UIDAI

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

Biometric Iris Sensor Integration API

Draft

Version 0.99

Revised Date: 10-Oct-2013

Contents

1 Introduction

The Unique Identification Authority of India (UIDAI) has been created, with the mandate of providing a unique identity to all Indian residents. The UIDAI proposes to use biometrics also to conduct authentication transactions in future. Proof of concept studies is being conducted in various phases in order to ascertain state of technology as well as viability of the concept.

Iris sensors are vital components required for carrying out iris authentication. As discussed in various concept papers, UIDAI proposes to offer biometric authentication through combination of UID number + biometric. UIDAI supports both Fingerprint and Iris based biometric authentication.

2 Objective of the document

The current version of this document has been provided for feedback from the Application developers, as well as Biometric Iris Capture Device manufacturers. This API is intended for Iris Biometric device manufacturers to publish API implementation. This API will facilitate the smooth integration of the API into the UIDAI POC application.

3 Application Overview

The API is specified as communication protocol between the UID POC application Software and the Iris Biometric Capture Devices. The proposed interface does not address any security-related issues. After the security requirements are defined, the specifications may need to be modified to address them.

UID biometric POC Application software is being developed on Java Platform. POC application is designed to accept UID number of the resident, complete the iris capture process and then transmit the data in encrypted packet to the Authentication servers as per the UIDAI authentication specification. Following functions are expected to be performed by the UIDAI POC application.

1. *Accept the UID number from the resident (No API calls required here)*
2. *Choose one among many sensors connected to the PC through USB. (init() will be called)*
3. *Proceed to collect the biometric solution from the resident.*
    a. *Iris Raw Image as per ISO format (ISO 19794-6) (startPreview() for continuous feedback and startCapture() for the actual capture of Iris)*
    b. *Raw to BMP of Iris Image in order to display the image in the application UI.*
4. *In cases where more than one iris is collected, application is expected to compare the captured iris with already captured iris and verify for duplication. (compareSamples() will be called if implemented)*
5. *Form the authentication request as per the Authentication API and send it to server and store the images in the log files of the device for future analysis.*
6. *All buffered transactions will be stored in table in mysql db in the system*

4 Installer Design

*All device vendors are expected to provide the installers that install the device drivers, libraries and other required components on the system. Initial expectation is that the installer be available on windows XP and above platform (**both 32 and 64bit**).*

Vendors can choose either one of the two approached, JNI [Java Native Interface] or Java Interface implementation of the API. Please refer the sections below for both JNI and Java Interface API. Please send us your valuable feedback and suggestions to make this API easily implementable by any vendors.

4.1 JNI - Approach 1

In addition installing the libraries, the interface API as per specification is expected to be installed in the following directory as per the naming convention provided below.

Please expect to find the **IrisSensors** *folder in the client application directory of every system. Within the **IrisSensors** directory, installer has to create a **DIRECTORY** with naming convention listed below to identify the unique devices connected to the system.*

DeviceManufacturer-DeviceModelIdentifier-DeviceVendor

For Example:
*Device manufacturer: **XYZ***
*Device Model: **Iris3000***
*Device Vendor: **ABCD***

Here, the Device Manufacturer tag is "XYZ", Device Model Identifier is "Iris 3000 series" and the Device Vendor is ABCD, hence the folder name in the destination name **could** *be* ***XYZ-Iris3000-ABCD***

/IrisSensors/XYZ-Iris3000-ABCD/

Within this folder, the installer is expected to have the following files.

1. *A DLL file which implements the API as per the specification cited above. The naming convention for single point of access (Entry Point) -* ***DeviceManufacturerDeviceModelIdentifier-DeviceVendor*****.DLL**
    a. *If more than one dll is required for any sensor, for example as in case of a sensor with a different extractor matcher, then needs to add all other dll files in the same folder. All API calls should be wrapped in one dll file* ***DeviceManufacturer-DeviceModelIdentifier-DeviceVendor*****.DLL**

    Note:
    There should not be any dependency on this dll from other dll files.
    The name of the dll for eg.: XYZ-Iris3000-ABCD.dll

7. *Configuration or property files (if any) which customizes the output of the DLL. Text files are preferred. This will help UIDAI teams also to customize the output of the sensor.*

8. *Readme Document - that explains the property files and any other important information related to the API and configuration and the version of the software.*

4.2 Java Interface - Approach 2

The implemented class for the API should be bundled in a jar file along with the supporting classes with the documentation and the version of the software. Please provide the jars with the below naming convention.

