If two load processes are requested successively, only the templates of the second try remain active.

Note: the template is a binary stream of several KB for one user

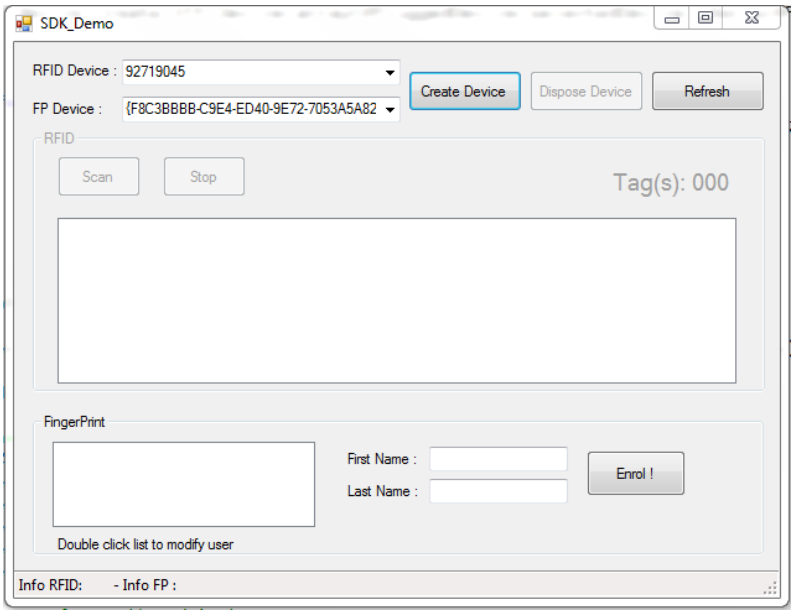
Example of binary for one user and one finger:

AAEAAAD/////AQAAAAAAAAAAMAgAAAE9EYXRhQ2xhc3MsIFZlcnNpb249MS4wLjkxMC4yNCwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNLZXlUb2tlbj03Zjg4M2YzMmM0YTIyODEyBQEAAAAbRGF0YUNsYXNzLlVzZXJDbGFzc1RlbXBsYXRlBQAAAAlma
XJzdE5hbWUIbGFzdE5hbWUIdGVtcGxhdGUQaXNGaW5nZXJFbnJvbGxlZA1CYWRnZVJlYWRlcklEAQEBBwEBAgAAAAYDAAAACkNocmlzdG9waGUGBAAAAAZSYW91bHQGBQAAALAZQUFFQUFBRC8vLy8vQVFBQUFBQUFBQUFNQWdBQUFFMVRSRXR
mVTBOZlJtbHVaMlZ5Y0hKcGJuUXNJRlpsY25OcGIyNDlNaTQxTGpreE1DNDJOek1zSUVOMWJIUjFjbVU5Ym1WMWRISmhiQ3dnVUhWaWJHbGpTMlY1Vkc5clpXNDlibIZzYkFVQkFBQUFIRk5FUzE5VFExOUdhVzVuWlhKd2NtbHVkQzVWYzJWeVEyeGhj
M01FQUFBQUNXWnBjbk4wVG1GdFpRaHNZWE4wVG1GdFpRNXpkSEpHYVc1blpYSndjbWx1ZEJORmJuSnZiR3hsWkVacGJtZGxjbk5OWVhOckFRRUdBQWdDQUFBQUJnTUFBQUFLUTJoeWFYTjBiM0JvWlFZRUFBQUFCbEpoYjNWc2RBa0ZBQUFBUUFB
QUFCRUZBQUFBQ2dBQUFBb0dCZ0FBQUlBUlFWQnBRa0ZqWjNFME0wNWpkMFZGTTBOaGRIaHpTa0ZWVmxwTWFrTnBTRzh6VTFreWVHOU5TakkyUzFSTVVYbzFZVVZ2YVhoWk5IQXlNMmhOUjFFelQySlBSa1k1UkdzNGJFNXRaMk5LYzFaNlpEQlFh
MU0xTjBWdlJIaDZXbVJMU0VwVVkwRmFlRWhQTldaUmFXYzBRbkJFV0VSQ05FVnlVWGxSYkZGSVRISmFaWFpqU2xjNWJXdEVVR2hOY0U4dmRIa3lWWGxIV1VreldjcEhaRlZHV2tzM1UyNHZVellyTlROa2VrVklWekp0SzJ4NWFsRjFRM0ZYVHhOUmVU
SlJZbG95YlZkdlJHdERPVE5uVVZCcVNXWlFaSEpIV2tKd00wWnhaMGRIZGpOS1NFUnlVbFZ5TUdSaWJrZFFSV1ZUUjA1UFEzbFZXRlZRZW05cVRURm5iV3hKUlRSNFlYZHRjbEZyUldwVEZaHdRM0oxYVZCVlpsWkhPR1pHY0VObWF6ZPWEJFTUdoS
2VuQjNTVlJpY0haVFZpOXBWMjF5VEZONmJFRXJRU3QyVVV0clRsRndlbkZSWlU5RmFtc3hhek5uVUVWeVkwdHpaRXBhTkdGcWQyUTJaMnRxVVZWc01VTlFSXhxUWtGc1RUWnNXbVp0WWpsc1MwaDRiWFJLWlZSMVkyaG9OV2R6VERSkeU0yNWlTR
GxEZEVSdk5qZ3pPVGRZZHpsTVNqWXpNRGgzTmxBMVVVYVjBkM0JxZDNOd1kyMUpaV0pRY0ZZKMWFuSm1iVUp1TjFCV2RGWlFPQzlRTTFWRGEzZFBibTU1WVM5MlFUVnlaaXN6ZFVWamFWUk1aVlIzVmpoU00wSktTVlpaT0hwcGRFdEVlVGs1SzNOWl
NtaEVhR0k1YnpjMlpWVTFZbTVPZGtGUWFVRkJZMmR4TkROT1kzZEZSVE5EWVhSNFkwcDNWVlphVEc1a01VVXljRWRDTTFWREVwMTJVMjVuU0M5M1RESXdWMEZwZUdJeVJVeEVkR053YUhwS0wyRTFVbEp3VTBOcGEVwVTVOM0ZOTnpCek1XaFdh
VTlTYzJKMFVVaEJhREIwYVhOM1VYZGtiM0V3VEZSQ2RGQTRVMDhyZGtWYWVXZDNhVkZUZFNU0zZHFaVEI2VlZoMVFXcEdjRGsyWVVkdlZreFdjbXhNYTNJd1QycHhSVkIzY1RCWkswTXhUa3hTY0VOak1VcFFaSFZTVW5jMVIxZ3JObkpLUkdaVkw
wUnlXRXBpYTBGS1ZHcFNUV2RzZGsxYWJFTk9TR1JNTmpGdGMxbzJSV292VDFaTlZFbDFVMkZ6VldscVFURTFNbUpMV1RGVmRXNVpVEoyZFVsMGFrbDRaelIxVTFnck0ybE1OamcyUlZFdmFqUkVlRVJNUzFsa01FbGljbEJRVFVFV0SGRrdHNReklyYURW
MlNEWlJkbGRVYkdaKaGQwbDJaR1IxTldGSmJITkJOVU5JYUV4MFRqUkxkWFJpU0ZaWE5GVkNNbmgwUTJoRlZtTmpZVkpQVG1oSmVUZG9Za3RDUm1wVk5EVmFabUZsVVdsV2NubEtWVXA0T0c1MlVERkpLM1p3U0dZM1FYcFlIRVkwWTFBemVFZHVN
