.NET MAUI (v2.7.x)

Introduction

.NET Multi-platform App UI (.NET MAUI) is a cross-platform framework for creating native mobile and desktop apps with C# and XAML. Using .NET MAUI, you can develop apps that can run on Android and iOS from a single shared code-base.

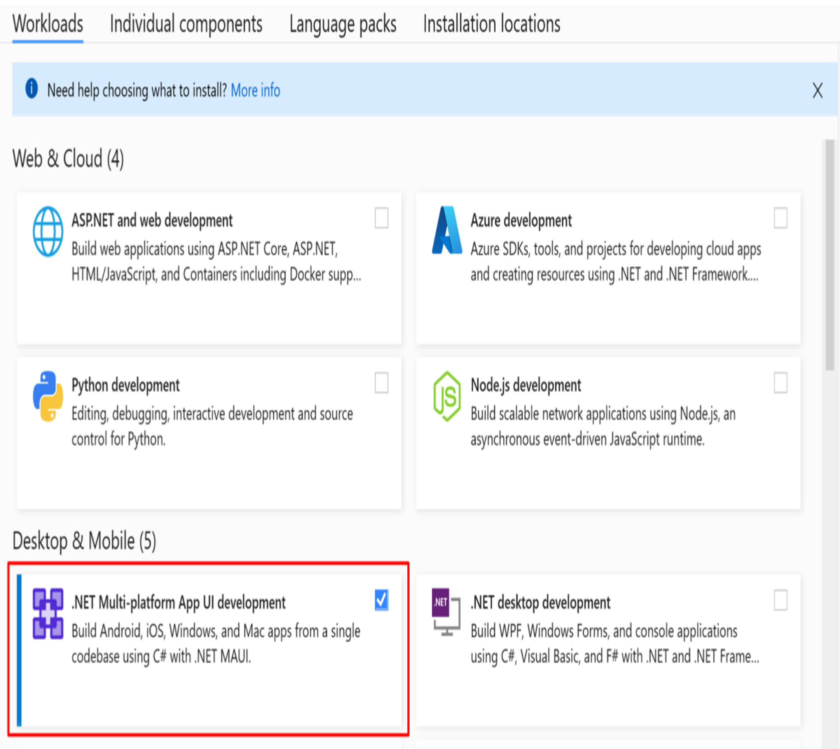
.NET MAUI is open-source and is the evolution of Xamarin.Forms, extended from mobile to desktop scenarios, with UI controls rebuilt from the ground up for performance and extensibility. If you've previously used Xamarin.Forms to build cross-platform user interfaces, you'll notice many similarities with .NET MAUI. However, there are also some differences. Using .NET MAUI, you can create multi-platform apps using a single project, but you can add platform-specific source code and resources if necessary. One of the key aims of .NET MAUI is to enable you to implement as much of your app logic and UI layout as possible in a single code-base.

Here you can read more about MAUI and how it works.