DeviceManufacturer-DeviceModelIdentifier-DeviceVendor

For Example:
Device manufacturer: ***XYZ***
Device Model: ***Iris3000***
Device Vendor: ***ABCD***

Here, the Device Manufacturer tag is "XYZ", Device Model Identifier is "Iris 3000 series" and the Device Vendor is ABCD, hence the folder name in the destination name **could** *be* ***XYZ-Iris3000-ABCD***

5 API Specifications

The development platform: JAVA 1.6, Windows XP or above

The below Interface API should be implemented by the device vendors for their supporting devices.

The Interfaces to be implemented by the device vendors:

Note:
New API calls and constants are highlighted.

AuthenticationIrisAPI.java [JNI API]

```
/*
 *   $$AuthenticationIrisAPI.java$$
 *
 * Copyright (c) 2011 UID Authority of India. All rights reserved.
 * Printed in India.
 *
 * No part of this artifact may be reproduced or transmitted in any form or by
 * any means, electronic or mechanical, whether now known or later invented,
 * for any purpose without the prior and express written consent of UID
 * Authority of India.
 *
 *   $License$
 *
 *    ================================================================================
 */

package in.gov.uidai.auth.biometric;


/**
 * This class contains Native API's for performing the following tasks: 1.
 * initialize the device 2. check the device connection 3. starts the preview of
 * the frames 4. capturing iris samples from the device (Raw and ISO) 5. samples
 * verification with the captured samples if implemented (Not mandatory for
```

```
 * Iris) 6. creation of BMP image from the captured sample to display on the UI
 * 7. device information API 8. close the connection and uninitialize the device
 * and repeat from step 1 for next device
 *
 * @author UIDAI, Bangalore.
 * @version 0.99
 */
public class AuthenticationIrisAPI implements IAuthenticationIrisAPI {

        /**
         * The init() API initializes the device connected to the system Returns 0
         * for successful initialization else -1 Please refer interface
         * IAuthenticationIrisAPI for return codes
         */
        public native int init();

        /**
         * The isDeviceConnected() API checks the connection is active or not with
         * the system Returns true for success else false Please refer interface
         * IAuthenticationIrisAPI for return codes
         */
        public native boolean isDeviceConnected();

        /**
         * The startPreview() API call gets the frames from the sensor device to the
         * callback class to view on the UI
         *
         * @param biometricSubtype
         *  @param  authenticationCallback
         */
        public native void startPreview(int biometricSubtype,
                        IAuthenticationCallback authenticationCallback);

        /**
         * The cancelPreview() API call interrupts the preview of the frames
         */
        public native void cancelPreview();

        /**
         * The startCapture() API call actually initiates the auto capture of the
         * sample
         *
         * @param purpose
         * @param imageKind
         */
        public native void startCapture(int purpose, int imageKind);

        /**
         * The cancelCapure() API cancels the capture of the sample
         *
         */
        public native void cancelCapture();

        /**
         * The compareSamples() API compares the samples captured. Returns the
         * MatchResult object which contains match score and true or false for match
         * found Error code is set in MatchResult object accordingly
         *
         * @param iris1
         *            is the previous captured sample
         * @param iris2
         *            is the current captured sample
         */
        public native MatchResult compareSamples(IrisSample iris1, IrisSample iris2);

        /**
         * The getDeviceInfo() API returns the device information like serial
         * number, make, model etc Returns DeviceInfo object which contains all the
         * details about the device Error code is set in DeviceInfo object
         * accordingly
         */
        public native DeviceInfo getDeviceInfo();

        /**
         * The getExtractorInfo() API returns the extractor information like vendor, name, version etc
         * Returns ExtractorInfo object which contains all the details about the extractor
         * Error code is set in ExtractorInfo object accordingly
        */
        public native ExtractorInfo getExtractorInfo();

        /**
         * The close() API is to uninitialize the device from the system. Device
         * will become deactive after this call. NOTE: For multi vendor and multi
         * device connection, please call this API before initializing the next
         * device.
         */
        public native int close();

}
```