a2d3YWpCYVlrVkdZa3dyVjBwWWIwbE5jM0pqZFVvM2VUa3hTWE5UUm5GNlIzcERjazFRZG0wNGFreDBZMWx4T1c1d2JWcGxUM3BVVmtrdk1sazRia3BuV2pWS1ZtdG9TRXh0Y1dacFpWUkxTRzk1UzNWVFJ6VkNNR3MxUlhobGFpdHViWGNyZDJaW
GEzQjRSMDlWTmpSNk56bEhPRUVyU0RSQ2VVTnlhbU14ZWtGUlZHTktjjVE5IZDJ4NFVsWnJjMjAzVUhSNGJEUkZOR3R1VmpjeVprMUphMHcyY1dKdVpqSmFXVGN4YUZOSGJFaEVWalZFUzFOcGFXbEtOR1pUZUhnelFYSjNOM0JUUTFjMGFURkhMMVp
IVkdaS2NFODNOWEpKVlZneFYyaDZWbEpvVlRWaFFUTm1RWGRJTW5CWVJ6WkJkMVUyU0U5NVNIUTRWV3h0U1hJNVVFaDZPVVJSUTFkVGJIZEZUV0ZPYTNGVWExTXdibk5zSzFWWFVrcVhM0EwZUhwUFpFUXZaV0l5UjNoV2JXMVdkMnRxU1RjMm
RuTlVlbmRQTkRoRk5lIlYSlRTbEpHTTB0Wlp5OUNhbWM0YlRsek5tdHNPR1JyU21KKUmNVazJNRloyWTBsVFMyWldVMWRXVHpCMloxcHhSblV3Y1VFelFWQnhUWE5JV2pobVJFNTZXU3R0U1doakswNTNabkJVY1ZwMFVHZEdNRWsxYzNoQmRHMUNORU5
EUTB0TFFXeHBiR3d4Y1ZFek5UQm5kRkpaUzBsc1dFb3lXa3BFWkVaaFJFSTJSMWxpV1dKNU9XMVVTMDFuU1hKTlJFbE9jMjlzUzBGQldqaHlWM05hU3pKWlNIVnBjMHBYZW0xbU1FVlRSbGRvTW5GUWQxUXZaM2xVVkdzMmNIWjFWM1pSUkhhSlZtO
UVjMFp6UlhoV2NtVk5VRmcwVlZCVFpaVEa1TR3R3ZFRoU1VVMXZlVFDUmxwTVZ6QkROakJxUzFnMFlXWldOMlE1VkZkeVJVTlVWRFk1YlZRME1FWXpXa3hQTW5WRFVWWWkVOM0JoU1dWT1MwOUNRVEJpZVdKck1XNWlSVmxLVVZaaVNHSTBNRlI
wWWpObIZXRjVjWEl5T0VFVFMlNEaENlVU55YW1NeGVrRlJWR05LY1ROaVJWJWR05LY1RORmQyMTRVbFpyY1dOS05VSkZSRnBRVlZKUFlrZFZXbFZXVFRlWm9WREJsZHpHcWQ2NpdGxXR1ZGUjFGMlV6UndZVzA0TTdsdmFGa3djjMjVqYVdZME4yeczVkUzlKY1VJd1FuZENUbEZ3TVU1RGNG
QXZUMGRwYUVzdmJUSmxMekZZTTNOcFlrRlVPRzExYTFWRVFGVGWnJNVFJQYWxrelVrMTZWbVZQY0hwMFR5dDBjMVZPYkhKalpteG5VRFp6WTNwb2VGZFhSbWRhYkUxVVZUVk5VVmRPVUVSNE9WbElhMDB2YlVaaEwyVlpabXhhHTVhCc05YRkNRMGc
1WW1ObE5tMWhUWFpTUzFjNWMza3JiVkZpTlZGdVpFWjJNRVo2YW1oelJ6WmhNWEJVYmtaNFIzTnBabEZWTnpGNWRIcGlhREJvTW1GTWRtWXhVa2gyUmtKaE1WTktjMjluZDFKeFRc3pVSEpqWXk5aU5WQkxWelZvV0c1cU1Tc3dURlZNVm1ORFR
HWXhPVlZsY25WSlRtSTNXRE5EWkZSaFpFbFBRVkU0VEROdlZYY3lVa3g1Ym5odFkwSmFUSFptWldZMVpsVlBiVGsyYm5adlRqTlBhRTFtVGxOd1NreHRha1JoT1daMGNFazRUSFpoWm5GYWJrMUJiVlpKUXpOeVVFNHdibEIzWlRWRGJubG1aME01V0Ux
cFNFTlVUbnBqWjFCUE9YSkJOR1pVVFZVeU1qTkpUREpZTVdGRFVYZDFUa2R1U2xSS05EQlFUMGQ0Wm5FeFRtOXpiM2d3WnlzNFlrRlphRmxRUjJOVFVUZzNhMFZKTkdKVFYxcFRaMHcwU2tseE1VRndkR3dyYzJ0emRsTXZiR3AxT0dvMVFVVm5WSEZ1
YW01alFuWkJRVUZCUVVGQlFVRkJRVUZCUVVGQlFVRkJRVUZCUVVGQlFVRkJRVUZCUVVGQlFVRkJRVUZCUVVGQlFVRkJRVUZCUVVGQlFVRkJRVUZCUVVGQlFVRkJRVUZCUVVGQlFVRkJRVUZCUVVGQlFVRkJRVUZCUVVGQlFVR
kJRVUZCUVEwSUN3PT0JBgAAAAYHAAAACkIxMDE1NDI0QTAPBgAAAAoAAAABAAEAAAAAAAAAAAs=

