Xamarin.Android (v2.1.x)

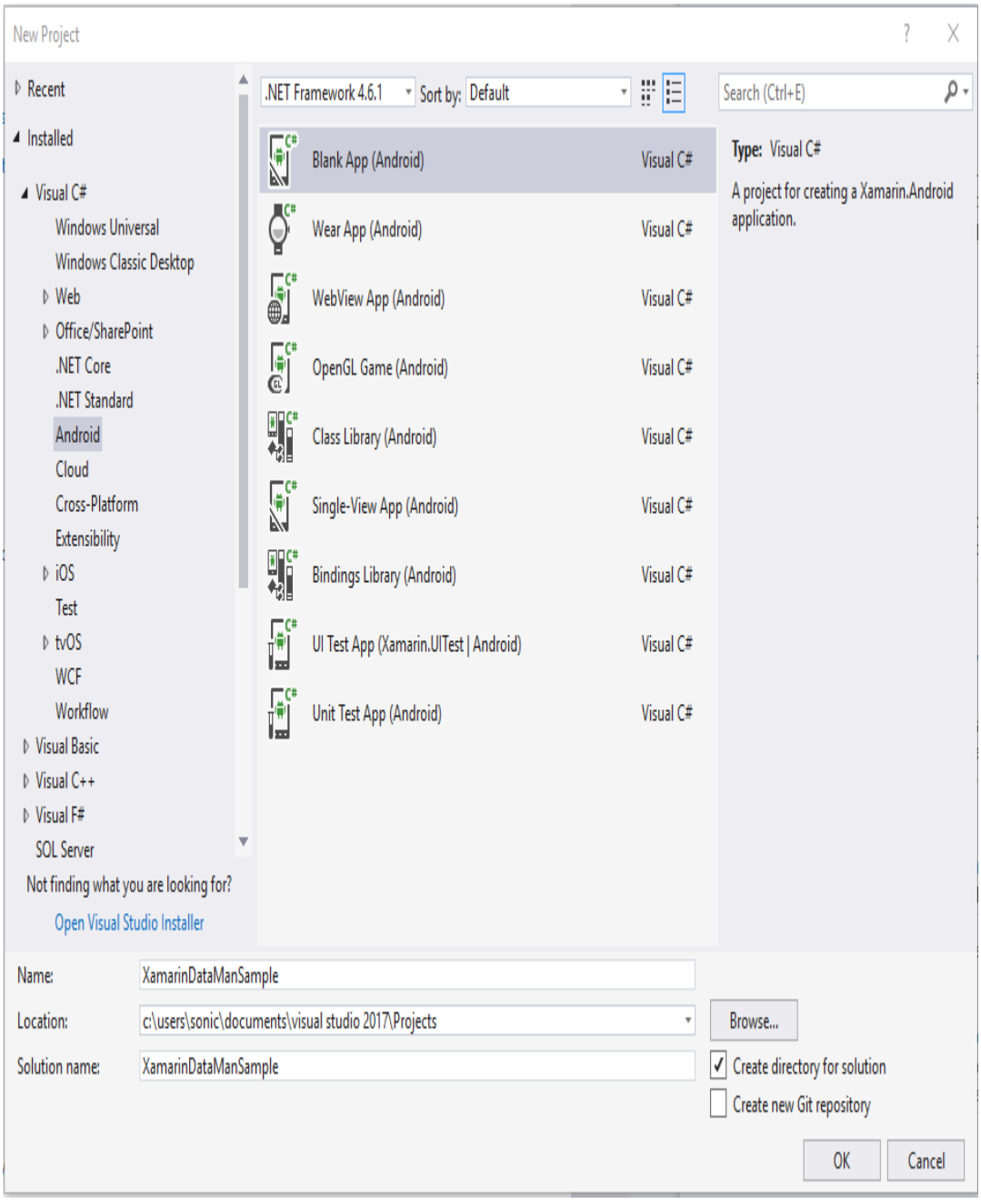
Getting Started

In the following sections we will explain how our sample app is developed step by step.