IAuthenticationIrisAPI.java [Java API]

```
/*
 *   $$IAuthenticationIrisAPI.java$$
 *
 * Copyright (c) 2011 UID Authority of India. All rights reserved.
 * Printed in India.
 *
 * No part of this artifact may be reproduced or transmitted in any form or by
 * any means, electronic or mechanical, whether now known or later invented,
 * for any purpose without the prior and express written consent of UID
 * Authority of India.
 *
 *  $License$
 *
 *   ================================================================================
 */

package in.gov.uidai.auth.biometric;

/**
 * This interface contains all error codes for the API
 *
 * APIs for performing the following tasks:
 * 1. initialize the device
 * 2. check the device connection
 * 3. starts the preview and gets the frames from the device
 * 4. capturing iris samples from the device (Raw and ISO)
 * 5. samples verification with the captured samples if implemented (Not mandatory for Iris)
 * 6. creation of BMP image from the captured sample to display on the UI
 * 7. device information API
 * 8. close the connection and uninitialize the device and repeat from step 1 for next device
 *
 * @author UIDAI, Bangalore.
 * @version 0.99
 */
public interface IAuthenticationIrisAPI {

        public final int IRIS_API_DEVICE_INIT_SUCCESSFUL = 0;
        public final int IRIS_API_DEVICE_INIT_UNSUCCESSFUL = -1;
        public final int IRIS_API_DEVICE_ERR_UNABLE_TO_CAPTURE = -2;
        public final int IRIS_API_DEVICE_ERR_TIMEOUT_EXPIRED = -4;
        public final int IRIS_API_DEVICE_ERR_PURPOSE_NOT_SUPPORTED = -5;
        public final int IRIS_API_SAMPLES_MATCH = 6;
        public final int IRIS_API_SAMPLES_NOTMATCH = -6;
        public final int IRIS_API_SAMPLE_RECTILINEAR_FORMAT = 7;
        public final int IRIS_API_SAMPLE_POLAR_FORMAT = 8;
        public final int IRIS_API_DEVICE_CAPTURE_SUCCESSFUL = 10;
        public final int IRIS_API_DEVICE_CAPTURE_UNSUCCESSFUL = -10;
        public final int IRIS_API_DEVICE_PURPOSE_VERIFICATION = 12;
        public final int IRIS_API_SAMPLE_ISO_FORMAT = 16;
        public final int IRIS_API_DEVICE_SAMPLE_QUALITY_GOOD = 18;
        public final int IRIS_API_DEVICE_SAMPLE_QUALITY_POOR = -18;
        public final int IRIS_API_DEVICE_INIT_DLL_ALREADY_LOADED = -21;
        public final boolean IRIS_API_DEVICE_CONNECTED = true;
        public final boolean IRIS_API_DEVICE_DISCONNECTED = false;
        public final int IRIS_API_DEVICE_CLOSE_SUCCESSFUL = 22;
        public final int IRIS_API_DEVICE_CLOSE_UNSUCCESSFUL = -22;
        public final int IRIS_API_EYE_UNDEF = 0;
        public final int IRIS_API_EYE_RIGHT = 1;
        public final int IRIS_API_EYE_LEFT = 2;
        public final int IRIS_API_EYE_BOTH = IRIS_API_EYE_RIGHT | IRIS_API_EYE_LEFT;

        public final int IRIS_API_START_IRIS_PREVIEW_SUCCESS = 23;
        public final int IRIS_API_START_IRIS_PREVIEW_FAIL = -23;
        public final int IRIS_API_IMAGE_TYPE_UNKNOWN = 0;
        public final int IRIS_API_IMAGE_TYPE_UNCROPPED = 1;
        public final int IRIS_API_IMAGE_TYPE_VGA = 2;
        public final int IRIS_API_IMAGE_TYPE_CROPPED = 3;

        public final int IRIS_API_IMAGE_TYPE_CROPPED_AND_MASKED_1_5 = 7;

        /** For certification of device the POC client application will consider only kind 8 and will pass this argument in
       statCapture method */

        public final int IRIS_API_IMAGE_TYPE_CROPPED_AND_MASKED_2_5 = 8;


        public final int IRIS_API_IMAGE_TYPE_CROPPED_AND_MASKED_3_5 = 9;
        public final int IRIS_API_IMAGE_TYPE_CROPPED_AND_MASKED_5_0 = 10;


        /**
         * The init() API initializes the device connected to the system
         * Returns 0 for successful initialization else -1
         * Please refer interface IAuthenticationIrisAPI for return codes
         */
        public int init();

        /**
         * The isDeviceConnected() API checks the connection is active or not with the system
         * Returns true for success else false
         * Please refer interface IAuthenticationIrisAPI for return codes
         */
        public boolean isDeviceConnected();
```

```
        /**
         * The startPreview() API call gets the frames from the sensor device
         * to the callback class to view on the UI
         * @param biometricSubtype
         *  @param  authenticationCallback
         */
        public void startPreview(int biometricSubtype, IAuthenticationCallback authenticationCallback);

        /**
         * The cancelPreview() API call interrupts the preview of the frames
         */
        public void cancelPreview();

        /**
         * The startCapture() API call actually initiates the auto capture of the sample
         * @param purpose
         * @param imageKind
         */
        public void startCapture(int purpose, int imageKind);

        /**
         * The cancelCapure() API cancels the capture of the sample
         */
        public void cancelCapture();

        /**
         * The compareSamples() API compares the samples captured.
         * Returns the MatchResult object which contains match score and true or false for match found
         * Error code is set in MatchResult object accordingly
         * @param iris1 is the previous captured sample
         * @param iris2 is the current captured sample
         */
        public MatchResult compareSamples(IrisSample sampel1, IrisSample sample2);

        /**
         * The getDeviceInfo() API returns the device information like serial number, make, model etc
         * Returns DeviceInfo object which contains all the details about the device
         * Error code is set in DeviceInfo object accordingly
         */
        public DeviceInfo getDeviceInfo();

        /**
         * The getExtractorInfo() API returns the extractor information like vendor, name, version etc
         * Returns ExtractorInfo object which contains all the details about the extractor
         * Error code is set in ExtractorInfo object accordingly
        */
        public ExtractorInfo getExtractorInfo();

        /**
         * The close() API is to uninitialize the device from the system.
         * Device will become deactive after this call.
         * NOTE: For multi vendor and multi device connection,
         * please call this API before initializing the next device.
         */
        public int close();

}
```

