UIDAI

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

UIDAI BIOMETRIC CAPTURE API

Draft

Table of Contents

1. INTRODUCTION 3

1.1 OBJECTIVE OF THIS DOCUMENT 3
1.2 INTERFACE OVERVIEW 3

2. API METHODS 5

2.1 BIOMETRIC DEVICE MANAGEMENT AND DISCOVERY 5
2.2 VDM COMMANDS 5
2.2.1 Shutdown 5
2.2.2 Reset Device 5
2.3 VDM EVENTS 6
2.3.1 Device Arrival 6
2.3.2 Device Removal 8
2.4 BIOMETRIC DEVICE MANAGEMENT API METHODS 8
2.5 BIOMETRIC DEVICE COMMAND API METHODS 8
2.5.1 Subscribe 8
2.5.2 Start Capture 9
2.5.3 Force Capture 9
2.5.4 Stop Capture 9
2.6 BIOMETRIC DEVICE NOTIFICATION API METHODS 10
2.6.1 Capture Complete 10
2.6.2 Detection 10
2.6.3 User Feedback 11
2.7 BIOMETRIC DEVICE VIDEO STREAMING AND SAMPLE API METHODS 11
2.7.1 Get Frame 12
2.7.2 Get Sample 12
2.8 RETURN CODES 12

3. WORKFLOWS 14

3.1 START SEQUENCE 14
3.2 DEVICE DISCOVERY AND PNP 14
3.3 FINGERPRINT CAPTURE 15

4. NOTES & CLARIFICATIONS 16

4.1 SUPPORTING DIFFERENT VIDEO FORMATS 16
4.1.1 Iris device showing video of the portion of the face 16
4.1.2 Iris device showing two videos of the two eyes 17

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 to eliminate duplicates and ensure uniqueness during the enrolment process. Quality of collected biometric data is critical for the accuracy of de-duplication and a key component for the success of the program. While the program will be using the biometric capture devices from different vendors, it is critical to maintain consistent data collection process. This will be achieved by standardizing the biometric capture process flow around the UID Enrolment Software.

This Biometric Capture API is to be used by the UID Enrolment Software to communicate with the Biometric Capture Devices.

1.1 Objective of this document

The current version of this document has been provided for feedback from the Application developers, as well as Biometric Capture Device manufacturers. A final API document will be released after their feedback has been incorporated.

1.2 Interface Overview

The API is specified as communication protocol between the UID Enrolment Software and the Biometric Capture Devices, and also between the Device Manager and the Vendor-specific Device Manager (VDM).

The UID Enrolment Software will use a TCP/IP sockets to communicate with the Biometric Capture Devices. This serves two purposes. First is isolation: the software from each vendor will be executed in a separate process. Second is platform-independence: the devices will be directly accessible from the different platforms and environments: native, Java, .NET. The communication will be done by exchanging the XML messages. The API method will be executed by sending the request message and waiting for the corresponding response message. The response will be sent after the method execution is completed. Multiple API methods, even of the same type, can be executed in parallel.

There will be two types of API methods: *commands* and notification *events*. The *command* API methods are called by UID Enrolment Software, while the *event* API methods are called by the Biometric Capture Device.

Video stream from the Biometric Capture Device will be delivered using the binary protocol over a separate channel. The final captured biometric samples will also be delivered using the binary protocol through a separate channel. The request and response messages for both video stream and biometric samples will be encoded using ASN1 DER. See http://en.wikipedia.org/wiki/Asn1

The proposed protocol does not address any security-related issues. After the security requirements are defined, the protocol may need to be modified to address them.

2. API Methods

Requests and responses will be formatted as XML.

2.1 Biometric Device Management and Discovery

The biometric device manager service will be provided by the UID. This service will responsible for the following:

- Maintain the list of the all the supported biometric devices available for the applications.
- Notify the applications about arrival and removal of the supported devices (PNP).
- Start and shut down the vendor-specific device executables provided from the Biometric Capture Device vendors. Service configuration files determine the executables to start.
- Listen on the device arrival and removal events from the vendor-specific device manager (VDM) executables on the standard input/output of the VDM.

The exact configuration details are TBD.

