

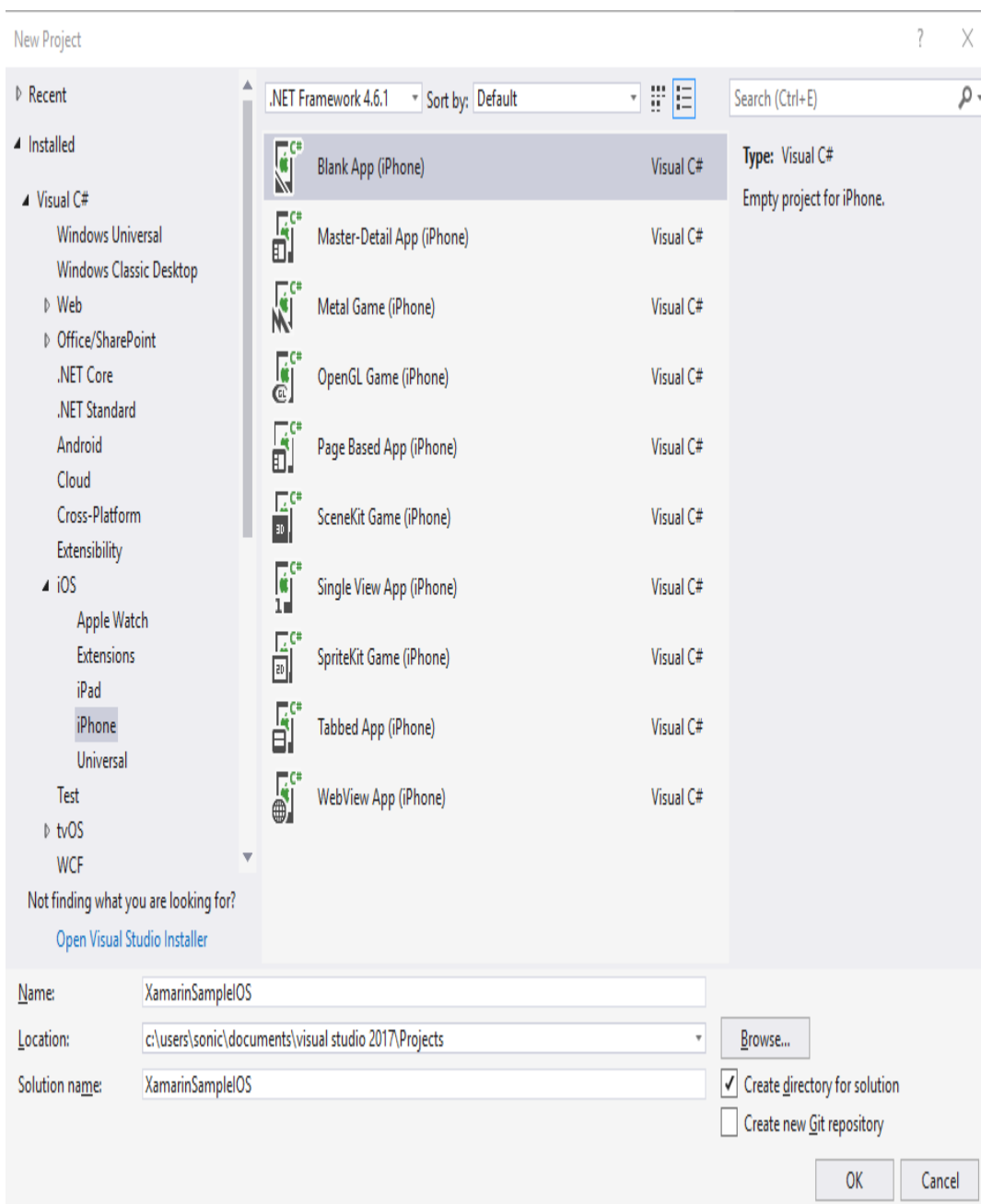
Xamarin.iOS (v2.1.x)