IAuthenticationCallback.java

```
/*
 *    $$IAuthenticationCallback.java$$
 *
 * Copyright (c) 2011 UID Authority of India. All rights reserved.
 * Printed in India.
 *
 * No part of this artifact may be reproduced or transmitted in any form or by
 * any means, electronic or mechanical, whether now known or later invented,
 * for any purpose without the prior and express written consent of UID
 * Authority of India.
 *
 *   $License$
 *
 *    ================================================================================
 */
package in.gov.uidai.auth.biometric;

/**
 * This interface contains all callback methods for the actions performed on the
 * device
 *
 * @author UIDAI, Bangalore.
 * @version 0.99
 */
public  interface  IAuthenticationCallback  {
        /**
         * To be implemented if there are any preview errors in the device
         *
         * @param errorCode
         *  @param  errorString
         */
        void onPreviewError(int errorCode, String errorString);

        /**
```

```
         * To be implemented to start the iris preview UID Authority of India
         */
        void onPreviewStarted();

        /**
         * To be implemented to cancel the preview before capturing the sample
         */
        void onPreviewCancelled();

        /**
         * To be implemented to stream the Iris image frames on GUI.
         *
         * @param width
         * @param height
         * @param imageData
         * @param biometricSubtype
         *            : left or right eye : dual-eye cameras can send the preview
         *            images for the left and right eye in any order, one eye at a
         *            time.
         *  @throws  Exception
         */
        void onIrisImageAvailable(int width, int height, byte[] imageData, int biometricSubtype);

        /**
         * To be implemented to stream the Iris image frames on GUI. For
         * compatibility with single-eye cameras
         *
         * @param width
         * @param height
         *   @param   imageData
         *  @throws  Exception
         */
        void onIrisImageAvailable(int width, int height, byte[] imageData);

        /**
         * To be implemented to start the auto capture of the iris sample.
         */
        void onCaptureStarted();

        /**
         * To be implemented for capture errors in the device
         *
         * @param errorCode
         *  @param  errorString
         */
        void onCaptureError(int errorCode, String errorString);

        /**
         * To be implemented to cancel the capture or interrupt the capture before
         * the auto capture of the samples
         */
        void onCaptureCancelled();

        /**
         * To be implemented to load the final sample object. It is called by the
         * one-eye cameras
         *
         *  @param  sample
         */
        void onCaptureCompleted(IrisSample sample);

        /**
         * To be implemented to load the final sample objects. It is called by the
         * two-eye cameras, one for each eye.
         *
         * @param samples
         *            : array of two sample objects samples[0] - right eye,
         *            samples[1] - left eye if only one eye is captured, set the
         *            other sample object to null
         */
        void onCaptureCompleted(IrisSample[] samples);

        /**
         * To provide actionable feedback to the operator
         *
         * @param feedback
         *            - can be HTML or RTF for formatting
         */
        void onFeedbackAvailable(String feedback);
}
```

