MANTRA.MFS100 WINDOWS - .NET

Ver.: 9.0.2.4

Mantra Softech India Pvt. Ltd.

Version	9.0.2.7
Release Date	1st Feb, 2017
Author	Mr. Mahesh Patel
Software Support	softwaresupport@mantratec.com +91-92-272-66-229, +91-83-470-02-127

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 –

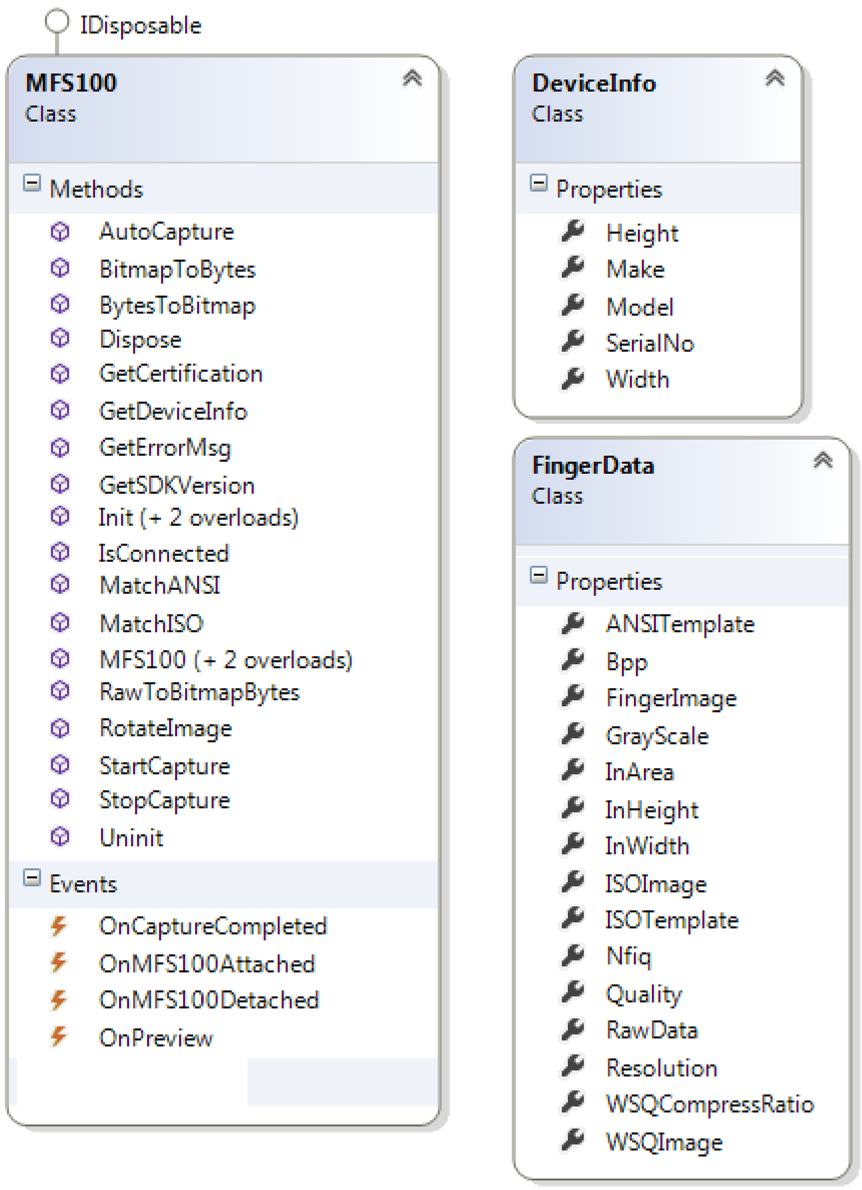
- Bitmap Image
- Raw fingerprint image
- ISO (ISO-19794-2/FMR) Template
- ANSI (ANSI-19794-2) Template
- ISO (ISO-19794-4/FIR) Image
- WSQ Image
- Quality of fingerprint
- NFIQ of fingerprint