Getting Started

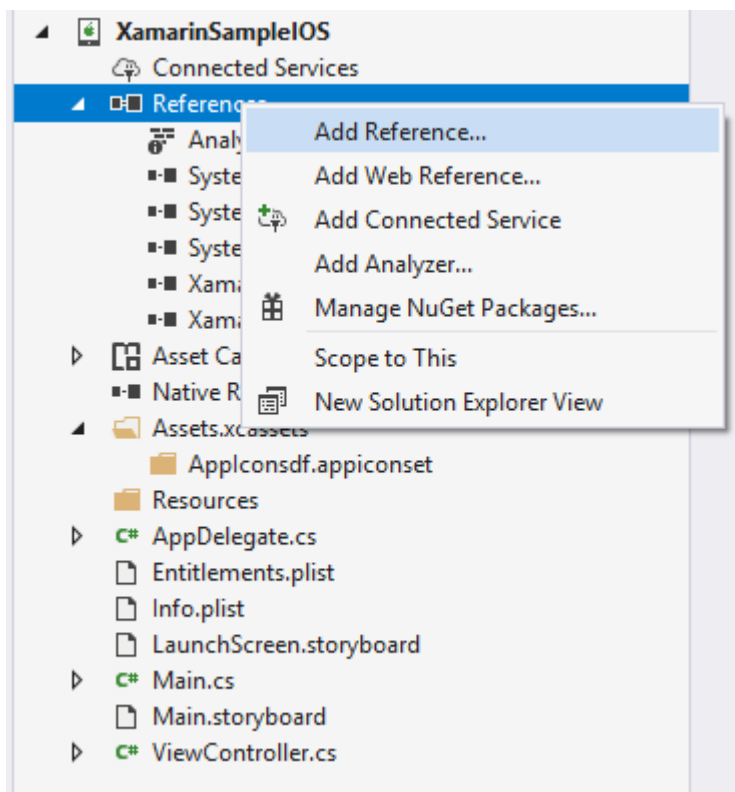
In the following sections we will explain how our sample app is developed step by step.

Open Visual Studio and follow this steps:

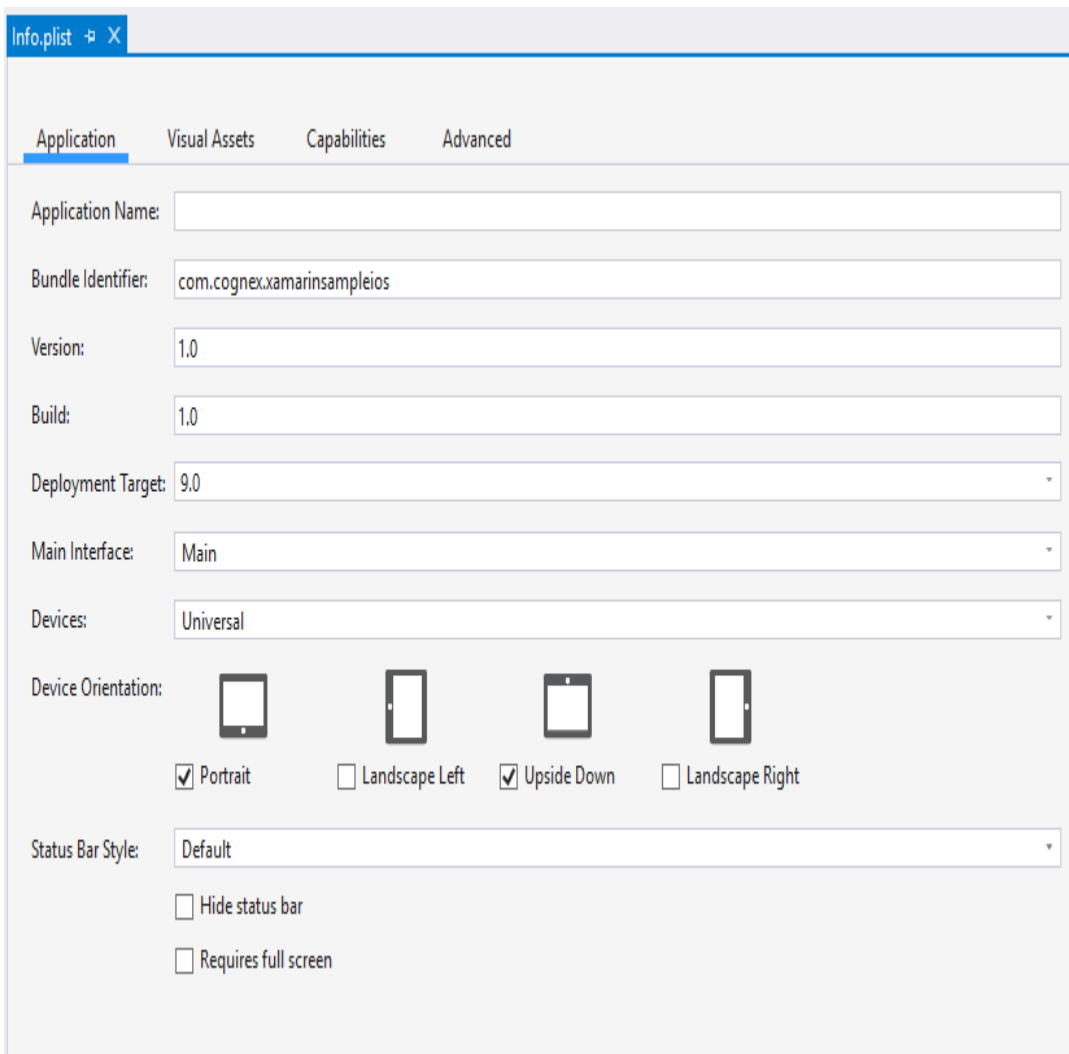
1. Go to **File -> New -> Project**.
2. Create **Blank App (iPhone)**.