6. DEMO APPLICATION

6.1. APPLICATION

A small sample application is provided to show the code implementation.



To use this sample application you have to plug the device and launch the application.

If only one device is present, powered and ready, the serial numbers of the RFID device and the fingerprint will appear in the combo box. If several are present, take care to choose the pair printed on your device (serial number on product sticker).

Select the device and related fingerprint and click on *Create Device.* If the device succeeds to connect, the control beneath will be enabled. The status will be displayed on the status bar.

If no devices are found, try by clicking refresh or check if the device is supplied and if the USB cable is plugged. Check also in the device manager that the fingerprint and the USB converter are present and installed.

To perform a scan, click on *scan*.
To interrupt a scan click on stopYou can click on *stop* to interrupt a running scan.

If Tags are present inside the volume, their UID (Unique Identifier) will be display in the Listbox.

You can register an user by entering a first Name and a Last Name and click on Enroll. These two fields are mandatory. You will see the Enrollment process described in the *EnrollUser* method.

If the Enrollment succeeds, the name of the user is added in the Listbox and the finger registered can be presented on the sensor. If the finger is recognized, the door will be unlocked.

If the user opens the door, the device will wait until the door is closed to launch the scan.

To modify this user, double click on it in the Listbox.

Caution: This demo application does not store the template so that a new enrollment has to be performed on each restart of the application.

6.2. SOURCE CODE

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;

using SDK_SC_RFID_Devices;
using DataClass;

namespace SDK_RFID_Sample
{
    public partial class SDK_Demo : Form
    {

        // define max user for array size - No limitation of size.
        private const int nbMaxUser = 100;
        private int indexUser = 0;
        UserClassTemplate[] userArray = new UserClassTemplate[nbMaxUser];

        rfidPluggedInfo[] arrayOfPluggedDevice = null;
        string[] fpDevArray = null;
        int selectedDevice = 0;
        int selectedFP = 0;

        //General device object
        private RFID_Device device = null;

        public SDK_Demo()
        {
            InitializeComponent();
        }

        //Function to discover device
        //This function will search all the RFID device and fingerprint plugged on the PC
        //Take care to choose the good value depending of the serial numbers printed on the device.
        private void FindDevice()
        {
            arrayOfPluggedDevice = null;
            RFID_Device tmp = new RFID_Device();
            arrayOfPluggedDevice = tmp.getRFIDpluggedDevice();
            fpDevArray = tmp.getFingerprintPluggedGUID();
            tmp.ReleaseDevice();
            comboBoxDevice.Items.Clear();
            if (arrayOfPluggedDevice != null)
            {
                foreach (rfidPluggedInfo dev in arrayOfPluggedDevice)
                {
                    comboBoxDevice.Items.Add(dev.SerialRFID);
                }
            }

            comboBoxFP.Items.Clear();
            if (fpDevArray != null)
            {
                foreach (string fpDev in fpDevArray)
                {
                    comboBoxFP.Items.Add(fpDev);
                }
            }

            if ((comboBoxDevice.Items.Count > 0) && (comboBoxFP.Items.Count > 0))
            {
                comboBoxDevice.SelectedIndex = 0;
                comboBoxFP.SelectedIndex = 0;
                buttonCreate.Enabled = true;
            }
            else
            {
                buttonCreate.Enabled = false;
                Invoke((MethodInvoker)delegate { toolStripStatusLabelInfo.Text = "Info : No RFID and/or
Fingerprint detected - try Refresh"; });
            }
```