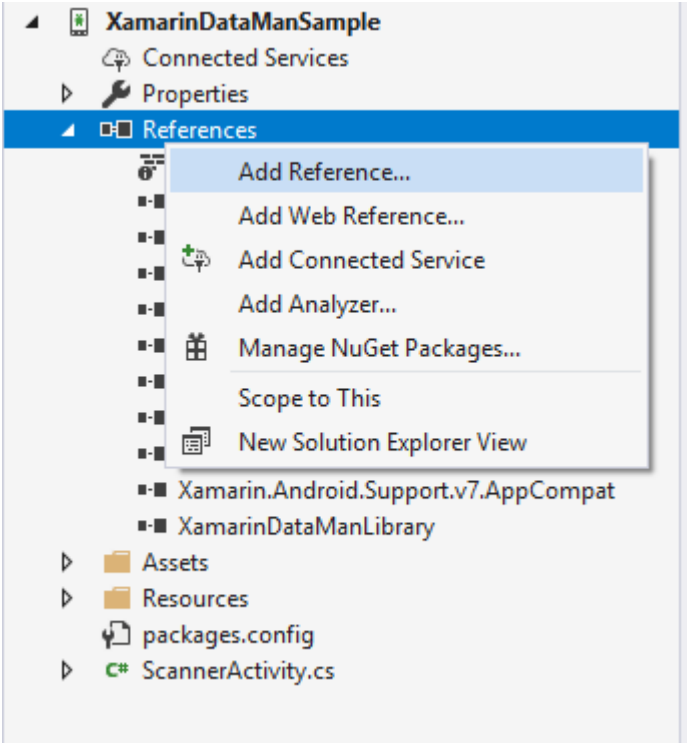
Open Visual Studio and follow these steps:

1. Go to **File -> New -> Project.**

2. Create **Blank App (Android).**



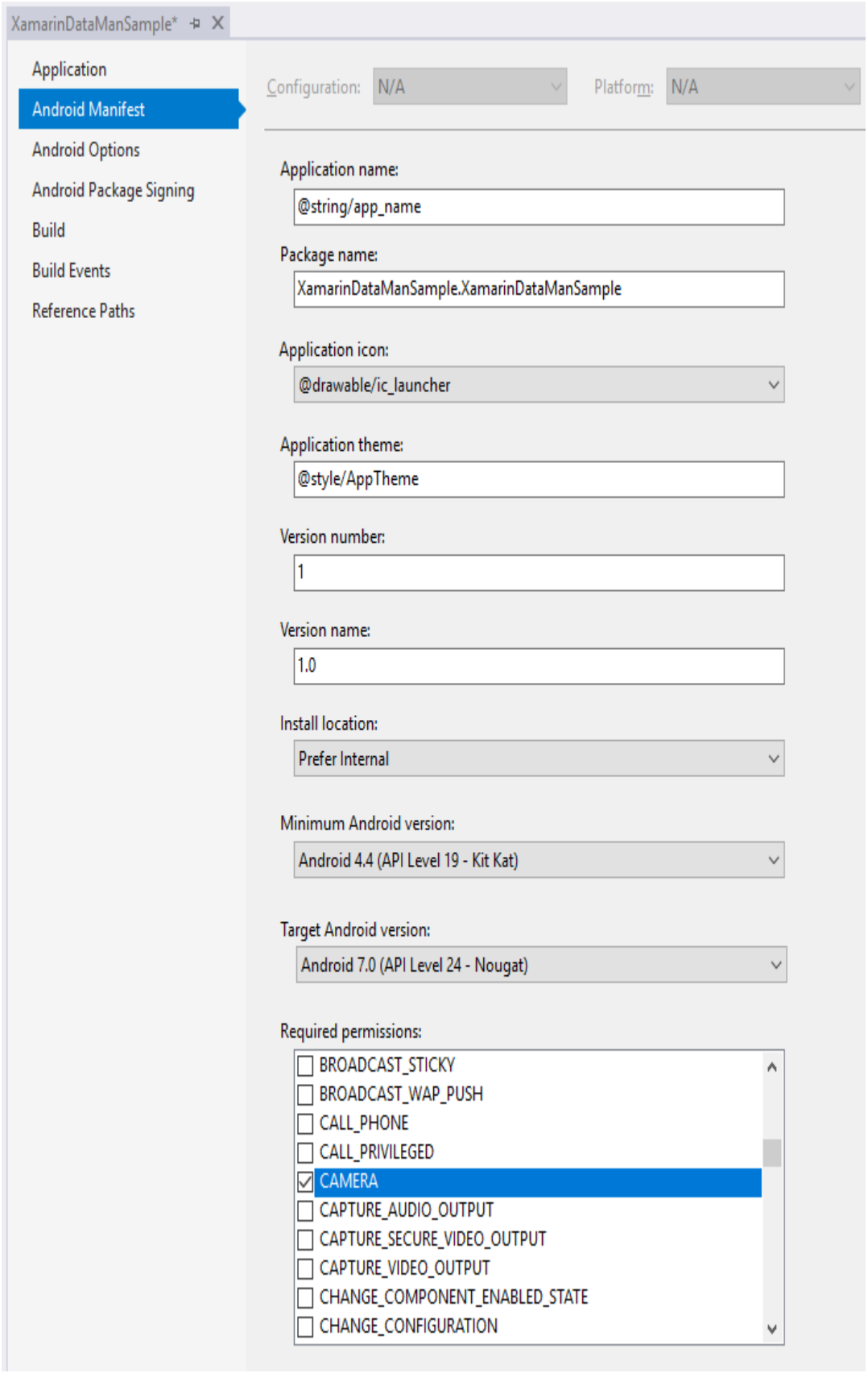
When your new project is loaded add a reference to the XamarinDataManLibrary.dll file.



