# MANTRA.MFS100 ANDROID SDK

Ver.: 9.0.3.2

Mantra Softech India Pvt. Ltd.

| Version | 9.0.3.2 |
|---|---|
| Release Date | 30th July, 2019 |
| Author | Mr. Mahesh Patel |
| Software Support | softwaresupport@mantratec.com<br>079-49068000 (Extension: 1) |

## About:

The document provides the functional and implementation information to work with MFS100 (Mantra Fingerprint Sensor). By using this SDK, you can capture fingerprint from MFS100. This SDK provided facility to extract different-different fingerprint formats like –

- Bitmap Image
- Raw fingerprint image
- ISO (ISO-19794-2/FMR) Template
- ANSI (ANSI-19794-2) Template
- ISO (ISO-19794-4/FIR) Image
- WSQ Image
- Quality of fingerprint
- NFIQ of fingerprint

## Dependency:

1. USB HOST ENABLED in device
2. mantra.mfs100.jar
3. libMFS100V9032.so

## Contents:

**Interface Implementation**

```
public class MFS100SampleWrapper extends Activity implements MFS100Event
```

**Initialization of MFS100 Class:**

```
MFS100 mfs100 = new MFS100(this);
```

**General Functions:**

```
private void DisplayFinger(final Bitmap bitmap) {
      imgFinger.post(new Runnable() {
              @Override
              public void  run() {
                     imgFinger.setImageBitmap(bitmap);
              }
});
}

private void SetTextonuiThread(final String str) {
      lblMessage.post(new Runnable() {
              public void  run() {
                     lblMessage.setText(str, BufferType.EDITABLE);
              }
      });
}

private void SetLogOnUIThread(final String str) {
      txtEventLog.post(new Runnable() {
              public void  run() {
                     txtEventLog.setText(txtEventLog.getText().toString() + "\n"
                                         + str, BufferType.EDITABLE);
              }
      });
}

Private void WriteFile(String filename, byte[] bytes) {
      try {
              String path = Environment.getExternalStorageDirectory()+ "//FingerData";
              File file = new File(path);
              if (!file.exists()) {
                     file.createNewFile();
              }
              path = path + "//" + filename;
              file = new File(path);
              if (!file.exists()) {
                     file.createNewFile();
              }
              FileOutputStream stream = newFileOutputStream(path);
              stream.write(bytes);
              stream.close();
      } catch (Exception e1) {
              e1.printStackTrace();
      }
}
```

1. **SetApplicationContext(Context context)**
   **Return:** Boolean
   Set current application context sdk. Function should be used in "OnCreate" event of activity.

```
mfs100.SetApplicationContext(this);
```

2. **GetSDKVersion()**
   **Return:** String

   Return the version of SDK.

```
string version = mfs100.GetSDKVersion();
SetTextonuiThread("SDK Version: " + version);
```

3. **IsConnected()**
   **Return:** Boolean

   Used to check whether scanner is connected or not. It will return boolean true if scanner is connected, else boolean false.

```
if (mfs100.IsConnected())
{
SetTextonuiThread("Device Connected");
}
else
{
SetTextonuiThread("Device not connected");
}
```

4. **LoadFirmware()**

   **Return:** Integer

   Used to load firmware in device. Return 0 if success.

```
int ret = mfs100.LoadFirmware();
if (ret != 0)
{
SetTextonuiThread(mfs100.GetErrorMsg(ret));
}
else
{
SetTextonuiThread("Success");
}
```

5. **Init()**

   **Return:** Integer

   Used to initialize scanner. Return 0 if success.

```
DeviceInfodeviceInfo = null;
int ret = mfs100.Init();//You can pass key here for locked sensor.
if (ret != 0)
{
SetTextonuiThread(mfs100.GetErrorMsg(ret));
}
else
{
deviceInfo = mfs100.GetDeviceInfo();
if (deviceInfo != null)
     {
       stringInfo = "SERIAL: " + deviceInfo.SerialNo + " MAKE: " + deviceInfo.Make + " MODEL:
       " + deviceInfo.Model;
       SetTextonuiThread(Info);
}
else
     {
SetTextonuiThread("Error");
    }

}
```

6. **GetDeviceInfo()**

   **Return:** DeviceInfo

   Function can be used after successful initialization of scanner. It will return object of DeviceInfo class which contains device related information like serial number, make, model, height of scanner image and width of scanner image.