Dependency:

1. VC++ 2008 redistributable
2. Microsoft .Net Framework 3.5
3. MANTRA.MFS100.dll
4. MFS100 Driver

Contents:

Class Diagram:



Initialization of MFS100 Class:

```
        MFS100 mfs100 = new MFS100();
        OR
        MFS100 mfs100 = new MFS100(String Key); //For locked Sensor
        OR
        MFS100 mfs100 = new MFS100(Byte[] Key); //For locked Sensor
        mfs100.OnMFS100Attached += OnMFS100Attached;
        mfs100.OnPreview += OnPreview;
        mfs100.OnCaptureCompleted += OnCaptureCompleted;
```

General Fucntion:

```
void ShowMessage(string msg, bool iserror)
{
MessageBox.Show(msg, "MFS100", MessageBoxButtons.OK, (iserror ?   MessageBoxIcon.Error
: MessageBoxIcon.Information), MessageBoxDefaultButton.Button1);
}
```

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

   Return the version of SDK.

```
string version = mfs100.GetSDKVersion();
ShowMessage("SDK Version: " + version, false);
```

2. **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())
{
      ShowMessage("Device Connected", false);
}
else
{
      ShowMessage("Device not connected", true);
}
```

3. **Init()**
   Return: Integer

   Used to initialize scanner. Return 0 if success.

```
DeviceInfo deviceInfo = null;
int ret = mfs100.Init(); //You can pass key here if you have not passed in initialization of
class.
if (ret != 0)
{
    ShowMessage(mfs100.GetErrorMsg(ret), true);
}
else
{
     deviceInfo = mfs100.GetDeviceInfo();
if (deviceInfo != null)
     {
string scannerInfo = "SERIAL NO.: " + deviceInfo.SerialNo + " MAKE: " + deviceInfo.Make + "
MODEL: " + deviceInfo.Model;
        lblSerial.Text = scannerInfo;
}
else
     {
        lblSerial.Text = "";
     }
     ShowMessage(mfs100.GetErrorMsg(ret), false);
}
```

4. **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

5. **GetCertification()**
   Return: String

```
string cert = mfs100.GetCertification();
if(cert != "")
{
    ShowMessage(cert, false);
}
else
{
    ShowMessage("MFS100 not initialized", true);
}
```

6. **StartCapture(**int Quality, int TimeOut, bool ShowPreview**)**
   Return: Integer
   Quality: Threshold for minimum required quality (1 to 1000, Default = 60).
   Timeout: value in milliseconds to stop capture automatically (default = 10, 0 = no limit).
   ShowPreview: it will define that whether preview event is to generate or not.

Function is used to start capturing of fingerprint from device in asynchronous mode. It will return 0 if scanning started successfully. It will raise OnPreview if the parameter ShowPreview has been set to true. After completion of scanning it will raise OnCaptureCompleted events.

```
int ret = mfs100.StartCapture(quality, timeout, true);
if (ret != 0)
{
    ShowMessage(mfs100.GetErrorMsg(ret), true);
}
```

7. **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();
ShowMessage(mfs100.GetErrorMsg(ret), false);
```

8. **AutoCapture(**ref FingerData fingerprintData, int TimeOut, bool ShowPreview bool IsDetectFinger)
   Return: Integer
   FingerData: object of Fingerdata as reference
   Timeout: value in milliseconds to stop capture automatically (default = 10, 0 = no limit).
   ShowPreview: it will define that whether preview event is to generate or not.
   IsDetectFinger: it will define that whether detect finger before/while capture or not.