After creating a blank application, create all resources you will use (icons, images, styles, layouts, etc.). You can copy them from our sample.

Next right click on your project file, then click Properties and go to Android Manifest section.

Setup your Manifest file (app name, app icon, minimum and maximum android versions), make sure to enable Camera permission for this application.



Scanner Activity

After we've set some necessary properties, we can create the **ScannerActivity** which will be the **MainLauncher** for this application, and inherit some Interfaces.

```
[Activity(Label = "@string/app_name", MainLauncher = true, Icon = "@drawable/ic_launcher", ScreenOrientation = ScreenOrient
    public class ScannerActivity : Activity, IOnConnectionCompletedListener, IReaderDeviceListener, Android.Support.V4.App.
    {
        private static int REQUEST_PERMISSION_CODE = 12322;

        private ListView listViewResult;
        private TextView tvConnectionStatus;
        private Button btnScan;
        private RelativeLayout rlPreviewContainer;
        private ImageView ivPreview;

        private List<ReadResult> resultList;
        private JavaList<IDictionary<string, object>> resultListData;
        private ResultListViewAdapter resultListAdapter;
        public static int listViewResultSelectedItem = -1;

        private bool isScanning = false;


        private static ReaderDevice readerDevice;

        private enum DeviceType { MX, PHONE_CAMERA }

        private static bool isDevicePicked = false;
        private static DeviceType param_deviceType = DeviceType.PHONE_CAMERA;

        private static bool dialogAppeared = false;

        private static string selectedDevice = "";

        private bool listeningForUSB = false;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            SetContentView(Resource.Layout.activity_scanner);

         ......
```

Here we will define some UI elements and variables that will be used later in the activity:

tvConnectionStatus - TextView an UI element for current connection status;

listViewResult - ListView an UI element for results that will be read;

btnScan - Button an UI element that will trigger ***readerDevice.StartScanning*** or ***readerDevice.StopScanning***;

rlPreviewContainer - ViewGroup (RelativeLayout) container for camera preview;

ivPreview - ImageView control with matching size as its parent for showing the last frame of a preview or scanning session;

ReaderDevice - cmbSDK object that will present MX Device or Phone Camera depends of our configuration;

Configure ReaderDevice

Every time the activity starts, we are calling the **initDevice** where we configure the reader device object.

If we want to use an MX Device for scanning we can use:

```
readerDevice = GetMXDevice(this);
if (!listeningForUSB)
{
```

```
    readerDevice.StartAvailabilityListening();
    listeningForUSB = true;
}
```

The availability of the MX Device can change when the device turns **ON** or **OFF**, or if the USB cable gets connected or **disconnected**, and this is handled by the **IReaderDeviceListener** interface.

