UIDAI

Unique Identification Authority of India
Planning Commission, Govt. of India (GoI),
3rd Floor, Tower-II,
Jeevan Bharati Building,
Connaught Circus,
New Delhi - 110001

A Note on the Usage of Common API Standard for Biometric Devices

February 2015
Version 1.0

Table of Contents

1 Introduction

A standard approach is sought to develop a common API for interfacing with client applications so that the usage of biometric devices does not have dependency on vendor specific device drivers/ libraries. This is specifically relevant as Government of India (GoI) has launched several Aadhaar based schemes such as Aadhaar enabled Public Distribution System (AePDS), MGNREGA, Social Security Pension (SSP), Aadhaar Enabled Biometric Attendance System (AEBAS) and "Jeevan Pramaan" scheme that leverage biometric authentication of Aadhaar holders.

It is foreseen that many benefit schemes would be launched by GoI based on the digitization approach and with the increased number of biometric authentication transactions, resulting in huge demand for biometric devices. Hence, the client interface API provided by different device vendors needs to be standardized and all device vendors / suppliers need to implement a common API for biometric devices (FP, IRIS). This is to ensure uniformity, portability across various form factors that include handheld, tablets, personal computers (laptops and desktops) and support for different target platforms viz. Windows (32 bit, 64 bit), Linux, Mac, Android etc. The common interface API should also be made available in different languages and address the types of clients operating on .NET, Java, C/C++, etc. To make the API release robust, an arrangement would be established with the device vendors by STQC to ensure compliance towards device standards and in addressing the need for common interface API and thorough testing processes would be carried as per the UIDAI compliance for biometric devices.

An initial draft of the common API documentation has been created for using FP and IRIS devices in client applications and highlighted in Section 2. As it is foreseen that there would be a huge demand for biometric devices based on the roll out of more Aadhaar based schemes, the above approach would ensure that all device suppliers / OEMs would be able to implement the common interface to client applications so that biometric devices (FP/IRIS) supplied by any device supplier would work on a plug-and-play mode without requiring much effort from the end user to use the biometric devices with any Aadhaar based client application.

UIDAI recommends the usage of STQC certified biometric sensor / device in all applications based on Aadhaar Authentication. With the introduction of Common API standard for Biometric Devices (FP, IRIS), the current certification procedures and processes defined by STQC would be aligned for certifying Authentication Devices (Fingerprint, IRIS) and inclusion of the 'Common API' standard in the UIDAI Biometric Device Specifications.

In the above context, this document is intended to provide a guideline to adopt "Common API specifications for Biometric Devices" in UIDAI Biometric Device

Specifications for Authentication. Also, it aims to allow STQC to accommodate the Common API specifications in the current STQC certification process for Authentication Devices (Fingerprint, IRIS) and inclusion of the 'Common API' specifications in the UIDAI Biometric Device Specifications.

2 Common API Standard

The primary Methods in the common API across all the UIDAI certified devices should be as following: There can be a handler class, which can be instantiated to perform various biometric device related tasks.

Eg.

```
BiometricDeviceHandler handler = new BiometricDeviceHandler(<current  Object>,
<UniqueBiometricDeviceID>);
```

2.1 Method to Initialize Device

This method should initialize the biometric device (both iris and fingerprint alike). The parameters passed into the method shall be unique device id and optional vendor id. The return value of the method can be an integer value. On successful initialization the return can be numeric 0. In case of any error the returned value can be an integer value corresponding to predefined description.

To API should also be able to give the device id and vendor id when any supported device is connected (plug 'n' play).

Method Signature:

```
<Integer>=handler.GetAttachedDeviceID();
```

```
<Integer>=handler.GetAttachedDeviceVendorID();
```

```
<Integer> =handler.InitDevice(<dataType>  id);
```

The initialization module should also be able to provide the make, model and serial number of the device.

Method Signature:

```
<String>=handler.GetDeviceMake();
```

```
<String>=handler.GetDeviceModel();
```

```
<String>=handler.GetDeviceSerialNumber();
```

2.1.1 Special Case for Android

As ApplicationContext is required in Android therefore before calling the InitDevice() method, the SetApplicationContext() method should be called for passing the application context to the BioDeviceHandler instance and from there passing it to the specific biometric device wrapper class.