   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.

```
FingerData fingerprintData = null;
int ret = mfs100.AutoCapture(ref fingerprintData, timeout, true, true);
if (ret != 0)
{
    ShowMessage(mfs100.GetErrorMsg(ret), true);
}
else
{
       string info = "Quality: " + fingerprintData.Quality.ToString() + "     Nfiq: " +
       fingerprintData.Nfiq.ToString() + "     Bpp: " + fingerprintData.Bpp.ToString() + "
       GrayScale:" + fingerprintData.GrayScale.ToString() + "\nW(in):" +
       fingerprintData.InWidth.ToString() + "     H(in):" +
       fingerprintData.InHeight.ToString() + "     area(in):" +
       fingerprintData.InArea.ToString() + "     Dpi/Ppi:" +
       fingerprintData.Resolution.ToString() + "     Compress Ratio:" +
       fingerprintData.WSQCompressRatio.ToString();

       lblStatus.Text = info;
```

```
        File.WriteAllBytes(datapath + "//ISOTemplate.iso", fingerprintData.ISOTemplate);
        File.WriteAllBytes(datapath + "//ISOImage.iso", fingerprintData.ISOImage);
        File.WriteAllBytes(datapath + "//AnsiTemplate.ansi", fingerprintData.ANSITemplate);
        File.WriteAllBytes(datapath + "//RawData.raw", fingerprintData.RawData);
            fingerprintData.FingerImage.Save(datapath + "//FingerImage.bmp",
        System.Drawing.Imaging.ImageFormat.Bmp);
        File.WriteAllBytes(datapath + "//WSQImage.wsq", fingerprintData.WSQImage);

    ShowMessage("Capture Success.\nFinger data is saved at application path", false);
}
```

9. **MatchISO(byte[] probeISO, byte[] galleryISO, ref int score)**
   Return: Integer
   probeISO: First ISO(FMR) Template Bytes
   galleryISO: SecondISO(FMR) Template Bytes
   score: Matching score 0 to 100000

   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 >=14000 then it is considered as matched. Else not matched.

```
int score = 0
int ret = mfs100.MatchISO(ISOTemplate1, ISOTemplate2, ref score);
if (ret == 0)
{
if (score >= 14000)
    {
      ShowMessage("Finger matched with score: " + score.ToString(), false);
    }
else
    {
      ShowMessage("Finger not matched, score: " + score.ToString() + " is too low", false);
    }
}
else
   ShowMessage(mfs100.GetErrorMsg(ret), true);
}
```

10. **MatchANSI(byte[] probeANSI, byte[] galleryANSI, ref int score)**
   Return: Integer
   probeANSI: First ANSI Template Bytes
   galleryANSI: Second ANSITemplate Bytes
   score: Matching score 0 to 100000

   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 >=14000 then it is considered as matched. Else not matched.

```
int score = 0
int ret = mfs100.MatchANSI(ANSITemplate1, ANSITemplate2, ref score);
if (ret == 0)
{
if (score >= 14000)
    {
       ShowMessage("Finger matched with score: " + score.ToString(), false);
    }
else
    {
       ShowMessage("Finger not matched, score: " + score.ToString() + " is too low", false);
    }
}
else
   ShowMessage(mfs100.GetErrorMsg(ret), true);
}
```

11. 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)
{
ShowMessage("Success",false);
}
else
{
ShowMessage("Failed", true);
}
```

12. BitmapToBytes(Bitmap image)

Return: Byte array of bitmap image

Bitmap: bitmap image

Function is used to convert bitmap image to byte array. It will return byte array of bitmap image if function successfully executed. Else throw exception.

```
Bitmap MyBitmapImage ; //Coming from somewhere else
byte[] bitmapBytes = mfs100.BitmapToBytes(MyBitmapImage);
```

13. RawToBitmapBytes(byte[] RawData, int RawWidth, int RawHeight)

Return: Byte array of bitmap image

RawData: Byte array of fingerprint raw data.

RawWidth: Width of scanner/raw image

RawHeight: Height of scanner/raw image

Function is used to convert raw image bytes to bitmap byte array. It will return byte array of bitmap image if function successfully executed. Else throw exception.

```
byte[] bitmapBytes = mfs100.RawToBitmapBytes(FingerData.RawData, DeviceInfo.Width,
DeviceInfo.Height);
```

14. BytesToBitmap(byte[] bitmapBytes)

Return: Bitmap image
bitmapBytes: byte array of bitmap image.

Function is used to convert bitmap bytes to bitmap image. It will return bitmap image if function successfully executed. Else throw exception.