If we want to configure the reader device as a Phone Camera we can use:

```
readerDevice = GetPhoneCameraDevice(this, CameraMode.NoAimer, PreviewOption.Defaults, rlPreviewContainer);
```

The CameraMode parameter is of type **CameraMode** (defined in **CameraMode.java**) and it accepts one of the following values:

- **NO_AIMER**: Initializes the reader to use a live-stream preview (on the mobile device screen), so that the user can position the barcode within the camera's field of view for detection and decoding. Use this mode when the mobile device does not have an aiming accessory.
- **PASSIVE_AIMER**: Initializes the reader to use a passive aimer, an accessory attached to the mobile device or mobile device case that uses the built-in LED flash of the mobile device as a light source for projecting an aiming pattern. In this mode no live-stream preview is presented on the device screen, since an aiming pattern will be projected.
- **FRONT_CAMERA**: Initializes the reader to use the mobile front facing camera of the device, if available (not all mobile devices have a front camera). This is an unusual but possible configuration. Most front facing cameras do not have auto focus and illumination, and provide significantly lower resolution images. This option should be used with care. In this mode illumination is not available.

All of the above modes provide the following default settings for the reader:

- The rear camera is used.
- The zoom feature is available and a button to control it is visible on the live-stream preview (if displayed).
- The simulated hardware trigger is disabled.
- When the ***startScanning()*** is called, the decoding process is started (See PreviewOption.PAUSED below for more details).

Based on the selected mode, the following additional options and behaviors are set:

- **NO_AIMER** (NoAimer)
- The live-stream preview is displayed when the ***startScanning()*** method is called.
- Illumination is available, and a button to control it, is visible on the live-stream preview.
- If commands are sent to the reader for aimer control, they will be ignored.

- **PASSIVE_AIMER** (Passive Aimer)
- The live-stream preview will not be displayed when the ***startScanning()*** method is called.
- Illumination is not available, and the live-stream preview will not have an illumination button.
- If commands are sent to the reader for illumination control, they will be ignored, since it is assumed in this mode that the built-in LED of the mobile device is being used as the aimer.

- **FRONT_CAMERA** (FrontCamera)
- The live-stream preview is displayed when the ***startScanning()*** method is called.
- The front camera is used.
- Illumination is not available and the live-stream preview will not have an illumination button.
- If commands are sent to the reader for aimer or illumination control, they will be ignored.

The ***PreviewOption*** parameter is a type of ***PreviewOption*** (defined in **PreviewOption.java**), and is used to change the reader's default values or override defaults derived from the selected *CameraMode*. Multiple options can be specified by OR-ing them when passing the parameter. The available options are:

- **DEFAULTS**: Option to accept all defaults set by the **CameraMode**.
- **NO_ZOOM_BUTTON**: Option to hide the zoom button on the live-stream preview, preventing a user from adjusting the mobile device camera's zoom.
- **NO_ILLUMINATION_BUTTON**: Option to hide the illumination button on the live-stream preview, preventing a user from toggling the illumination.
- **HARDWARE_TRIGGER**: Option to enable a simulated hardware trigger (the volume down button) for starting scanning on the mobile device. This button only starts scanning when pressed. It does not need to be held like a more traditional purpose-built scanner's trigger.

Pressing the button a second time does not stop the scanning process.
- **PAUSED**: If using a live-stream preview, when this option is set, the preview will be displayed when the ***startScanning()*** method is called, but the reader will not start decoding (i.e. looking for barcodes) until the user presses the on-screen scanning button to actually start the scanning process.
- **ALWAYS_SHOW**: Option to force a live-stream preview to be displayed, even if an aiming mode has been selected (e.g. CameraMode == PASSIVE_AIMER).

Connecting to Device

After configuring the **ReaderDevice** we need to connect to the device.