Method Signature:

```
handler.SetApplicationContext(<currentApplicationContext>);
```

2.2 Method to get Biometric Device Type (Fingerprint or Iris)

There should be a method to initialize the biometric device type. The method should return either 0 or 1; 0 if the initialized device is a fingerprint device and 1 if it is iris device.

Method Signature:

```
<Integer>=handler.GetDevice Type();
```

In case the attached device is a fingerprint device, additional function for setting the biometric data type required from the captured fingerprint in terms of “FIR” or “FMR” and the passed values type should be string and the actual values can be “FIR” or “FMR”. The biometric data type set through this function should be returned by the hadler method.

Method Signature:

```
handler.SetFingerprintBiometricDataType(<string>);
```

2.3 Method to initialize the threshold value of biometric quality in percentage

There should be a method to initialize the quality parameter of the biometric capture in percentage.

Method Signature:

```
handler.SetThresholdQuality(<Integer>);
```

2.4 Method to initialize the capture timeout in milliseconds

There should be a method to initialize the timeout parameter in millisecond. When the timeout occurs, the current capture request will be aborted

Method Signature:

handler.SetCaptureTimeout(<Integer>);

2.5 Method to Start Live capture of biometric

This method should start the live capture of the biometric feature (both iris and fingerprint alike). The capture should end once the captured quality of the biometric data is equal to or greater than the initialized quality parameter.

Method Signature:

handler.BeginCapture();

2.6 Handler method to retrieve the biometric data in bytes which should be in UIDAI required format

This method should be implemented in the calling class/module, where the task to be implemented after successful biometric capture should be defined. The handler method should be called from the BiometricHandler class, either when there is successful capture of aborted capture.

Method Signature

HandlerFunction(< Raw Captured Image, val1>, <int image_height>,<int image_width><int status,val2>,<String errorMsg>,<boolean complete val3> ,<byte isoData, val4>,<Integer quality><Integer finalNFIQ>);

where:

val1= the raw image data in bytes

image_height= height of the raw image in integer

image_width= width of the raw image in integer

val2 = capture status, 0 if success, -ve if device error, +ve if user error

errorMsg = error message in string if val2 in not 0

val3 = flag to check if capture process is complete

val4 = the biometric data in ISO format in case the capture status (val2) is 0 and complete flag (val3) is true

quality = quality of the captured frame in % value

finalNFIQ = NFIQ value of the captured biometric in case the capture status (val2) is 0 and complete flag (val3) is true

2.7 Method to Uninit the device

Method Signature:

– *handler.UnInitDevice();*

2.8 Illustration of Common API through examples

2.8.1 Android/Java

```
//helperInterface.java

public interface HelperInterface {

    void handlerFunction(final byte rawImage[],final int imageHeight, final int imageWidth, final int status, final String errorMessage,final boolean complete,final byte isoData[],final int quality,final int finalNFIQ);


}

//MainActivity.java

public class MainActivity implements HelperInterface {

int deviceID;

BiometricDeviceHandler handler= new BiometricDeviceHandler(this,deviceID);

public void initDevice(){

        if(connectedDeviceID==deviceID){

        handler.InitDevice(deviceID);

        handler.SetApplicationContext(MainActivity.this); //only in case of Android

        handler.SetFingerprintBiometricDataType(bioDataType);
```

```
            handler.SetCaptureTimeout(captureTimeout);
            handler.SetThresholdQuality(quality);
            deviceMake = handler.GetDeviceMake();
            deviceModel = handler.GetDeviceModel();
            serialNumber = handler.GetDeviceSerialNumber();
}
}
//BiometricDeviceHandler.java
public class BiometricDeviceHandler  {
VendorWrapper wrapperVendor1;
HelperInterface obj;
public BiometricDeviceHandler( HelperInterface obj, int deviceID) {
                        if(deviceID==vendor1DeviceID){
                                obj = this.obj;
                                wrapperVendor1=new VendorWrapper();
                        }
}
//SetApplicationContext  Function Only for Android
public void SetApplicationContext(Context parentContext) {
                                if ( deviceID == vendor1DeviceID) {
                                        wrapperVendor1.SetApplicationContext(parentContext);
                                }
                        }
}
```