IrisSample.java

```
/*
```

```
 *   $$IrisSample.java$$
 *
 * Copyright (c) 2011 UID Authority of India. All rights reserved.
 * Printed in India.
 *
 * No part of this artifact may be reproduced or transmitted in any form or by
 * any means, electronic or mechanical, whether now known or later invented,
 * for any purpose without the prior and express written consent of UID
 * Authority of India.
 *
 *   $License$
 *
 *    ============================================================================
 */

package in.gov.uidai.auth.biometric;

/**
 * This class contains images captured from the API call
 * startCapture() and onCaptureCompleted() callback method
 * We may need to add some more attributes of the sample being captured in this class
 * Note: Currently this API supports Single Iris device only,
 * For dual Iris, the captureSample API will be called in sequence
 *
 * @author UIDAI, Bangalore.
 * @version  0.99
 */
public class IrisSample {

        /**
         * Set the raw image of the Iris
         */
        byte[] rawImage = null;

        /**
         * Set the Size of the raw image of Iris
         */
        int rawImageSize = 0;

        /**
         * Set the width of the raw image of Iris
         */
        int rawImageWidth = 0;

        /**
         * Set the height of the raw image of the Iris
         */
        int rawImageHeight = 0;

        /**
         * This is the Time taken in "milliseconds" to
         * capture a Iris by the device, i.e. the time in milliseconds
         * between "Iris capture start and end"
         */
        long captureTime = -1L;

        /**
         * Set an array of ISO sample of the Iris for the different image kinds
         * JPEG2000: ISO/IEC 19794-6:2005(E)
         * Kind 1, 2, 3 and 7: ISO/IEC 19794-6:2011(E)
         */
        ISOIrisSample[] isoSamples = null;

        /**
         * Null if no error, else set the reason for error.
         */
        String errorString = null;

        /**
         * Set the error code from the API.
         */
        int errorCode = 0;

        /**
         * @return the rawImage
         */
        public byte[] getRawImage() {
                return rawImage;
        }

        /**
         * @param rawImage
         *            the rawImage to set
         */
        public void setRawImage(byte[] rawImage) {
                this.rawImage = rawImage;
        }
```

```
/**
 * @return the rawImageSize
 */
public int getRawImageSize() {
        return rawImageSize;
}

/**
 * @param rawImageSize
 *            the rawImageSize to set
 */
public void setRawImageSize(int rawImageSize) {
        this.rawImageSize = rawImageSize;
}

/**
 * @return the rawImageWidth
 */
public int getRawImageWidth() {
        return rawImageWidth;
}

/**
 * @param rawImageWidth
 *            the rawImageWidth to set
 */
public void setRawImageWidth(int rawImageWidth) {
        this.rawImageWidth = rawImageWidth;
}

/**
 * @return the rawImageHeight
 */
public int getRawImageHeight() {
        return rawImageHeight;
}

/**
 * @param rawImageHeight
 *            the rawImageHeight to set
 */
public void setRawImageHeight(int rawImageHeight) {
        this.rawImageHeight = rawImageHeight;
}

/**
 * @return the errorString
 */
public String getErrorString() {
        return errorString;
}

/**
 * @param errorString
 *            the errorString to set
 */
public void setErrorString(String errorString) {
        this.errorString = errorString;
}

/**
 * @return the errorCode
 */
public int getErrorCode() {
        return errorCode;
}

/**
 * @param errorCode
 *            the errorCode to set
 */
public void setErrorCode(int errorCode) {
        this.errorCode = errorCode;
}

/**
 * @return the fpCaptureTime
 */
public long getCaptureTime() {
        return captureTime;
}

/**
 * @param fpCaptureTime the fpCaptureTime to set
 */
public void setCaptureTime(long captureTime) {
```

```
                this.captureTime = captureTime;
        }

        /**
         * @return the isoSamples
         */
        public ISOIrisSample[] getIsoSamples() {
                return isoSamples;
        }

        /**
         * @param isoSamples the isoSamples to set
         */
        public void setIsoSamples(ISOIrisSample[] isoSamples) {
                this.isoSamples = isoSamples;
        }
}
```