Please see Init() Method

7. **GetCertification()**
   **Return:** String
   Returns MFS100 Certification information.

```
String cert = mfs100.getCertification();
if (cert != "")
{
        SetTextonuiThread(cert);
}
else
{
        SetTextonuiThread("MFS100 not initialized");
}
```

8. **StopCapture()**
   **Return:** Integer

   Function is used to stop capturing of fingerprint from device against StartCapture function. It return 0 if scanning stopped successfully.

```
int ret = mfs100.StopCapture();
SetTextonuiThread(mfs100.GetErrorMsg(ret));
```

9. **AutoCapture(FingerData fingerprintData, int TimeOut, boolean DetectFinger)**
   **Return:** Integer
   **FingerData:** object of FingerData as reference
   **Timeout:** value in milliseconds to stop capture automatically (default = 10000, 0 = no limit).
   **DetectFinger:** Boolean, will detect if finger is placed properly or not if value is passed as true.

   Function is used to capture fingerprint form device in synchronous mode. It returns 0 if captured successfully. It will raise OnPreview if the parameter ShowPreview has been set to true. In from version 9.0.2.5 additional finger formats like ANSI, WSQ, ISO Image conversion has been removed from this function, and separate functions are added for those conversion.

```
new Thread(new Runnable() {
        @Override
        public void  run() {
                SetTextonuiThread("");
                try {
                        FingerDatafingerData = newFingerData();
                        int ret = mfs100.AutoCapture(fingerData, timeout, true);
                        if (ret != 0) {
                                SetTextonuiThread(mfs100.GetErrorMsg(ret));
                        } else {
                                SetTextonuiThread("Quality: " + fingerData.Quality()
                                                                + " NFIQ: " + fingerData.Nfiq());
                                SetData1(fingerData);
                        }
                } catch (Exception ex) {
                        SetTextonuiThread("Error");
                }
        }
}).start();

public void SetData1(FingerDatafingerData) {
        WriteFile("Raw.raw", fingerData.RawData());
        WriteFile("Bitmap.bmp", fingerData.FingerImage());
        WriteFile("ISOTemplate.iso", fingerData.ISOTemplate());
}
```

**10. MatchISO(byte[] probeISO, byte[] galleryISO)**

**Return:** Integer (error code or score)
**probeISO:** First ISO(FMR) Template Bytes
**galleryISO:** SecondISO(FMR) Template Bytes
**score:**Matching score 0 to 1000

Function is used to match two ISO Templates. It will return 0 if function executed successfully. Score defines the matching value of two templates. If score >=96 then it is considered as matched. Else not matched.

```
int ret = mfs100.MatchISO(ISOTemplate1, ISOTemplate2);
if (ret >= 0)
{
if (ret>= 96)
    {
SetTextonuiThread("Finger matched with score: " + ret);
    }
else
    {
SetTextonuiThread("Finger not matched, score: "+ ret + " is too low",);
    }
}
else
SetTextonuiThread(mfs100.GetErrorMsg(ret));
}
```

**11. MatchANSI(byte[] probeANSI, byte[] galleryANSI)**

**Return:** Integer (error code or score)
**probeANSI:** First ANSI Template Bytes
**galleryANSI:** Second ANSITemplate Bytes

Function is used to match two ANSI Templates. It will return 0 if function executed successfully. Score defines the matching value of two templates. If score >=96 then it is considered as matched. Else not matched.

```
int ret = mfs100.MatchANSI(ANSITemplate1, ANSITemplate2);
if (ret >= 0)
```

```
{
if (ret>= 96)
    {
SetTextonuiThread("Finger matched with score: " + ret);
    }
else
    {
SetTextonuiThread("Finger not matched, score: " + ret + " is too low");
    }
}
else
SetTextonuiThread(mfs100.GetErrorMsg(ret));
}
```