2.2 VDM Commands

When the biometric device manager service is requested to shutdown, it will try to shutdown all the VDMs. If the VDM does not respond to shutdown request within reasonable time, the biometric device manager can try to kill the VDM process. When the biometric device manager service starts, it will try to kill all the currently running VDMs, before starting any new ones.

2.2.1 Shutdown

The Shutdown request will be sent by the device manager to the VDM on the standard input in order to initiate the VDM shutdown. The response will be sent back on the standard output and this will be the last message from the VDM before the VDM process terminates.

```
<DeviceManagerCommandRequest RequestID=”999”>
  <Shutdown/>
</DeviceManagerCommandRequest>
```

```
<DeviceManagerCommandResponse RequestID=”999”>
  <Return Value=”1” FailureReason=”0”/>
</DeviceManagerCommandResponse>
```

2.2.2 Reset Device

The Reset Device request will be sent by the device manager to the VDM on the standard input in order to reset the specific device, for example if the device is unresponsive or is behaving incorrectly. The response will be sent back on the standard output after the device has been reset. The VDM may try to disconnect and re-connect the device in response to the reset request, in order to force the application to re-establish the connection to the device. In this case the response to Reset Device command should be sent after the VDM sends Device Arrival event following Device Removal event. If VDM cannot restore the connection with the device (unrecoverable failure) the VDM should send the response to Reset Device command after sending the Device Removal, indicating that the Reset Device command has failed.

```
<DeviceManagerCommandRequest RequestID=”999”>
  <ResetDevice DeviceURI=”localhost:1234”/>
</DeviceManagerCommandRequest>
```

```
<DeviceManagerCommandResponse RequestID=”999”>
  <Return Value=”1” FailureReason=”0”/>
</DeviceManagerCommandResponse>
```

2.3 VDM Events

The device arrival and removal events will be sent by the each VDM executable to the standard output. The responses are sent to VDM on the standard input. The VDM executable will start it will send the device arrival event for each device already present in the system. Every time the PNP device is connected the VDM will send device arrival event, and every time the PNP device is disconnected the VDM will send device removal event.

2.3.1 Device Arrival

Will notify the device manager about the device arrival.

```
<DeviceManagerEventRequest RequestID=”999”>
  <Arrival
     DeviceURI=”localhost:1234”
     Modality=”Fingerprint Slap”
     DeviceMake=”Manufacturer Name”
     DeviceModel=”DEVICE MODEL NAME / IDENTIFIER”
     HardwareRev=”1.0.0”
     FirmwareRev=”1.0.1”
     SerialNumber=”ABC1234567”
  >
    <Capabilities
       Detection=”True”
       Video=”True”
       AutoCapture=”True”
       DisableAutoCapture=”True”
       UserFeedback=”True”
       GraphicalFeedback=”False”
    >
      <VideoFormats>
        <VideoFormat VideoFormatID =”1”
          Modality=”Fingerprint Slap”>
          <FrameType
            Position=”Any”
            Size=”800,750”
            PixelFormat=”Gray8”
            PixelResolution=”250ppi”
          />
        </VideoFormat>
      </VideoFormats>
      <SampleFormats>
        <SampleFormat FormatID=”1”
          SampleFormatID=”ISO IEC 19794-4 2005”
          Views=”1”
          Size=”1600,1500”
          PixelResolution=”500ppi”
        />
      </SampleFormats>
    </Capabilities>
  </Arrival>
</DeviceManagerEventRequest>
```

```
<DeviceManagerEventResponse RequestID=”999”>
  <Return Value=”1” FailureReason=”0”/>
</DeviceManagerEventResponse>
```

DeviceURI is URI, where the device listens to connection from the application. It is also used to uniquely identify the device object in the system.

Modality is the biometric modality, for example: “Fingerprint” for single-finger devices, “Fingerprint Slap” for the slap devices, “Iris”, “Face”.

DeviceMake is the manufacturer or brand of the device.

SerialNumber is the serial number of the device. The combination of DeviceMake, DeviceModel and SerialNumber should globally uniquely identify the specific device. This is important, for example to identify all biometric samples that originated from the specific device, for example in the case of device malfunction.