```
Bitmap bitmap = mfs100.BytesToBitmap(bitmapBytes);
```

15. 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();
ShowMessage(mfs100.GetErrorMsg(ret), false);
```

16. 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);
```

17. OnMFS100Attached

Type: Event

This event will be raised after two second of creating an object of MFS100 class. Even this event will be raised when you attached scanner to your windows system every time. In this event you can call Init() function for device initialization automatically without user intervention.

```
ShowMessage("MFS100 found, You can initialized now", false);
```

18. OnPreview(FingerData fingerprintData)

Type: Event

FingerData: object of fingerprint related data.

The event will raise when ShowPreview has been set to true. Here you can preview fingerprint as video on your screen with quality display.

```
void OnPreview(FingerData fingerprintData)
{
try
   {
if (fingerprintData != null)
       {
          picFinger.Image = fingerprintData.FingerImage;
          picFinger.Refresh();
          lblStatus.Text = "Quality: " + fingerprintData.Quality.ToString();
          lblStatus.Refresh();
}
 }
catch (Exception ex)
   {}
}
```

19. **OnCaptureCompleted(bool status, int errorCode, string errorMsg, FingerData fingerprintData)**

Type: Event
status: capturing status
errorCode: if status is true then it is 0 else non zero.
errorMsg: error description of error code
FingerData: object of fingerprint related data.

The event will raise when asynchronous capture is completed with success or false. If status is success, then FingerData will contains the fingerprint related data.

```
void OnCaptureCompleted(bool status, int errorCode, string errorMsg, FingerData
fingerprintData)
{
try
  {
if (status)
     {
       picFinger.Image = fingerprintData.FingerImage;
       picFinger.Refresh();

       string info = "Quality: " + fingerprintData.Quality.ToString() + "     Nfiq: " +
       fingerprintData.Nfiq.ToString() + "     Bpp: " + fingerprintData.Bpp.ToString() + "
       GrayScale:" + fingerprintData.GrayScale.ToString() + "\nW(in):" +
       fingerprintData.InWidth.ToString() + "     H(in):" +
       fingerprintData.InHeight.ToString() + "     area(in):" +
       fingerprintData.InArea.ToString() + "     Dpi/Ppi:" +
       fingerprintData.Resolution.ToString() + "     Compress Ratio:" +
       fingerprintData.WSQCompressRatio.ToString();

       lblStatus.Text = info;
```

```
        File.WriteAllBytes(datapath + "//ISOTemplate.iso", fingerprintData.ISOTemplate);
        File.WriteAllBytes(datapath + "//ISOImage.iso", fingerprintData.ISOImage);
        File.WriteAllBytes(datapath + "//AnsiTemplate.ansi", fingerprintData.ANSITemplate);
        File.WriteAllBytes(datapath + "//RawData.raw", fingerprintData.RawData);
                fingerprintData.FingerImage.Save(datapath + "//FingerImage.bmp",
        System.Drawing.Imaging.ImageFormat.Bmp);
        File.WriteAllBytes(datapath + "//WSQImage.wsq", fingerprintData.WSQImage);

         ShowMessage("Capture Success.\nFinger data is saved at application path", false);
      }
else
      {
         lblStatus.Text = "Failed: error: " + errorCode.ToString() + " (" + errorMsg + ")";
      }
      lblStatus.Refresh();
    }
catch (Exception ex)
    {
      ShowMessage(ex.ToString(), true);
    }
}
```

20. Dispose()

MFS100 class is IDisposible, so it is compulsory to call this method on FormClosing event of windows form.

```
privatevoid frmMFS100_FormClosing(object sender, FormClosingEventArgs e)
{
try
    {
      mfs100.Dispose();
    }
catch
    {}
}
```

21. DeviceInfo Class

Please see class diagram.

22. FingerData Class

Please see class diagram.

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