**12. RotateImage(int Direction)**

**Return:** Boolean

**Direction:** angle of rotation 0 or 180

Function is used to rotate image as per 0 or 180 degree. It will return true if success.

```
bool ret = mfs100.RotateImage(180);
if (ret == true)
{
SetTextonuiThread("Success");
}
else
{
SetTextonuiThread("Failed");
}
```

**13. Uninit()**

**Return:** Integer

Function is used to uninitialized scanner. All object will dispose after de-initialization. It will return 0 if success.

```
int ret = mfs100.Uninit();
SetTextonuiThread(mfs100.GetErrorMsg(ret));
```

**14. GetErrorMsg(int errorCode)**

**Return:** String

**errorCode:** Integer error code return by each function.

Function is used to analyze the error codes returns by functions. It will return error description of error code.

```
int ret ;//Coming from other sdk function
string errorDescription= mfs100.GetErrorMsg(ret);
```

**15. OnDeviceAttached(int vid, int pid, Boolean hasPermission)**

**Type:** Event

**vid:** integer vendor id of device

**pid:** integer product id of device

**hasPermission:** Boolean value as per user has treated permission dialog.

This event will raised once application context has been set to object of MFS100 class. Even this event will raised when you attached scanner to your android device every time. In this event you can call LoadFirmware() or Init() function for device initialization automatically without user intervention. Based on pid, you can decided that whether you have to call LoadFirmware() or Init(). If pid = 34323 and hasPermission is true then call LoadFirmware(), and if pid = 4101 and hasPermissioin is true then call Init().

```
@Override
public void OnDeviceAttached(int vid, intpid, booleanhasPermission) {
        int ret = 0;
        if (!hasPermission) {
                SetTextonuiThread("Permission denied");
                return;
        }
        if (vid == 1204 || pid == 11279) {
                if (pid == 34323) {
                        ret = mfs100.LoadFirmware();
                        if (ret != 0) {
                                SetTextonuiThread(mfs100.GetErrorMsg(ret));
                        } else {
                                SetTextonuiThread("Loadfirmware success");
                        }
                } elseif (pid == 4101) {
                        ret = mfs100.Init();
                        if (ret != 0) {
                                SetTextonuiThread(mfs100.GetErrorMsg(ret));
                        } else {
                                SetTextonuiThread("Init success");
                                String info = "Serial: "
                                                + mfs100.GetDeviceInfo().SerialNo() + " Make: "
                                                + mfs100.GetDeviceInfo().Make() + " Model: "
                                                + mfs100.GetDeviceInfo().Model();
                                SetLogOnUIThread(info);
                        }
                }
        }
}
```

**16. OnDeviceDetached()**

**Type**: Event

Occurs when scanner removed from android device. You can call Uninit() here.

```
@Override
public void OnDeviceDetached() {
        SetTextonuiThread("Device removed");
        UnInitScanner();
}
```

**17. ExtractISOTemplate(byte[] RawData,byte[] Data)**

**Return:** Integer (Length of ISOTemplate)

**RawData:** Byte array of raw image

**Data:** Blank Byte array with length of 2000

```
byte[] tempData = newbyte[2000]; // length 2000 is mandatory
byte[] isoTemplate = null;
intdataLen = mfs100.ExtractISOTemplate(fingerData.RawData(), tempData);

if(dataLen<=0)
{
        if(dataLen==0)
        {
                SetTextonuiThread("Failed to extract ISO Template");
        }
        else
        {
                SetTextonuiThread(mfs100.GetErrorMsg(dataLen));
        }
        return;
}
else
{
        isoTemplate = newbyte[dataLen];
        System.arraycopy(tempData, 0, isoTemplate, 0, dataLen);
}
```