Before we make a connection the **ReaderDeviceListener** object is set in order to receive events:

```
readerDevice.SetReaderDeviceListener(this);
```

Additionally, you can enable sending the last triggered image and SVG from the reader by:

```
readerDevice.EnableImage(true);
readerDevice.EnableImageGraphics(true);
```

Then we can connect with:

```
readerDevice.Connect(this);
```

Events that will be invoked are:

```
public void OnConnectionStateChanged(ReaderDevice reader)
public void OnConnectionCompleted(ReaderDevice reader, Throwable error)
```

If there is an error while trying to connect, the error will be thrown as a parameter in the **OnConnectionCompleted** method, otherwise, if no error occurs, the error parameter will be **null**.

If the connection is successful, the statement **reader.ConnectionState == ConnectionState.Connected** will be true.

There are couple of API methods for changing some public properties for configuring the connected device and you should invoke them when the **ConnectionState** is connected.

For example if Mobile Camera is used as a **ReaderDevice** there are **no symbologies enabled by default**. You must enable the symbologies that you want to use with the **SetSymbologyEnabled** API method:

```
readerDevice.SetSymbologyEnabled(Symbology.C128, true, null);
readerDevice.SetSymbologyEnabled(Symbology.Datamatrix, true, null);
readerDevice.SetSymbologyEnabled(Symbology.UpcEan, true, null);
readerDevice.SetSymbologyEnabled(Symbology.Qr, true, null);
```

You can do the same directly by sending a command to the connected device with:

```
readerDevice.DataManSystem.SendCommand("SET SYMBOL.MICROPDF417 ON");
```

Scanning Barcodes

With a properly configured reader, you are now ready to scan barcodes. This can be done by calling the ***startScanning*** method from your **ReaderDevice** object.

What happens next is based on the type of **ReaderDevice** and how it has been configured, but in general:

- If using an MX Device, the user can press a trigger button on the device to turn the scanner on and read a barcode;
- If using the camera reader, the **cmbSDK** starts the camera, displays the configured live-stream preview, and begins analyzing the frames from the video stream, looking for a configured barcode symbology;

Scanning stops under one of the following conditions:

- The reader found and decoded a barcode;
- The user released the trigger or pressed the stop button on the live-stream preview screen;
- The camera reader timed out without finding a barcode;
- The application itself calls the ***stopScanning()*** method.

When a barcode is decoded successfully (the first case), you will receive a **ReadResults** iterable result collection object in the **ReaderDevice** listener method.

If your MX Device is configured to work with multi code scanning, you can access all the scanned results from the **results.SubResults** property which is an array that contains **ReaderResult** objects and it will be **null** if single code scanning is used.

Example

```
public void OnReadResultReceived(ReaderDevice reader, ReadResults results)
        {

            listViewResultSelectedItem = -1;
            resultList.Clear();
            resultListData.Clear();
            ivPreview.SetImageBitmap(null);

            if (results.SubResults != null && results.SubResults.Count > 0)
            {
                foreach (ReadResult subResult in results.SubResults)
                {
                    if (subResult.IsGoodRead)
                    {
                        resultList.Add(subResult);

                        JavaDictionary<string, object> item = new JavaDictionary<string, object>();
                        item.Add("resultText", subResult.ReadString);

                        Symbology sym = subResult.Symbology;
                        if (sym != null)
                            item.Add("resultType", subResult.Symbology.Name);
                        else
                            item.Add("resultType", "UNKNOWN SYMBOLOGY");

                        resultListData.Add(item);
                        listViewResultSelectedItem = resultListData.Count - 1;
                    }
                    else
                    {
                        resultList.Add(subResult);

                        JavaDictionary<string, object> item = new JavaDictionary<string, object>();
```

```
                    item.Add("resultText", "NO READ");
                    item.Add("resultType", "");

                    resultListData.Add(item);
                    listViewResultSelectedItem = resultListData.Count - 1;
                }

                if (subResult.Image != null)
                {
                    ivPreview.SetImageBitmap(renderSvg(subResult.ImageGraphics, subResult.Image));
                }
                else
                {
                    if (subResult.ImageGraphics != null)
                    {
                        ivPreview.SetImageBitmap(renderSvg(subResult.ImageGraphics, ivPreview.Width, ivPreview.Height))
                    }
                    else
                        ivPreview.SetImageBitmap(null);
                }
            }
        }
        else if (results.Count > 0)
        {
            ReadResult result = results.GetResultAt(0);

            if (result.IsGoodRead)
            {
                resultList.Add(result);

                JavaDictionary<string, object> item = new JavaDictionary<string, object>();
                item.Add("resultText", result.ReadString);

                Symbology sym = result.Symbology;
                if (sym != null)
                    item.Add("resultType", result.Symbology.Name);
                else
                    item.Add("resultType", "UNKNOWN SYMBOLOGY");

                resultListData.Add(item);
                listViewResultSelectedItem = resultListData.Count - 1;
            }
            else
            {
                resultList.Add(result);

                JavaDictionary<string, object> item = new JavaDictionary<string, object>();
                item.Add("resultText", "NO READ");
                item.Add("resultType", "");

                resultListData.Add(item);
                listViewResultSelectedItem = resultListData.Count - 1;
            }

            if (result.Image != null)
            {
                ivPreview.SetImageBitmap(renderSvg(result.ImageGraphics, result.Image));
            }
            else
            {
                if (result.ImageGraphics != null)
                {
                    ivPreview.SetImageBitmap(renderSvg(result.ImageGraphics, ivPreview.Width, ivPreview.Height));
                }
                else
                    ivPreview.SetImageBitmap(null);
            }
        }

        isScanning = false;
        btnScan.Text = "START SCANNING";
        resultListAdapter.NotifyDataSetChanged();
    }
```