ISOIrisSample.java [New Class for Capturing Kinds]

```
/*
 *   $$ISOIrisSample.java$$
 *
 * Copyright (c) 2011 UID Authority of India. All rights reserved.
 * Printed in India.
 *
 * No part of this artifact may be reproduced or transmitted in any form or by
 * any means, electronic or mechanical, whether now known or later invented,
 * for any purpose without the prior and express written consent of UID
 * Authority of India.
 *
 *   $License$
 *
 *    ================================================================================
 */

package in.gov.uidai.auth.biometric;

/**
 * This class contains image captured from the API call startCapture() and
 * onCaptureCompleted() callback method
 * We may need to add some more attributes of the sample being captured in this class
 * Note: Currently this API supports Single Iris device only,
 * For dual Iris, the captureSample API will be called in sequence
 *
 * @author UIDAI, Bangalore.
 * @version 0.99
 */
public class ISOIrisSample {
        /**
         * Set the ISO image of the Iris for the image kind
         * JPEG2000: ISO/IEC 19794-6:2005(E)
         * Kind 1, 2, 3 and 7: ISO/IEC 19794-6:2011(E)
         */
        byte[] iso_19794_6_Image = null; // IIR

        /**
         * Set the size of the ISO image of the Iris
         */
        int iso_19794_6_ImageSize = 0; //ISO Header + Image size

        /**
         * Set the kind of the image
         * 0: Unknown
         * 1: Uncropped
         * 2: VGA
         * 3: Cropped
         * 7: Cropped & Masked
         */
        int imageKind = 0;

        /**
         * Set the compression ratio used, deafult 0 means no compression
         */
        int compressionRatio = 0;

        /**
         * @return the iso_19794_6_Image
         */
        public byte[] getIso_19794_6_Image() {
                return iso_19794_6_Image;
        }

        /**
         * @param iso_19794_6Image the iso_19794_6_Image to set
```

```
         */
        public void setIso_19794_6_Image(byte[] iso_19794_6Image) {
                iso_19794_6_Image = iso_19794_6Image;
        }

        /**
         * @return the iso_19794_6_ImageSize
         */
        public int getIso_19794_6_ImageSize() {
                return iso_19794_6_ImageSize;
        }

        /**
         * @param iso_19794_6ImageSize the iso_19794_6_ImageSize to set
         */
        public void setIso_19794_6_ImageSize(int iso_19794_6ImageSize) {
                iso_19794_6_ImageSize = iso_19794_6ImageSize;
        }

        /**
         * @return the imageKind
         */
        public int getImageKind() {
                return imageKind;
        }

        /**
         * @param imageKind the imageKind to set
         */
        public void setImageKind(int imageKind) {
                this.imageKind = imageKind;
        }

        /**
         * @return the compressionRatio
         */
        public int getCompressionRatio() {
                return compressionRatio;
        }

        /**
         * @param compressionRatio the compressionRatio to set
         */
        public void setCompressionRatio(int compressionRatio) {
                this.compressionRatio = compressionRatio;
        }
}
```