**18. ExtractANSITemplate(byte[] RawData, byte[] Data)**

**Return:** Integer (Length ofANSITemplate)

**RawData:** Byte array of raw image

**Data:** Blank Byte array with length of 2000

```
byte[] tempData = newbyte[2000]; // length 2000 is mandatory
byte[] ansiTemplate = null;
intdataLen = mfs100.ExtractANSITemplate(fingerData.RawData(), tempData);

if(dataLen<=0)
{
        if(dataLen==0)
        {
                SetTextonuiThread("Failed to extract ANSI Template");
        }
        else
        {
                SetTextonuiThread(mfs100.GetErrorMsg(dataLen));
        }
        return;
}
else
{
        ansiTemplate = newbyte[dataLen];
        System.arraycopy(tempData, 0, ansiTemplate, 0, dataLen);
}
```

**19. ExtractISOImage(byte[] RawData, byte[] Data, int isoType)**

**Return:** Integer (Length of ISOImage)

**RawData:** Byte array of raw image

**Data:** Blank Byte array with length of ((scanner width * height)+1078)

**isoType:** isoType=1 means extract ISOImage template without compression

isoType=2 means extract ISOImage template with compression (WSQ Compression)

```
tempData = newbyte[(mfs100.GetDeviceInfo().Width() *
mfs100.GetDeviceInfo().Height())+1078];
byte[] isoImage = null;
intdataLen = mfs100.ExtractISOImage(fingerData.RawData(),tempData, 2);
if(dataLen<=0)
{
        if(dataLen==0)
        {
                SetTextonuiThread("Failed to extract ISO Image");
        }
        else
        {
                SetTextonuiThread(mfs100.GetErrorMsg(dataLen));
        }
        return;
}
else
{
        isoImage = newbyte[dataLen];
        System.arraycopy(tempData, 0, isoImage, 0, dataLen);
}
```

**20. ExtractWSQImage(byte[] RawData, byte[] Data)**

**Return:** Integer (Length of WSQImage)

**RawData:** Byte array of raw image

**Data:** Blank Byte array with length of ((scanner width * height) +1078)

```
tempData = newbyte[(mfs100.GetDeviceInfo().Width() *
mfs100.GetDeviceInfo().Height())+1078];
byte[] wsqImage = null;
intdataLen = mfs100.ExtractWSQImage(fingerData.RawData(), tempData);
if(dataLen<=0)
{
        if(dataLen==0)
        {
                SetTextonuiThread("Failed to extract WSQ Image");
        }
        else
        {
                SetTextonuiThread(mfs100.GetErrorMsg(dataLen));
        }
        return;
}
else
{
        wsqImage = newbyte[dataLen];
        System.arraycopy(tempData, 0, wsqImage, 0, dataLen);
}
```

**21. Dispose()**

MFS100 class is Disposible, so it is compulsory to call this method on onDestroy event of activity.

```
@Override
protectedvoidonDestroy() {
        if (mfs100 != null) {
                mfs100.Dispose();
        }
        super.onDestroy();
}
```

**22. DeviceInfo Class**

| **Property** | **Description** |
|---|---|
| SerialNo() | Serial number of scanner |
| Make() | Make of scanner |
| Model() | Model of scanner |
| Width() | Width of image comes from scanner |
| Height() | Height of image comes from scanner |

**23. FingerData Class**

| **Property** | **Description** |
|---|---|
| FingerImage() | Byte array of finger bitmap image |
| RawData() | Byte array of finger raw data |
| ISOTemplate() | Byte array of ISO 19794-2 Template (FMR) |
| ISOImage() | Byte array of ISO19794-4 Image (FIR) |
| ANSITemplate() | Byte array of ANSI template |
| WSQImage() | Byte array of WSQ Image |
| Quality() | Quality of fingerprint |
| Nfiq() | Nfiq of fingerprint |
| InWidth() | Image width in inch |
| InHeight() | Image height in inch |
| InArea() | Fingerprint image area in inch |

| | |
|---|---|
| Resolution() | Fingerprint image resolution (DPI/PPI) |
| GrayScale() | Gray level of fingerprint image |
| Bpp() | Bits per pixel of fingerprint image |
| WSQCompressRatio() | Compression ratio of WSQ image |
| WSQInfo() | WSQ image format information |

-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-