result.Image is the last frame from the scanning process and it will be displayed in the **ivPreview ImageView,** and **result.ImageGraphics** is SVG image that locate barcode on image.

Disconnecting from Device

In the **ScannerActivity** we override the **OnPause** and the **OnStop** events so we can do the **Disconnect** and the ***StopAvailabilityListening*** to release all connection when we navigate from or destroy that activity.

```
protected override void OnPause()
{
      base.OnPause();

      if (readerDevice != null)
      {
          readerDevice.Disconnect();
      }
}


protected override void OnStop()
{
      if (readerDevice != null)
        try
        {
             readerDevice.StopAvailabilityListening();
        }
        catch (System.Exception e) { }

      listeningForUSB = false;

      base.OnStop();
}
```

Keep in mind there might be cases when a device disconnects due to low battery condition or manual cable disconnection.

Licensing the SDK

If you plan to use the **cmbSDK** to do mobile scanning with a smartphone or a tablet (without the MX mobile terminal), the SDK requires the installation of a license key. Without a license key, the SDK will still operate, although scanned results will be blurred (the SDK will randomly replace characters in the scan result with an asterisk character).

Contact your Cognex Sales Representative for information on how to obtain a license key including trial licenses which can be used for 30 days to evaluate the SDK.

After obtaining your license key there are two ways to add your license key in an application.

The first one is to add it as a meta tag in application tag in your manifest file:

```
<application android:label="XamarinDataManSample">
        .....

  <meta-data android:name="MX_MOBILE_LICENSE" android:value="YOUR_MX_MOBILE_LICENSE"/>
</application>
```

Second way is to implement the activation directly from code. When you create your readerDevice set license key as input parameter in constructor:

```
...
readerDevice = GetPhoneCameraDevice(this, CameraMode.NoAimer, PreviewOption.Defaults, rlPreviewContainer, "YOUR_MX_MOBILE_L
```

Api Methods

Beep()

To add sound "beep" after successful scan please use the code below:

```
readerDevice.Beep()
```

It plays an audio signal on the MX device.

Connect()/Disconnect()

The process of connecting and disconnecting to a MX device is done via the Connect/Disconnect methods

Connect()

```
readerDevice.Connect(IOnConnectionCompletedListener listenerObject)
```

will try to connect to the device. When the **Connect()** is executed we get the status of the action in the **OnConnectionCompleted** listener

```
OnConnectionCompleted(ReaderDevice reader,Throwable error){
 if(error != null){
    //do something with the error
    readerDisconnected();
 }
}
```

Disconnect()

```
readerDevice.Disconnect()
```

will disconnect from the MX device.