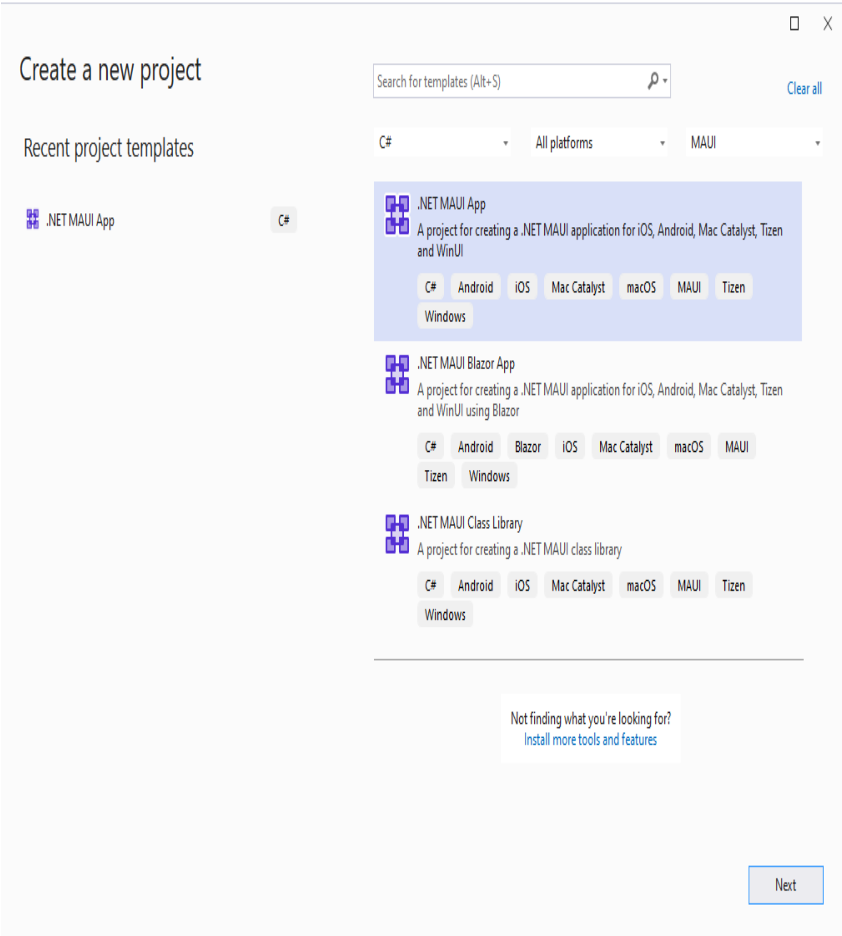
Getting Started

1. To create .NET MAUI apps, you'll need at least Visual Studio 2022 17.3 Preview or later:

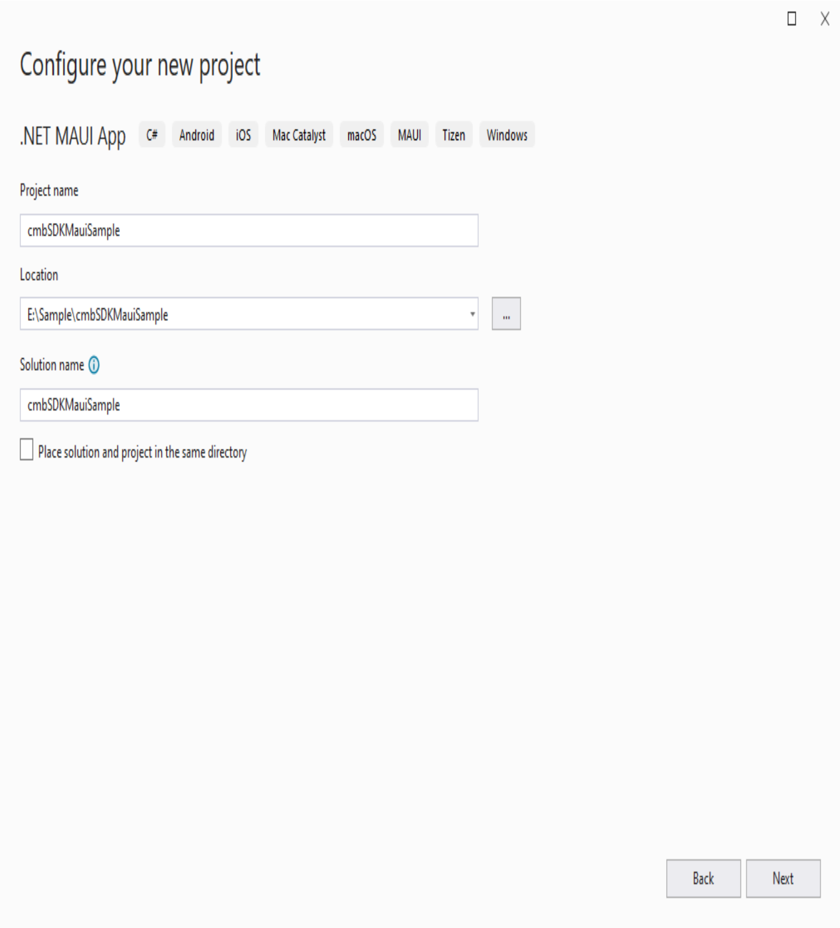
Either install Visual Studio, or modify your installation, and install the .NET Multi-platform App UI development workload with its default optional installation options:



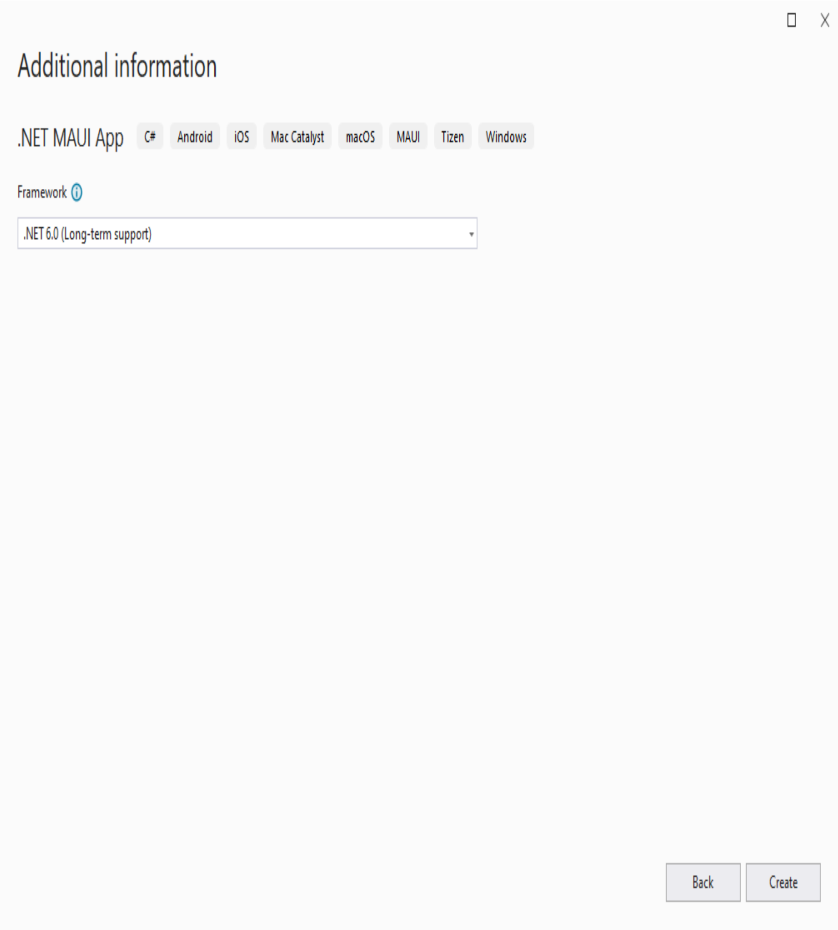
2. Launch Visual Studio, and in the start window click **Create a new project** to create a new project. In the **Create a new project** window, select **MAUI** in the **All project types** drop-down, select the **.NET MAUI App** template, and click the **Next** button:



3. In the **Configure your new project** window, name your project, choose a suitable location for it, and click the **Next** button:

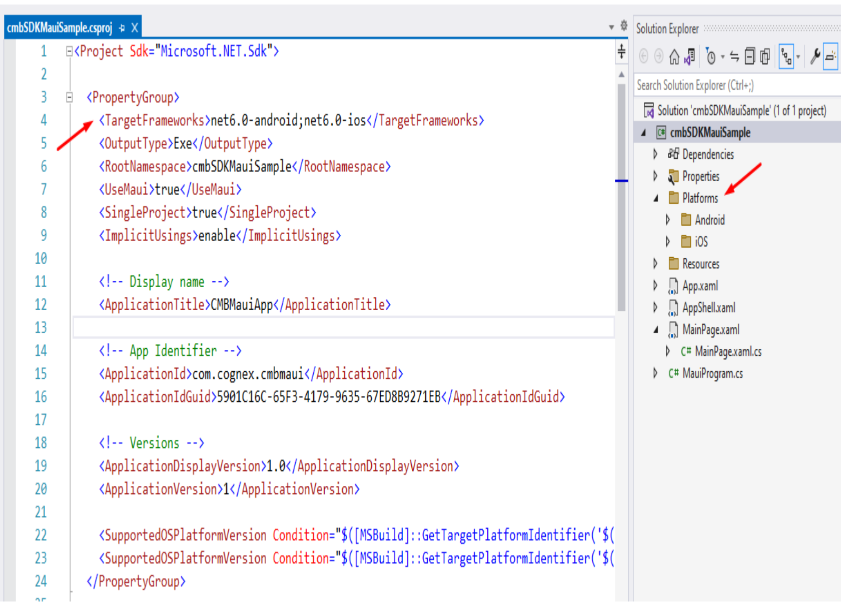


4. In the **Additional information** window, click the **Create** button:



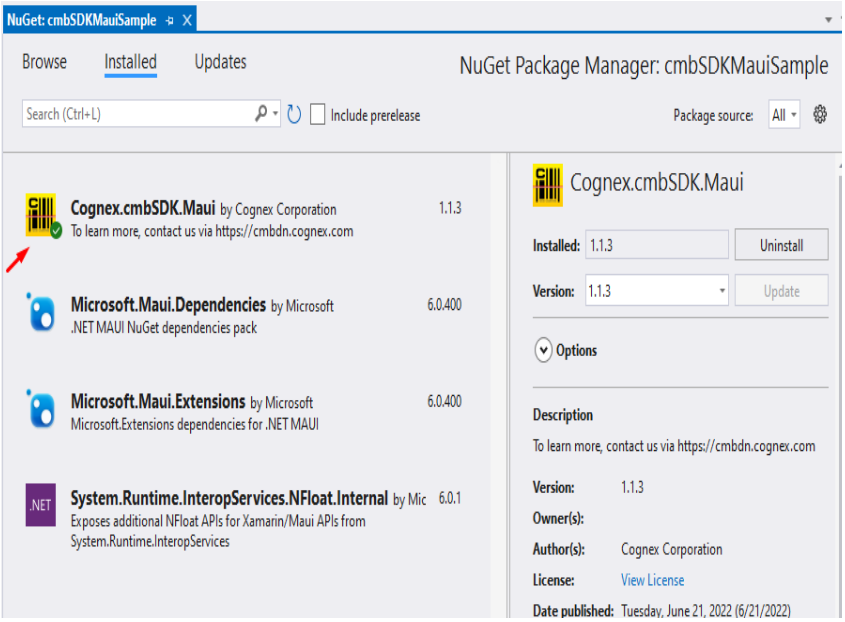
Wait for the project to be created, and its dependencies to be restored.

5. For now our SDK support only Android and iOS platforms. That's why you will need to remove the other platforms from the project (maccatalyst and windows). You should remove them from the TargetFrameworks in the **cmbSDKMauiSample.csproj** and from the project files in Platforms:

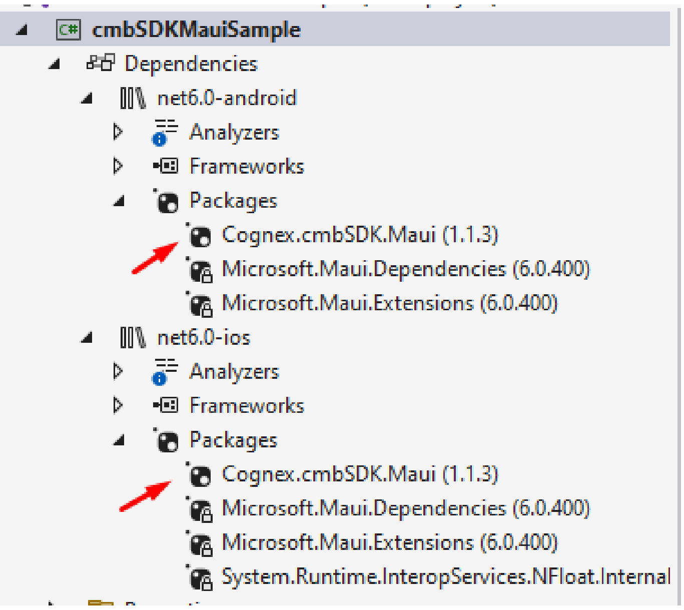


Reference cmbSDK

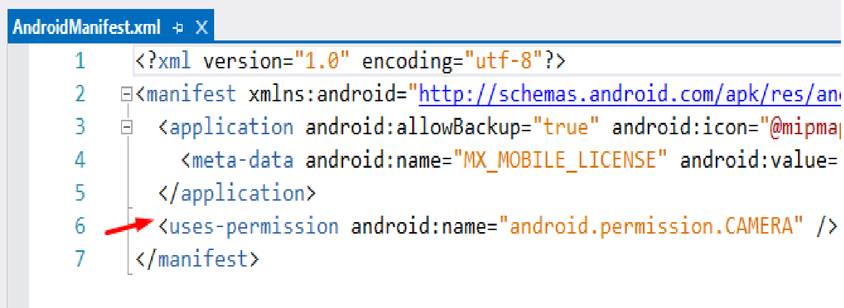
We published our SDK as nuget package on https://www.nuget.org. Please install Cognex.cmbSDK.Maui nuget package for this project.



If everything is fine Cognex.cmbSDK.Maui should be listed in Packages for both platforms:



In order to use the built-in camera of the mobile device you need to declare the camera permission.
For android set in AndroidManifest.xml file and for iOS in Info.plist (you need to open this file with text editor):



Info.plist

```xml
<key>UISupportedInterfaceOrientations</key>
<array>
  <string>UIInterfaceOrientationPortrait</string>
  <string>UIInterfaceOrientationLandscapeLeft</string>
  <string>UIInterfaceOrientationLandscapeRight</string>
</array>
<key>UISupportedInterfaceOrientations~ipad</key>
<array>
  <string>UIInterfaceOrientationPortrait</string>
  <string>UIInterfaceOrientationPortraitUpsideDown</string>
  <string>UIInterfaceOrientationLandscapeLeft</string>
  <string>UIInterfaceOrientationLandscapeRight</string>
</array>
<key>XSAppIconAssets</key>
<string>Assets.xcassets/appicon.appiconset</string>
<key>NSCameraUsageDescription</key>
<string>Camera</string>
```

When all this is done as last step we need to register our native handlers for this program. Open **MauiProgram.cs** file and configure maui handlers:

MauiProgram.cs

cmbSDKMauiSample (net6.0-android) | cmbSDKMauiSample.MauiProgram

```csharp
{
    2 references
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .UseMauiCompatibility()
            .ConfigureMauiHandlers(handlers =>
            {
#if __ANDROID__
                handlers.AddCompatibilityRenderer(typeof(cmbSDKMaui.ScannerControl),
                    typeof(cmbSDKMaui.Android.ScannerControlRenderer));
#endif
#if __IOS__
                handlers.AddCompatibilityRenderer(typeof(cmbSDKMaui.ScannerControl),
                    typeof(cmbSDKMaui.iOS.ScannerControlRenderer));
#endif
            })
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
            });

        return builder.Build();
    }
}
```

Licensing the SDK

To use cmbSDK for barcode scanning with a mobile device without an MX mobile terminal, you need to install a license key. If the license key is missing, asterisks will appear instead of scanned results.

Contact your Cognex Sales Representative for information on how to obtain a license key, including 30-day trial licenses.

If you obtain cross platform license (one license for both platforms) implement an activation directly from the code when you create camera scanner:

```csharp
if (param_deviceClass == ScannerDevice.PhoneCamera)
        {
            //*****************************************************************************************
            // Create a camera scanner
            //
            // NOTE: If we're scanning using the built-in camera
            //       of the mobile phone or tablet, then the SDK requires a license key. Refer to
```

```
            //       the SDK's documentation on obtaining a license key as well as the methods for
            //       passing the key to the SDK (in this example, we're relying on an entry in
            //       info.plist and androidmanifest.xml--there are also ScannerCameraMode methods where it can be passed
            //       as a parameter).
            //*****************************************************************************************
            scannerControl.GetPhoneCameraDevice(param_cameraMode, ScannerPreviewOption.Defaults, true, "SDK_KEY");
        }
```

Otherwise if you have different keys (key for iOS only and key for Android only) add these keys inside AndroidManifest.xml or Info.plist files.