2.8.2 C#, .NET

BiometricDeviceHandler class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using DeviceWrapper;
namespace CommonAPI_Combined
{
   public class BiometricDeviceHandler
   {
      XYZDeviceHandler handler = null;
      private static String selecteddev = "";
      int XYZdevid = 10;
      public delegate void MainHandlerFunction(byte[] ImgData, int height, int width, int status, string ErrMsg, bool complete, byte[] ISOFeature,  int quality, int FinNFIQ);
     public event MainHandlerFunction CallHandler;
     public BiometricDeviceHandler(object parent, int devid)
      {
         try
         {
               handler = new XYZDeviceHandler(parent);
               handler.CallHandler += new XYZDeviceHandler.HandlerFunction(handler_CallHandler);
           }
         catch (Exception e)
         {
            Console.Write(e.ToString());
         }
      }
```

```
 void handler_CallHandler(byte[] rawData, int height, int width, int status, string errorMessage, bool complete,
byte[] isoData, int quality, int finalNFIQ)
    {
        CallHandler(rawData, height, width, status, errorMessage, complete, isoData, quality, finalNFIQ);
    }
    public int InitDevice(int id)
    {
        if (id == 10)
        {
            selecteddev = "Device-X";
            return handler.InitDevice(id);
        }
        else
        {
            return -1;
        }
    }
    public int UnInitDevice()
    {
        int ret = -1;
        if (selecteddev.Equals("Device-X "))
        {
            ret = handler.UnInitDevice();
        }
        return ret;

    }
    public int GetDeviceType()
    {
```

```
    int ret = -1;
    if (selecteddev.Equals("Device-X "))
    {
        ret = handler.GetDeviceType();
    }
    return ret;
}
public void SetThresholdQuality(int val)
{
    if (selecteddev.Equals("Device-X "))
    {
        handler.SetThresholdQuality(val);
    }
}
public void SetCaptureTimeout(int val)
{
    if (selecteddev.Equals("Device-X "))
    {
        handler.SetCaptureTimeout(val);
    }
}
public void SetFingerprintBiometricDataType(string val)
{
    if (selecteddev.Equals("Device-X"))
    {
        handler.SetFingerprintBiometricDataType(val);
    }
}
```

```
public int GetAttachedDeviceVendorID()
{
    int ret = -1;
    if (selecteddev.Equals("Device-X"))
    {
        ret = handler.GetAttachedDeviceVendorID();
    }
 }
public int GetAttachedDeviceID()
{
    int ret = -1;
    if (selecteddev.Equals("Device-X"))
    {
        ret = handler.GetAttachedDeviceID();
    }
    return ret;
}
public string GetDeviceMake()
{
    string ret = "";
    if (selecteddev.Equals("Device-X"))
    {
        ret = handler.GetDeviceMake();
    }
    return ret;
}
```

```
public string GetDeviceModel()
{
    string ret = "";
    if (selecteddev.Equals("Device-X"))
    {
        ret = handler.GetDeviceModel();
    }
    return ret;

}

public string GetDeviceSerialNumber()
{
    string ret = "";
    if (selecteddev.Equals("Device-X"))
    {
        ret = handler.GetDeviceSerialNumber();
    }
    return ret;
}

public void BeginCapture()
{
    if (selecteddev.Equals("Device-X"))
    {
        handler.BeginCapture();
    }
}
```

```
        public Bitmap GetFingerBitmap(byte[] image, int width, int height)
        {
            Bitmap ret = null;
            if (selecteddev.Equals("Startek FM220"))
            {
                ret =  startekhandler.GetFingerBitmap(image, width, height);
            }
            return ret;
        }
    }
}
```

MainForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Mantra;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.Threading;
using System.Net;
using System.IO;
using System.Web;
using System.Xml;

namespace CommonAPI_Combined
{
    public partial class CommonAPIForm : Form
    {
        BiometricDeviceHandler handler = null;
        ColorPalette pal;
        public CommonAPIForm()
        {
```

```
        InitializeComponent();
        CommonAPIForm.CheckForIllegalCrossThreadCalls = false;
    }

    private void Form1_Load(object sender, EventArgs e)
    {
```