Capabilities

- *Detection* tells whether the device can automatically detect the presence of the biometric sample, i.e. if the fingerprint reader can detect if the finger is placed on the platen, even if the device is not capturing the data.
- *Video* tells if the device can produce a video stream during capture, useful for the operator.
- *AutoCapture* tells whether the device can capture the biometric sample automatically.
- *DisableAutoCapture* tells whether the automatic capture can be disabled.
- *UserFeedback* tells if the device can provide the actionable user feedback, compliant with this specification.
- *GraphicalFeedback* tells if the device can provide the additional graphical feedback for the operator in the video.

The Device Arrival event should be sent after the device component starts listening for the connections on the socket addressed by the *DeviceURI.*

2.3.2 Device Removal

Will notify the device manager about the device removal

```
<DeviceManagerEventRequest RequestID=”999”>
  <Removal DeviceURI=”localhost:1234”/>
</ DeviceManagerEventRequest>
```

```
<DeviceManagerEventResponse RequestID=”999”>
  <Return Value=”1” FailureReason=”0”/>
</DeviceManagerEventResponse>
```

The device component of the removed device should be listening on the *DeviceURI* until the response is received. After the response is received it can close all sockets of this device that not closed already by the application.

2.4 Biometric Device Management API Methods

When the application starts it connects to the Biometric Device Manager, listening on the pre-defined port. The device manager sends the Device Arrival events for all the connected devices to the application, and later sends all the Device Arrival/Device Removal events for the PNP devices. The protocol for the Device Arrival/Device Removal is the same as the corresponding VDM events, but the socket is used in place of the standard I/O.

2.5 Biometric Device Command API Methods

When the application needs to work with the device, it opens the socket to the location pointed by the *DeviceURI* in the Arrival event from the Device Manager. Opening the socket effectively opens the device. Only one application can open the device. The device should reject the connections on the *DeviceURI*, until the socket is closed.

2.5.1 Subscribe

Change the subscription to the device events: uses by the application to subscribe or unsubscribe to the specific categories of the device events. Some events will be fired only when capture is in progress.

```
<DeviceCommandRequest RequestID=”999”>
  <Subscribe>
    <Event EventCategory=”Detection”/>
    <Event EventCategory=”UserFeedback”/>
  </Subscribe>
</DeviceCommandRequest>
```

```
<DeviceCommandResponse RequestID=”999”>
  <Return Value=”1” FailureReason=”0”/>
  <State Detected=”False”/>
</DeviceCommandResponse>
```

Expected behaviour: change the list of subscribed events to the list in this command request. The events to which the device was previously subscribed can be sent by the device until the response is sent. Any new events can only be fired after the response is sent. If the command is subscribing for the notification to the change of state the response should include the corresponding state at a time of response.

2.5.2 Start Capture

Starts the capture process, also subscribes to Capture Complete and optionally User Feedback events.

```
<DeviceCommandRequest RequestID="999">
  <StartCapture
     BiometricPosition="13"
     AllowManualCapture="True"
     [VideoFormatID="1"]
     SampleFormatID="1"
  >
    [<MissingBiometrics>
      <MissingBiometric Position="6"/>
    <MissingBiometrics>]
  </StartCapture>
</DeviceCommandRequest>
```

```
<DeviceCommandResponse RequestID="999">
  <Return Value="1" FailureReason="0"/>
  [<Video VideoURI="localhost:2345"/>]
</DeviceCommandResponse>
```

Expected behaviour: starts the capture process. Any capture related event can be sent only after the response to the start capture event. *MissingBiometrics* is optional. Attribute *SampleFormatID* is indicating the requested output sample format. Optional attribute *VideoFormatID* is indicating that the video stream is requested, and the desired video format referred by *VideoFormatID* in the Device Arrival VDM event in "*Capabilities/VideoFormats/VideoFormat*".

2.5.3 Force Capture

Forces manual capture. Should not be issued when the capture is not started.

```
<DeviceCommandRequest RequestID="999">
  <ForceCapture/>
</DeviceCommandRequest>
```

```
<DeviceCommandResponse RequestID="999">
  <Return Value="1" FailureReason="0"/>
</DeviceCommandResponse>
```

Expected behaviour: force manual capture, whether the automatic capture is on or off. The capture complete event is sent right after the response to this event. If the capture complete event comes before the response, it means the event resulted from the automatic capture.