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



Next open your **Info.plist** file and set some project properties for your needs (app name, deployment target, main interface, etc..).



Info.plist

Application Visual Assets Capabilities Advanced

Application Name:

Bundle Identifier:

Version:

Build:

Deployment Target:

Main Interface:

Devices:

Device Orientation:

☒ Portrait ☐ Landscape Left ☒ Upside Down ☐ Landscape Right

Status Bar Style:

☐ Hide status bar

☐ Requires full screen

Important thing here is to add Camera permission for this app. In Visual Studio there is no options to add this permission from here. You need to open your **Info.plist** file in some text editor and add this lines:

```
<key>NSCameraUsageDescription</key>
<string>Camera used for scanning</string>
```

Also if you use MX Device as reader device add this protocols in Info.plist:

```
<key>UISupportedExternalAccessoryProtocols</key>
<array>
  <string>com.cognex.dmcc</string>
  <string>com.demo.data</string>
</array>
```

View Controller

The **ViewController** will be our first controller in **Main.storyboard**. Here we are creating some UI elements and variables that will be used later in this controller.

```
public partial class ViewController : UIViewController, ICMBReaderDeviceDelegate
{
    protected ViewController(IntPtr handle) : base(handle)
    {
        // Note: this .ctor should not contain any initialization logic.
    }

    CMBReaderDevice readerDevice;
    public bool isScanning = false;

    private NSMutableArray tableData;
    private MXResultsTableSource tableSource;

    public override void ViewDidLoad()
    {
        base.ViewDidLoad();

        ...
    }
}
```

lblConnection - Label UI element for current connection status.

tableSource - UITableViewSource for results that will be read.

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

ivPreview - **ImageView** UI element for showing the last frame of a preview or scanning session.

CMBReaderDevice - cmbSDK object that will present MX Device or Phone Camera depends of our configuration.

Configure ReaderDevice

Here we override the **ViewWillAppear** method to configure reader device object when this view will appear.

If we want to use MX Device for scanning we are using

```
readerDevice = CMBReaderDevice.ReaderOfMXDevice();
```

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 is handled by the **ICMBReaderDeviceDelegate** interface. We set this interface as property for reader device with

```
readerDevice.WeakDelegate = this;
```

and allow us to listen these three events:

```
public void DidReceiveReadResultFromReader(CMBReaderDevice reader, CMBReadResults readResults)
public void AvailabilityDidChangeOfReader(CMBReaderDevice reader)
public void ConnectionStateDidChangeOfReader(CMBReaderDevice reader)
```

If we want to configure reader device as Mobile Camera.

```
readerDevice = CMBReaderDevice.ReaderOfDeviceCameraWithCameraMode(CDMCameraMode.NoAimer, CDMPreviewOption.Defaults, ivPreview)
```

The **CameraMode** parameter is of the type **CDMCameraMode**, and it accepts one of the following values:

- **NoAimer**: Initializes the reader to use a live-stream preview (on the mobile device screen) so 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.

- **PassiveAimer:** Initializes the reader to use a passive aimer, which is an accessory that is 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.
- **FrontCamera:** Initializes the reader to use the front facing camera of the mobile 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 **startScanning()** is called, the decoding process is started. (Seek **CDMPreviewOptionPaused** for more details.)

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

- **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.
- **PassiveAimer**
 - 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 for the aimer.
- **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 **previewOptions** parameter (of type **CDMPreviewOption**) 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 the following:

- **Defaults:** Use this option to accept all defaults set by the **CameraMode**.
- **NoZoomBtn:** This option hides the zoom button on the live-stream preview, preventing a user from adjusting the zoom of the mobile device camera.
- **NoIllumBtn:** This hides the illumination button on the live-stream preview, preventing a user from toggling the illumination.
- **HwTrigger:** This enables 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.
- **AlwaysShow:** This forces live-stream preview to be displayed, even if an aiming mode has been selected (e.g. **CameraMode == PassiveAimer**)

The last parameter of the type **UIView** is optional and is used as a container for the camera preview. If the parameter is left nil, a full screen preview will be used.