```
        }


        // Button create device
        private void button1_Click(object sender, EventArgs e)
        {
            // release previous object if not connected
            if (device != null)
            {
                if (device.ConnectionStatus != ConnectionStatus.CS_Connected)
                    device.ReleaseDevice();
            }
            //Create a new object
            device = new RFID_Device();
            toolStripStatusLabelInfo.Text = "Info RFID : In Connection ";
            buttonCreate.Enabled = false;
            //subscribe the event
            device.NotifyRFIDEvent += new NotifyHandlerRFIDDelegate(rfidDev_NotifyRFIDEvent);
            device.NotifyFPEvent += new NotifyHandlerFPDelegate(rfidDev_NotifyFPEvent);
            //Create a DSB device
            //As the function search on all the serial port of the PC, this connection can
            //take some time and is under a thread pool to avoid freeze of the GUI
            ThreadPool.QueueUserWorkItem(
                delegate
                {

                    switch (arrayOfPluggedDevice[selectedDevice].deviceType)
                    {
                        case DeviceType.DT_DSB:
                            //  Use create with portcom parameter for speed connection (doesn't search again
the reader at is is previouly done;
                            //  recover guid FP in devFParray
                            //  bLoadTemplateFromDB mandatory to false
                            device.Create_1FP_Device(arrayOfPluggedDevice[selectedDevice].SerialRFID,
arrayOfPluggedDevice[selectedDevice].portCom,fpDevArray[selectedFP],false);

                            break;

                        default:
                            MessageBox.Show("Device not created - Only SmartBoard (SBR) allowed");
                                break;
                    }

                });

        }
        // button dispose
        private void buttonDispose_Click(object sender, EventArgs e)
        {
            if (device == null) return;
            if (device.ConnectionStatus == ConnectionStatus.CS_Connected)
                device.ReleaseDevice();
            buttonCreate.Enabled = true;
        }
        // Function to get rfid event
        private void rfidDev_NotifyRFIDEvent(object sender, SDK_SC_RfidReader.rfidReaderArgs args)
        {
            switch (args.RN_Value)
            {
                // Event when failed to connect
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_FailedToConnect:
                     Invoke((MethodInvoker)delegate  {toolStripStatusLabelInfo.Text  =  "Info  RFID  :  Failed  to
Connect";});
                     Invoke((MethodInvoker)delegate { buttonCreate.Enabled = true; });
                     Invoke((MethodInvoker)delegate { buttonDispose.Enabled = false; });
                     Invoke((MethodInvoker)delegate { groupBoxCtrl.Enabled = false; });
                     Invoke((MethodInvoker)delegate { groupBoxFP.Enabled = false; });
                break;
                // Event when release the object
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_Disconnected:
                     Invoke((MethodInvoker)delegate  {toolStripStatusLabelInfo.Text  =  "Info  RFID  :  Device
Disconnected";});
                     Invoke((MethodInvoker)delegate { buttonCreate.Enabled = true; });
                     Invoke((MethodInvoker)delegate { buttonDispose.Enabled = false; });
                     Invoke((MethodInvoker)delegate { groupBoxCtrl.Enabled = false; });
                     Invoke((MethodInvoker)delegate { groupBoxFP.Enabled = false; });
                break;

                //Event when device is connected
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_Connected:
                     Invoke((MethodInvoker)delegate  {toolStripStatusLabelInfo.Text  =  "Info  RFID  :  Device
Connected";});
                     Invoke((MethodInvoker)delegate { buttonCreate.Enabled = false; });
                     Invoke((MethodInvoker)delegate { buttonDispose.Enabled = true; });
```

```
                    Invoke((MethodInvoker)delegate { groupBoxCtrl.Enabled = true; });
                    Invoke((MethodInvoker)delegate { groupBoxFP.Enabled = true; });
                break;

                // Event when scan started
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_ScanStarted:

                     Invoke((MethodInvoker)delegate { buttonScan.Enabled = false; });
                     Invoke((MethodInvoker)delegate { buttonStop.Enabled = true; });
                     Invoke((MethodInvoker)delegate  {  toolStripStatusLabelInfo.Text  =  "Info  RFID  :  Scan
Started"; });
                     listBoxTag.Invoke((MethodInvoker)delegate { listBoxTag.Items.Clear(); });
                     labelInventoryTagCount.Invoke((MethodInvoker)delegate  {  labelInventoryTagCount.Text  =
"Tag(s): 0"; });
                break;

                //event when fail to start scan
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_ReaderFailToStartScan:
                    Invoke((MethodInvoker)delegate { buttonScan.Enabled = true; });
                    Invoke((MethodInvoker)delegate { buttonStop.Enabled = false; });
                    Invoke((MethodInvoker)delegate  {  toolStripStatusLabelInfo.Text  =  "Info  RFID:  Failed  to
start scan"; });
                break;

                //event when a new tag is identify
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_TagAdded:
                    listBoxTag.Invoke((MethodInvoker)delegate { listBoxTag.Items.Add(args.Message); });
                    labelInventoryTagCount.Invoke((MethodInvoker)delegate  {  labelInventoryTagCount.Text  =
"Tag(s) : " + listBoxTag.Items.Count.ToString("000"); });
                break;

                // Event when scan completed
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_ScanCompleted:
                    Invoke((MethodInvoker)delegate { buttonScan.Enabled = true; });
                    Invoke((MethodInvoker)delegate { buttonStop.Enabled = false; });
                    Invoke((MethodInvoker)delegate  {  toolStripStatusLabelInfo.Text  =  "Info  RFID:  Scan
Completed"; });
                break;

                 //error when error during scan
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_ReaderScanTimeout:
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_ErrorDuringScan:
                    Invoke((MethodInvoker)delegate { buttonScan.Enabled = true; });
                    Invoke((MethodInvoker)delegate { buttonStop.Enabled = false; });
                    Invoke((MethodInvoker)delegate  {  toolStripStatusLabelInfo.Text  =  "Info  RFID:  Scan  has
error"; });
                break;
                // Scan cancel by user
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_ScanCancelByHost:
                    Invoke((MethodInvoker)delegate { buttonScan.Enabled = true; });
                    Invoke((MethodInvoker)delegate { buttonStop.Enabled = false; });
                    Invoke((MethodInvoker)delegate { toolStripStatusLabelInfo.Text = "Info RFID : Scan cancel
by host"; });
                break;
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_Door_Opened:
                        Invoke((MethodInvoker)delegate  {  toolStripStatusLabelInfo.Text  =  "Info  RFID  :  Door
Open"; });
                break;
                case SDK_SC_RfidReader.rfidReaderArgs.ReaderNotify.RN_Door_Closed:
                Invoke((MethodInvoker)delegate { toolStripStatusLabelInfo.Text = "Info RFID : Door Close"; });
                break;

            }
            Application.DoEvents();
        }
        //function to get FP event
        private void rfidDev_NotifyFPEvent(object sender, SDK_SC_Fingerprint.FingerArgs args)
        {
           switch(args.RN_Value)
           {
               case SDK_SC_Fingerprint.FingerArgs.FingerNotify.RN_FingerprintConnect:

                    Invoke((MethodInvoker)delegate  {toolStripStatusLabelFP.Text  =  "            -  Info  FP  :
Connected";});
                    break;

               case SDK_SC_Fingerprint.FingerArgs.FingerNotify.RN_FingerprintDisconnect:
                   Invoke((MethodInvoker)delegate  {toolStripStatusLabelFP.Text  =  "            -  Info  FP  :
Disonnected";});
                    break;
               case SDK_SC_Fingerprint.FingerArgs.FingerNotify.RN_FingerTouch:
                   Invoke((MethodInvoker)delegate {toolStripStatusLabelFP.Text = "         - Info FP : Sensor
touched";});
                    break;
```