2.5.4 Stop Capture

Stops (cancels) current capture

```
<DeviceCommandRequest RequestID="999">
  <StopCapture/>
</DeviceCommandRequest>
```

```
<DeviceCommandResponse RequestID="999">
  <Return Value="1" FailureReason="0"/>
</DeviceCommandResponse>
```

Expected behaviour: stops capture process. No capture complete event should come after the response to Stop Capture.

2.6 Biometric Device Notification API Methods

The following methods must be called by the Device component to notify the UID Enrolment Software about the subscribed events. All requests and responses carry a RequestID, which is a numeric value (potentially very large, up-to 128 bit). Since the API is asynchronous, the RequestID is used to connect requests with the appropriate response. The biometric solution must not use RequestID outside the scope of a request, since this could be recycled.

2.6.1 Capture Complete

```
<DeviceEventRequest RequestID="999">
  <CaptureComplete SampleURI="localhost:1234"/>
</DeviceCommandRequest>
```

```
<DeviceEventResponse RequestID="999">
  <Return Value="1" FailureReason="0"/>
</DeviceCommandResponse>
```

Expected behaviour: the event should be sent upon successful completion of the capture. The biometric sample should be available until the response is received: as a result the device may have to maintain multiple samples and make them available at the different URIs. Samples must be provided in the format indicated by *SampleFormatID* in the Start Capture request.

2.6.2 Detection

```
<DeviceEventRequest RequestID="999">
  <Detection Detected="True"/>
</DeviceCommandRequest>
```

```
<DeviceEventResponse RequestID="999">
  <Return Value="1" FailureReason="0"/>
</DeviceCommandResponse>
```

Expected behaviour: notifies of the change of the state whether the biometric is detected by the device (for example if the finger is placed or removed from the reader).
Usage example: the application needs to capture one finger, followed by the other. The application issues capture request, which completes. The now application needs to wait for the finger removed event before issuing the next capture request, to make sure that the same finger is not captured again immediately.

2.6.3 User Feedback

```
<DeviceEventRequest RequestID=”999”>
  <UserFeedback>
     <Message Action=”MoveLeft”/>
     <Message Action=”PresentBiometric”/>
     <Message BioPosition=”6” Action=”PressHarder”/>
     <Message BioPosition=”7” Action=”PressLighter”/>
  </UserFeedback>
</DeviceCommandRequest>
```

```
<DeviceEventResponse RequestID=”999”>
  <Return Value=”1” FailureReason=”0”/>
</DeviceCommandResponse>
```

Expected behaviour: provide the actionable feedback.

2.7 Biometric Device Video Streaming and Sample API Methods

The Video Stream will be retrieved using the binary protocol for sending video frames (with the actual image data represented in ISO 19794-x) over the socket referenced by *VideoURI* using the pull model. The application will maintain pending Get Frame requests for all the time that it can keep-up with the visualization.

The sample will be retrieved over the socket referenced by *SampleURI*, with the actual image data represented in ISO 19794-x.

The requests and the responses are represented in ASN1 BER/DER encoding. The rationale in choosing BER/DER encoding is to transfer the binary data in the self-descriptive extensible data format with the reasonably low overhead.

```
MessageType ::= ENUMERATED { Request, Response }
RequestCode ::= ENUMERATED { GetSample, GetFrame, ... }

RequestMessage ::= SEQUENCE {
  messageType MessageType ::= Request,
  requestCode RequestCode,
  requstId    INTEGER,
  ...
}
```

```
Return ::= SEQUENCE {
  returnValue INTEGER { Success(1), Failed(2) }
                      (Success | Failed),
  failureReason INTEGER
}

BiometricSample ::= SEQUENCE {
  format OBJECT IDENTIFIER,
  data OCTET STRING
}
```

```
ResponseMessage ::= SEQUENCE {
  messageType MessageType ::= Response,
  requstId    INTEGER,
  return      Return,
  ...
}
```

2.7.1 Get Frame

```
RequestMessage ::= SEQUENCE {
  messageType MessageType ::= Request,
  requestCode RequestCode ::= GetFrame,
  requstId    INTEGER
}
```

```
ResponseMessage ::= SEQUENCE {
  messageType MessageType ::= Response,
  requstId    INTEGER,
  return      Return,
  biometricSample BiometricSample
}
```

2.7.2 Get Sample

```
RequestMessage ::= SEQUENCE {
  messageType MessageType ::= Request,
  requestCode RequestCode ::= GetSample,
  requstId    INTEGER
}
```

```
ResponseMessage ::= SEQUENCE {
  messageType MessageType ::= Response,
  requstId    INTEGER,
  return      Return,
  biometricSample BiometricSample
}
```

2.8 Return Codes

The following return codes are standard for all the Bio API requests.