Examples:

```
readerDevice = ReaderDevice.GetMXDevice(mContext);
readerDevice.StartAvailabilityListening();
readerDevice.SetReaderDeviceListener(this);
readerDevice.EnableImage(true);
readerDevice.Connect(this);

...
```

```
public void OnAvailabilityChanged(ReaderDevice reader)
{
    if (reader.GetAvailability() == Availability.Available)
    {
          readerDevice.Connect(this);
```

```
    }
 ...
```

EnableImage()

The result from a successful scan can return an image. This is the last frame that resolved in a successful scan. To enable / disable this, we can use the API method:

```
void readerDevice.EnableImage(bool enable)
```

```
readerDevice = ReaderDevice.GetPhoneCameraDevice(mContext, CameraMode.NoAimer, PreviewOption.Defaults, rlMainContainer);
readerDevice.EnableImage(true);
readerDevice.Connect(this);
```

EnableImageGraphics()

The result from a successful scan can return an SVG image graphics. This is the last frame that resolved in a successful scan. To enable / disable this, we can use the API method:

```
void readerDevice.EnableImageGraphics(bool enable)
```

```
readerDevice = ReaderDevice.GetPhoneCameraDevice(mContext, CameraMode.NoAimer, PreviewOption.Defaults, rlMainContainer);
readerDevice.EnableImageGraphics(true);
readerDevice.Connect(this);
```

GetAvailability()

Before we can connect to an MX device, we need to know if there's one available for that task.

```
Availability reader.GetAvailability()
```

It return **Availability** object that can be **Available, Unavailable** or **Unknown**

An MX device is available when there is an USB connection to our smartphone.

Example

```
public void OnAvailabilityChanged(ReaderDevice reader)
{
   if (reader.GetAvailability() == Availability.Available)
   {
       readerDevice.Connect(this);
   }
    ...
```

GetDeviceBatteryLevel()

If we want to check the battery level of the MX device we can use

```
void readerDevice.GetDeviceBatteryLevel(IOnDeviceBatteryLevelListener listenerObject)
```

Retrieves the current battery percentage level of the reader device as input parameter in **OnDeviceBatteryLevelReceived** listener method

Example

```
public void OnConnectionStateChanged(ReaderDevice reader)
{
   if (reader.ConnectionState == ConnectionState.Connected)
   {
        reader.GetDeviceBatteryLevel(this);

        .........
    }
}


public void OnDeviceBatteryLevelReceived(ReaderDevice p0, int p1, Throwable p2)
{
    int batteryLevel = p1;
}
```

IsLightsOn()/SetLightsOn()

If we want to check whether all lights of the MX device are turned on or off

```
void readerDevice.IsLightsOn(IOnLightsListener listenerObject)
```

Retrieves if lights of the reader device are turned on or off as input parameter in **OnLightsOnCompleted** listener method

To turned MX device light on or off we use

```
void reader.SetLightsOn(bool _enable, IOnLightsListener listenerObject)
```

Example

```
public void OnConnectionStateChanged(ReaderDevice reader)
{
   if (reader.ConnectionState == ConnectionState.Connected)
   {
        reader.IsLightsOn(this);
        .........
    }
}
public void OnLightsOnCompleted(ReaderDevice p0, Java.Lang.Boolean p1, Throwable p2)
{
   bool lightsON = p1.BooleanValue();
}
```

IsSymbologyEnabled()/SetSymbologyEnabled()

If we want to check some symbology is enabled or disabled we can use

```
void readerDevice.IsSymbologyEnabled(Symbology _symbology, IOnSymbologyListener listenerObject)
```

Retrieves if symbology is enabled or disabled as input parameter in **OnSymbologyEnabled** listener method

To enable specific symbology we use

```
void readerDevice.SetSymbologyEnabled(Symbology _symbology, bool _enable, IOnSymbologyListener listenerObject)
```

Example

```
public void OnConnectionStateChanged(ReaderDevice reader)
{
   if (reader.ConnectionState == ConnectionState.Connected)
   {
        reader.IsSymbologyEnabled(Symbology.Azteccode, this);
        .........
    }
}
public void OnSymbologyEnabled(ReaderDevice p0, Symbology p1, Java.Lang.Boolean p2, Throwable p3)

}
```