//Creates biometric device handler class object

```
        handler = new BiometricDeviceHandler(this, connectedDevId);
        handler.CallHandler += new
BiometricDeviceHandler.MainHandlerFunction(handler_MainHandlerFunction);

    }

    void handler_MainHandlerFunction(byte[] rawData, int height, int width, int status, string errorMessage, bool
complete, byte[] isoData, int quality, int finalNFIQ)
    {
        if (status == 0)
        {
            lblQuality.Text = "Quality: " + quality.ToString();
            lblQuality.Refresh();
            Thread t = new Thread(new ThreadStart(() =>
            {
                ShowImage(rawData, height, width);
            }));
            t.IsBackground = true;
            t.Start();
        }
        if (complete)
        {
```

```
        if (status == 0)
        {
       //On capture success
            String min = Convert.ToBase64String(isoData);
        }
        else
        {
            MessageBox.Show("Status: " + status.ToString() + "\nError Message: " + errorMessage, true);
        }
    }
}
private void ShowImage(byte[] rawData, int height, int width)
{
    try
    {
        Bitmap b = new Bitmap(width, height, PixelFormat.Format8bppIndexed);
        pal = b.Palette;
        b.Dispose();
        for (int i = 0; i < pal.Entries.Length; i++)
        {
            pal.Entries[i] = Color.FromArgb(255, i, i, i);
        }
        b = new Bitmap(width, height, PixelFormat.Format8bppIndexed);
        BitmapData data;
       data = b.LockBits(new Rectangle(0, 0, width, height), ImageLockMode.ReadWrite,
       PixelFormat.Format8bppIndexed);
        Marshal.Copy(rawData, 0, data.Scan0, width * height);
        b.UnlockBits(data);
```

```
            b.Palette = pal;
            b.SetResolution(500, 500);
            picFinger.Image = b;
        }
        catch (Exception ex)
        {
        }
    }
    private void btnDeviceID_Click(object sender, EventArgs e)
    {
//Returns Device ID
        String devid = handler.GetAttachedDeviceID().ToString();
    }
    private void btnVendorID_Click(object sender, EventArgs e)
    {
//Returns Vendor ID
        String vendorid = handler.GetAttachedDeviceVendorID().ToString();
    }

    private void btnInit_Click(object sender, EventArgs e)
    {
//Initializes the device
         int res = handler.InitDevice(connectedDevId);
    }
    private void btnSerialNo_Click(object sender, EventArgs e)
    {
//Returns Device Serial
        String serial = handler.GetDeviceSerialNumber();
```

```
    }

    private void btnMake_Click(object sender, EventArgs e)
    {
//Returns Device Make
        String deviceMake = handler.GetDeviceMake();
    }
    private void btnModel_Click(object sender, EventArgs e)
    {
//Returns device model
        String devModel =  handler.GetDeviceModel();
    }
    private void btnQuality_Click(object sender, EventArgs e)
    {
//Sets fbiometric threshold quality
            handler.SetThresholdQuality(Convert.ToInt32(qualityValue));
    }
    private void btnTimeout_Click(object sender, EventArgs e)
    {
//Set capture timeout in milliseconds
            handler.SetCaptureTimeout(Convert.ToInt32(timeoutValue));
    }
    private void btnBioType_Click(object sender, EventArgs e)
    {
//set biometric type FMR for fingerprint, IIR for Iris
            handler.SetFingerprintBiometricDataType(bioType);
    }
    private void btnDeviceType_Click(object sender, EventArgs e)
```

```
        {
//Returns Device Type
            String devtype =  handler.GetDeviceType().ToString();
        }
        private void btnCapture_Click(object sender, EventArgs e)
        {
//Begins Biometric Capture
            handler.BeginCapture();
        }
        private void btnUnInit_Click(object sender, EventArgs e)
        {
//Unitializes the device
            int res = handler.UnInitDevice();
        }
    }
}
```