0	Not Used
1	Success
2	Failed

The following failure reasons are also used for all Bio API requests

0	Success

1	Internal Error – Unknown
2	Aborted
3	Invalid Request –XML Error
4	Invalid Request –Parameter value invalid
5	Invalid Request – Capture already in progress
6	Invalid Request – Capture is not in progress
7	Unexpected Error – Unable to access Biometric Data
8	Unable to perform request

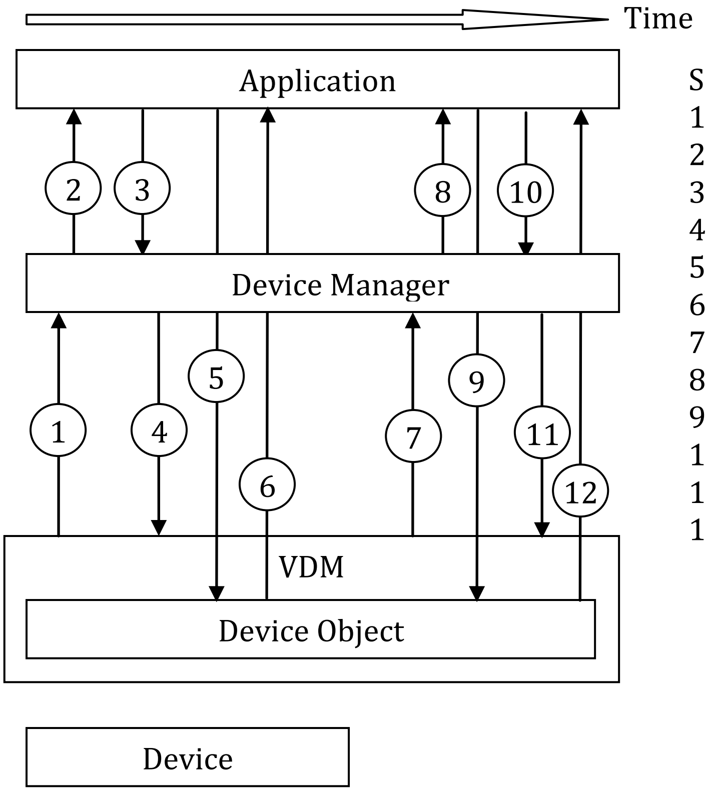
3. Workflows

These workflows are indicative, and provided for a better understanding of the use case of this API. Other uses may be made of the API, and the vendors should not assume only these workflows.

3.1 Start Sequence

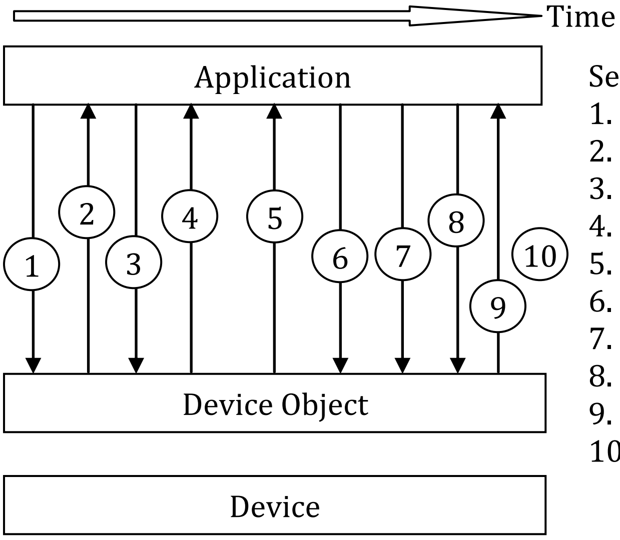
1. Device Manager Service starts
2. Device Manager Service reads and processes configuration file
3. Device Manager Service starts VDMs one by one
   3.1. VDM starts
   3.2. VDM notifies device manager about each device detected
4. Application starts and connects to device manager
5. Device manager notifies the application about all the devices
6. Application opens the device by connecting to URI that the VDM listens on
7. Application talks to device object maintained by VDM directly