ResetConfig()

To reset MX Device configuration settings to default use

```
void reader.ResetConfig(IOnResetConfigListener listenerObject)
```

In **OnResetConfigCompleted** listener we can check if resetting was successful by checking if Throwable input parameter error is null or not

```
public void OnResetConfigCompleted(ReaderDevice p0, Throwable p1)
{
      if (p1 != null)
         Console.WriteLine("Resetting was unsuccessful. Error: " + p1.Message);
}
```

SetReaderDeviceListener()

To register listener functions **OnAvailabilityChanged**, **OnConnectionStateChanged** and **OnReadResultReceived** we need to call this method before we try to connect to reader device

```
void readerDevice.SetReaderDeviceListener(IReaderDeviceListener listenerObject)
```

```
public void OnAvailabilityChanged(ReaderDevice reader)
{
```

```
   if (reader.GetAvailability() == Availability.Available)
   {
      readerDevice.Connect(this);
   }
}
```

```
public void OnConnectionStateChanged(ReaderDevice reader)
{
    if (reader.ConnectionState == ConnectionState.Connected)
    {
    }
    else if (reader.ConnectionState == ConnectionState.Disconnected)
    {
    }
}
```

```
public void OnReadResultReceived(ReaderDevice reader, ReadResults results)
{
   if (results.Count > 0)
   {
        ReadResult result = results.GetResultAt(0);

        if (result.IsGoodRead)
        {
             ...
```

Example

```
readerDevice = ReaderDevice.GetPhoneCameraDevice(mContext, CameraMode.NoAimer, PreviewOption.Defaults, rlMainContainer);
readerDevice.SetReaderDeviceListener(this);
readerDevice.Connect(this);
```

StartAvailabilityListening()/StopAvailabilityListening()

After we call **readerDevice.SetReaderDeviceListener(IReaderDeviceListener listenerObject)** method and register **OnAvailabilityChanged** listener function we can start/stop availability listening

StartAvailabilityListening()

void readerDevice.StartAvailabilityListening()

will start listening reader device availability and will trigger listener function every time when availability is changed

StopAvailabilityListening()

void readerDevice.StopAvailabilityListening()

will stop listening reader device availability

Examples:

```
readerDevice = ReaderDevice.GetMXDevice(mContext);
readerDevice.StartAvailabilityListening();
```

```
readerDevice.SetReaderDeviceListener(this);
readerDevice.Connect(this);
```

```
protected override void Dispose(bool disposing)
{
     if (readerDevice != null && readerDevice.ConnectionState == ConnectionState.Connected)
     {
          readerDevice.SetReaderDeviceListener(null);
          readerDevice.Disconnect();
     }

     if (readerDevice != null)
     try
     {
         readerDevice.StopAvailabilityListening();
     }
     catch (System.Exception e) { }

   base.Dispose(disposing);
}
```

StartScanning()/StopScanning()

After we connect and configure our reader device and set all settings that we need we can start scanning process.

To start scanning we use method

```
void readerDevice.StartScanning()
```

Scanning process will stop when our reader successfully scan some barcode or we can stop scanning manually with this method

```
void readerDevice.StopScanning()
```

When scanning is stopped, no matter with successful scan or with stopScanning() method **OnReadResultReceived** listener function will be called where we can check our scan result

```
public void OnReadResultReceived(ReaderDevice reader, ReadResults results)
{
     if (results.Count > 0)
     {
        ReadResult result = results.GetResultAt(0);

      if (result.IsGoodRead)
      {
          Symbology sym = result.Symbology;
          if (sym != null)
          {
               tvSymbology.Text = sym.Name;
          }
          else
          {
              tvSymbology.Text = "UNKNOWN SYMBOLOGY";
          }
          tvCode.Text = result.ReadString;
      }
     else
     {
         tvSymbology.Text = "NO READ";
         tvCode.Text = "";
      }
      ivPreview.SetImageBitmap(result.Image);
     }
}
```