DeviceInfo.java

```
/*
 *   $$DeviceInfo.java$$
 *
 * Copyright (c) 2011 UID Authority of India. All rights reserved.
 * Printed in India.
 *
 * No part of this artifact may be reproduced or transmitted in any form or by
 * any means, electronic or mechanical, whether now known or later invented,
 * for any purpose without the prior and express written consent of UID
 * Authority of India.
 *
 *   $License$
 *
 *    ================================================================================
 */

package in.gov.uidai.auth.biometric;

/**
 * This class contains device information from the getDeviceInfo() API call
 *
 * @author UIDAI, Bangalore.
 * @version 0.99
 */
public class DeviceInfo {

        /**
         * Set the make of the device
         */
        String make = null;

        /**
         * Set the model of the device
         */
        String model = null;
```

```
/**
 * Set the type of the iris device
 * 0 - for Single Iris
 * 1 - for Dual Iris
 *  2  -  for  unknown
 */
int deviceType = 0;

/**
 * Set the firmware version of the device
 */
String firmwareVersion = null;

/**
 * Set this field with info related to any
 * certifications or any other important information
 * related to device only
 */
String info = null;

/**
 * Set the serial number of the device
 */
String serialNumber = null;


/**
 * Null if no error, else set the reason for error.
 */
String errorString = null;

/**
 * Set the error code from the API.
 */
int errorCode = 0;

/**
 * @return the make
 */
public String getMake() {
        return make;
}

/**
 * @param make
 *            the make to set
 */
public void setMake(String make) {
        this.make = make;
}

/**
 * @return the model
 */
public String getModel() {
        return model;
}

/**
 * @param model
 *            the model to set
 */
public void setModel(String model) {
        this.model = model;
}

/**
 * @return the serialNumber
 */
public String getSerialNumber() {
        return serialNumber;
}

/**
 * @param serialNumber
 *            the serialNumber to set
 */
public void setSerialNumber(String serialNumber) {
        this.serialNumber = serialNumber;
}

/**
 * @return the errorString
 */
public String getErrorString() {
        return errorString;
```

```
	}

	/**
	 * @param errorString
	 *            the errorString to set
	 */
	public void setErrorString(String errorString) {
		this.errorString = errorString;
	}

	/**
	 * @return the errorCode
	 */
	public int getErrorCode() {
		return errorCode;
	}

	/**
	 * @param errorCode
	 *            the errorCode to set
	 */
	public void setErrorCode(int errorCode) {
		this.errorCode = errorCode;
	}

	/**
	 * @return the deviceType
	 */
	public int getDeviceType() {
		return deviceType;
	}

	/**
	 * @param deviceType the deviceType to set
	 */
	public void setDeviceType(int deviceType) {
		this.deviceType = deviceType;
	}

	/**
	 * @return the firmwareVersion
	 */
	public String getFirmwareVersion() {
		return firmwareVersion;
	}

	/**
	 * @param firmwareVersion the firmwareVersion to set
	 */
	public void setFirmwareVersion(String firmwareVersion) {
		this.firmwareVersion = firmwareVersion;
	}

	/**
	 * @return the info
	 */
	public String getInfo() {
		return info;
	}

	/**
	 * @param info the info to set
	 */
	public void setInfo(String info) {
		this.info = info;
	}
}
```