3.2 Device Discovery and PNP



Sequence of events:
1. Device Arrive Request
2. Device Arrive Request
3. Device Arrive Response
4. Device Arrive Response
5. Socket Connect
6. Socket Accept
7. Device Remove Request
8. Device Remove Request
9. Application Closes Socket
10. Device Remove Response
11. Device Remove Response
12. VDM Closes Socket

3.3 Fingerprint Capture



Sequence of events:

1. Start Capture Request
2. Start Capture Response
3. Video Socket Connect
4. Video Socket Accept
5. Capture Complete Event Request
6. Application Closes Video Socket
7. Application Reads Sample
8. Capture Complete Event Response
9. Device Object Closes Video Socket
10. Sample is no longer available

4. Notes & Clarifications

4.1 Supporting Different Video Formats

Example: two-eye Iris camera can show video of either portion of the face with two eyes in it (like LG) or the two eyes separately (like Manufacturer Name). In the second case the frame header describes which eye is shown in this frame.

The application needs to distinguish between the two types of the devices before opening the device, to choose which device to open and to show the appropriate UI. This information is provided in the Device Arrival event from the VDM.

4.1.1 Iris device showing video of the portion of the face

The Device Arrival event:

```
<DeviceManagerEventRequest RequestID=”999”>
  <Arrival
     DeviceURI=”localhost:1234”
     Modality=”Two Iris”
     DeviceMake=”LG”
     DeviceModel=”iCAM TD100”
     HardwareRev=”1.0.0”
     FirmwareRev=”1.0.1”
     SerialNumber=”ABC1234567”
  >
    <Capabilities
       Detection=”True”
       Video=”True”
       AutoCapture=”True”
       DisableAutoCapture=”True”
       UserFeedback=”True”
       GraphicalFeedback=”True”
    >
      <VideoFormats>
        <VideoFormat VideoFormatID =”1”
          Modality=”Face Partial”>
          <FrameType Size=”640,240”
            PixelFormat=”RGB32”
            PixelResolution=”Unspecified”
          />
        </VideoFormat>
      </VideoFormats>
      <SampleFormats>
        <SampleFormat SampleFormatID=”1”
          Format=”ISO IEC 19794-6 2005”
          Views=”2”
          Size=”480,480”
          PixelResolution=”500ppi”
        />
      </SampleFormats>
    </Capabilities>
  </Arrival>
</DeviceManagerEventRequest>
```

4.1.2 Iris device showing two videos of the two eyes

The Device Arrival event:

```
<DeviceManagerEventRequest RequestID=”999”>
  <Arrival
     DeviceURI=”localhost:1234”
     Modality=”Two Iris”
     DeviceMake=”Manufacturer Name”
     DeviceModel=”I SCAN 2”
     HardwareRev=”1.0.0”
     FirmwareRev=”1.0.1”
     SerialNumber=”ABC1234567”
  >
    <Capabilities
       Detection=”True”
       Video=”True”
       AutoCapture=”True”
       DisableAutoCapture=”True”
       UserFeedback=”True”
       GraphicalFeedback=”False”
    >
      <VideoFormats>
        <VideoFormat VideoFormatID=”1” Modality=”Two Iris”>
          <FrameType Position=”Left Eye”
            Size=”240,240”
            PixelFormat=”Gray8”
            PixelResolution=”250ppi”
          />
          <FrameType Position=”Right Eye”
            Size=”240,240”
            PixelFormat=”Gray8”
            PixelResolution=”250ppi”
          />
        </VideoFormat>
      </VideoFormats>
      <SampleFormats>
        <SampleFormat SampleFormatID=”1”
          Format=”ISO IEC 19794-6 2005”
          Views=”2”
          Size=”480,480”
          PixelResolution=”500ppi”
        />
      </SampleFormats>
    </Capabilities>
  </Arrival>
</DeviceManagerEventRequest>
```