```
                case SDK_SC_Fingerprint.FingerArgs.FingerNotify.RN_FingerGone:
                    Invoke((MethodInvoker)delegate { toolStripStatusLabelFP.Text = "        - Info FP : Finger
Remove"; });
                    break;
                case SDK_SC_Fingerprint.FingerArgs.FingerNotify.RN_FingerUserUnknown:
                    Invoke((MethodInvoker)delegate { toolStripStatusLabelFP.Text = "            - Info FP :
Unknown user"; });
                    break;
                case SDK_SC_Fingerprint.FingerArgs.FingerNotify.RN_AuthentificationCompleted:

                    string[] strUser = args.Message.Split(';');
                    string FirstName = strUser[0];
                    string LastName = strUser[1];
                    SDK_SC_Fingerprint.FingerIndexValue                    fingerUsed                    =
(SDK_SC_Fingerprint.FingerIndexValue)int.Parse(strUser[2]);
                    string userInfo = FirstName + " " + LastName + "(" + fingerUsed.ToString() + ")";
                    Invoke((MethodInvoker)delegate { toolStripStatusLabelFP.Text = "        - Info FP : User -
" + userInfo; });
                    System.Threading.Thread.Sleep(1000);
                    break;
            }
        }

        // button Scan
        private void buttonScan_Click(object sender, EventArgs e)
        {
            if (device == null) return;
            if ((device.ConnectionStatus == ConnectionStatus.CS_Connected) &&
                (device.DeviceStatus == DeviceStatus.DS_Ready))
            {
                //Request a scan
                //Scan status will be notified by event
                device.ScanDevice();
            }
            else
                MessageBox.Show("Device not ready or not connected");
        }

        //Button Stop
        private void buttonStop_Click(object sender, EventArgs e)
        {
            if (device == null) return;
            if ((device.ConnectionStatus == ConnectionStatus.CS_Connected) &&
                  (device.DeviceStatus == DeviceStatus.DS_InScan))
                device.StopScan();

        }

        private void SDK_Demo_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (device == null) return;
            if (device.ConnectionStatus == ConnectionStatus.CS_Connected)
                device.ReleaseDevice();
        }

        private void buttonRefresh_Click(object sender, EventArgs e)
        {
            Invoke((MethodInvoker)delegate { toolStripStatusLabelInfo.Text = "Info RFID : Search connected
device"; });
            FindDevice();
        }

        private void SDK_Demo_Load(object sender, EventArgs e)
        {
            FindDevice();
        }

        private void comboBoxDevice_SelectedIndexChanged(object sender, EventArgs e)
        {
            selectedDevice = comboBoxDevice.SelectedIndex;
        }
        private void comboBoxFP_SelectedIndexChanged(object sender, EventArgs e)
        {
            selectedFP = comboBoxFP.SelectedIndex;
        }

        private void buttonEnrol_Click(object sender, EventArgs e)
        {

            UserClassTemplate uct = new UserClassTemplate();

            uct.firstName = textBoxFirstName.Text;
            uct.lastName = textBoxLastName.Text;
```

```
            // serial FP to null for all the FP device
            // template to null - it's a new one
            uct.template = device.EnrollUser(null, uct.firstName, uct.lastName, null);

            // To do
            // Save fingerprint template


            userArray[indexUser++] = uct;
            listBoxUser.Items.Add(uct.firstName + " " + uct.lastName);

            textBoxFirstName.Text = null;
            textBoxLastName.Text = null;

            loadTemplate();

        }

        private void listBoxUser_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            int selectedIndex = listBoxUser.SelectedIndex;
            if (selectedIndex == -1) return;

            userArray[selectedIndex].template  =  device.EnrollUser(null,  userArray[selectedIndex].firstName,
userArray[selectedIndex].lastName, userArray[selectedIndex].template);

            // To do
            // Save fingerprint template

            loadTemplate();
        }

        private void loadTemplate()
        {

            string[] templateToLoad = new string[indexUser + 1];

            for (int loop = 0; loop < indexUser; loop++)
            {
                templateToLoad[loop] = userArray[loop].template;
            }
            device.LoadFPTemplate(templateToLoad, device.get_FP_Master);
        }
    }


    public class UserClassTemplate
    {
        public string firstName;
        public string lastName;
        public string template;
    }
}
```