MatchResult.java

```
/*
 *   $$MatchResult.java$$
 *
 * Copyright (c) 2011 UID Authority of India. All rights reserved.
 * Printed in India.
 *
 * No part of this artifact may be reproduced or transmitted in any form or by
 * any means, electronic or mechanical, whether now known or later invented,
 * for any purpose without the prior and express written consent of UID
 * Authority of India.
 *
 *   $License$
 *
 *    ============================================================================
 */
```

```java
package in.gov.uidai.auth.biometric;

/**
 * This class contains match results from the API call compareSamples()
 *
 * @author UIDAI, Bangalore.
 * @version 0.99
 */
public class MatchResult {

	/**
	 * Set the boolean for match found
	 */
	boolean matchFound = false;

	/**
	 * Set the matchscore
	 */
	int matchScore = 0;


	/**
	 * Null if no error, else set the reason for error.
	 */
	String errorString = null;

	/**
	 * Set the error code from the API.
	 */
	int errorCode = 0;

	/**
	 * @return the mathFound
	 */
	public boolean isMatchFound() {
		return matchFound;
	}

	/**
	 * @param mathFound
	 *            the mathFound to set
	 */
	public void setMatchFound(boolean matchFound) {
		this.matchFound = matchFound;
	}

	/**
	 * @return the matchScore
	 */
	public int getMatchScore() {
		return matchScore;
	}

	/**
	 * @param matchScore
	 *            the matchScore to set
	 */
	public void setMatchScore(int matchScore) {
		this.matchScore = matchScore;
	}

	/**
	 * @return the errorString
	 */
	public String getErrorString() {
		return errorString;
	}

	/**
	 * @param errorString
	 *            the errorString to set
	 */
	public void setErrorString(String errorString) {
		this.errorString = errorString;
	}

	/**
	 * @return the errorCode
	 */
	public int getErrorCode() {
		return errorCode;
	}

	/**
	 * @param errorCode
	 *            the errorCode to set
	 */
```

```
        public void setErrorCode(int errorCode) {
                this.errorCode = errorCode;
        }

}
```

ExtractorInfo.java

```
/*
 * $$ExtractorInfo.java$$
 *
 * Copyright (c) 2011 UID Authority of India. All rights reserved.
 * Printed in India.
 *
 * No part of this artifact may be reproduced or transmitted in any form or by
 * any means, electronic or mechanical, whether now known or later invented,
 * for any purpose without the prior and express written consent of UID
 * Authority of India.
 *
 * $License$
 *
 * ================================================================================
 */
package in.gov.uidai.auth.biometric;

/**
 * This class contains extractor information from the getExtractorInfo() API call
 * @author UIDAI, Bangalore.
 * @version 0.99
 */

public class ExtractorInfo {

     String vendor = null;
     String name = null;
     String version = null;

     String errorString = null;
     int errorCode = 0;

     /**
      * @return the vendor
      */
     public String getVendor() {
      return vendor;
     }

     /**
      * @param vendor
      *  the vendor to set
      */
     public void setVendor(String vendor) {
      this.vendor = vendor;
     }

     /**
      * @return the name
      */
     public String getName() {
      return name;
     }

     /**
      * @param name
      *  the name to set
      */
     public void setName(String name) {
      this.name = name;
     }

     /**
      * @return the version
      */
     public String getVersion() {
      return version;
     }

     /**
      * @param version
      *  the version to set
      */
     public void setVersion(String version) {
      this.version = version;
     }

     /**
      * @return the errorString
      */
     public String getErrorString() {
      return errorString;
     }
```

```
    /**
     * @param errorString
     *            the errorString to set
     */
    public void setErrorString(String errorString) {
     this.errorString = errorString;
    }

    /**
     * @return the errorCode
     */
    public int getErrorCode() {
     return errorCode;
    }

    /**
     * @param errorCode
     *            the errorCode to set
     */
    public void setErrorCode(int errorCode) {
     this.errorCode = errorCode;
    }

}
```

End of API document.