Connecting to Device

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

```
readerDevice.ConnectWithCompletion((error) => {
    if (error != null)
    {
        new UIAlertView("Failed to connect", error.Description, null, "OK", null).Show();
    }
})
```

```
}  
});
```

If there is some error while trying to connect error will be thrown as parameter in callback function. If everything is fine error parameter will be null.

This function will trigger **ConnectionStateDidChangeOfReader** method. If connection is successful **reader.ConnectionState == ConnectionState.Connected**.

After successful connection we can set some settings for ReaderDevice. ReaderDevice settings can be set with already wrapped functions or directly with sending commands to the configured device.

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 **SetSymbology** wrapped function.

In this example we are enable some symbologies and set setting to get the last frame from scanning in **ivPreview ImageView**.

```
readerDevice.SetSymbology(CMBSymbology.DataMatrix, true, (error) =>  
{  
    if (error != null)  
    {  
        System.Diagnostics.Debug.WriteLine("FALIED TO ENABLE [DataMatrix], ", error.LocalizedDescription);  
    }  
});  
readerDevice.SetSymbology(CMBSymbology.Qr, true, (error) =>  
{  
    if (error != null)  
    {  
        System.Diagnostics.Debug.WriteLine("FALIED TO ENABLE [Qr], ", error.LocalizedDescription);  
    }  
});  
readerDevice.SetSymbology(CMBSymbology.C128, true, (error) =>  
{  
    if (error != null)  
    {  
        System.Diagnostics.Debug.WriteLine("FALIED TO ENABLE [C128], ", error.LocalizedDescription);  
    }  
});  
readerDevice.SetSymbology(CMBSymbology.UpcEan, true, (error) =>  
{  
    if (error != null)  
    {  
        System.Diagnostics.Debug.WriteLine("FALIED TO ENABLE [UpcEan], ", error.LocalizedDescription);  
    }  
});  
  
readerDevice.ImageResultEnabled = true;  
readerDevice.SVGResultEnabled = true;  
readerDevice.DataManSystem.SendCommand("SET IMAGE.SIZE 0");
```

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 Reader Device and how it has been configured, but in general:

- If using an MX Device, the user can now 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 with out 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 **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 DidReceiveReadResultFromReader(CMBReaderDevice reader, CMBReadResults readResults)
{
    btnScan.SetTitle("START SCANNING", UIControlState.Normal);
    isScanning = false;

    tableData.RemoveAllObjects();

    if (readResults.SubReadResults != null && readResults.SubReadResults.Length > 0)
    {
        tableData.AddObjects(readResults.SubReadResults);
        tvResults.ReloadData();
    }
    else if (readResults.ReadResults.Length > 0) {
        tableData.Add(readResults.ReadResults[0]);
    }

    tableSource.SetItems(tableData);
    tableSource.displayResult(0);

    tvResults.ReloadData();
    tvResults.SelectRow(NSIndexPath.FromRowSection(0,0), false, UITableViewScrollPosition.None);
}
```

Disconnecting from Device

There may be cases when a device disconnects due to low battery condition or manual cable disconnection. These cases can be detected by the **ConnectionStateDidChangeOfReader** callback of the **ICMBReaderDeviceDelegate**.

Note: The **AvailabilityDidChangeOfReader** method is also called when the device becomes physically unavailable. It means that the (re)connection is not possible. Always check the availability property of the **ReaderDevice** object before trying to call the **ConnectWithCompletion** method.

Licensing the SDK

If you plan to use the cmbSDK to do mobile scanning with a smartphone or a tablet (with no MX mobile terminal), then the SDK requires the installation of a license key. Without a license key, the SDK will still operate, although scanned results will be obfuscated (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 is two ways to add your license key in application.

The first one is to add it as a key with a value in the project specific **Info.plist** file:

```
<key>MX_MOBILE_LICENSE</key>  
<string>Your license key</string>
```

And the second way to implement an activation is directly from the code when you create your readerDevice:

```
....  
readerDevice = CMBReaderDevice.ReaderOfDeviceCameraWithCameraMode(CDMCameraMode.NoAimer, CDMPreviewOption.Defaults, ivPrevi
```