For Android add the following line in the AndroidManifest.xml file of your application under the application tag:

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

For iOS 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>
```

Implementing SDK

1. When you build your UI considering that if you want to show partial screen you need to set size for the scannerControl **by setting parent layout size.** Remember that scanner control always **fill the parent size.** ScannerControl represent the reader device and when you create camera scanner, in constructor send **false** as input parameter for fullScreen.

If you want to use full screen preview set 0 as size for parent layout of **scannerControl** and in constructor send **true** as input parameter for fullScreen (like in our sample app).

2. Set up the following event handlers to monitor the connection state of the scannerControl and receive information about the read code:

```
// This is called when a MX-1xxx device has became available (USB cable was plugged, or MX device was turned on),
// or when a MX-1xxx that was previously available has become unavailable (USB cable was unplugged, turned off due to inact
public void OnAvailabilityChanged(object sender, ScannerAvailability availability)
{
    ClearResult();
    if (availability == ScannerAvailability.Available)
    {
        ConnectToScannerDevice();
    }
    else if (availability == ScannerAvailability.Unavailable)
    {
        DisplayAlert("Device became unavailable", null, "OK");
    }
}
// The connect method has completed, here you can see whether there was an error with establishing the connection or not
// (args: ScannerExceptions exception, string errorMessage)
public void OnConnectionCompleted(object sender, object[] args)
{
    // If we have valid connection error param will be null,
    // otherwise here is error that inform us about issue that we have while connecting to scanner
    if ((ScannerExceptions)args[0] != ScannerExceptions.NoException)
    {
        // ask for Camera Permission if necessary (android only, for iOS we handle permission from SDK)
        if ((ScannerExceptions)args[0] == ScannerExceptions.CameraPermissionException)
            RequestCameraPermission();
        else
        {
```

```
            Debug.WriteLine(args[1].ToString());
            UpdateUIByConnectionState(ScannerConnectionStatus.Disconnected);
        }
    }
}
// This is called when a connection with the scanner has been changed.
// The scanner is usable only in the "Connected" state
public void OnConnectionStateChanged(object sender, ScannerConnectionStatus status)
{
    ClearResult();
    if (status == ScannerConnectionStatus.Connected)
    {
        // We just connected, so now configure the device how we want it
        ConfigureScannerDevice();
    }
    isScanning = false;
    UpdateUIByConnectionState(status);
}
// This is called after scanning has completed, either by detecting a barcode, canceling the scan by using the on-screen bu
public void OnReadResultReceived(object sender, List<ScannedResult> results)
{
    ClearResult();
    if (results?.Count > 0)
    {
       lblResultText.Text = results[0].ResultCode;
       lblResultType.Text = results[0].ResultSymbology;
    }
    isScanning = false;
    btnScan.Text = "START SCANNING";
}
```

3. Create scanner device. We will explain that in next section.

Using the MX Reader

Initialize a scanner device object for MX readers using the following factory method:

```
//*****************************************************************************************
// Create an MX-1xxx scanner  (note that no license key in needed)
//*****************************************************************************************
scannerControl.GetMXDevice();
```

The availability of the MX mobile terminal can change when the device turns on or off, or if the USB cable gets connected or disconnected. You can handle those changes using the following method from *AvailabilityChanged* event handler:

```
public void OnAvailabilityChanged(object sender, ScannerAvailability availability)
```

Using the Camera Reader

You are recommended to use an MX mobile terminal to scan barcodes. However, cmbSDK also supports using the built-in camera of a mobile device. This includes the support of optional external aimers or illumination, and the customization of the live-stream preview's appearance.

To scan barcodes using the built-in camera of a mobile device, initialize the *scanner device* object using the *scannerControl.GetPhoneCameraDevice* method. The camera reader has several options when initialized. The following parameters are required:

- *CameraMode*
- *PreviewOption*
- *FullScreen*

- *RegistrationKey*

The *ScannerCameraMode* parameter is of type *ScannerCameraMode* enum and it accepts one of the values listed in the following table.

These modes provide the following default settings for the scanner:

- The zoom feature is available and a button to control it is visible on the live-stream preview (if displayed).
- The simulated hardware trigger (volume control buttons) is disabled.
- When *StartScanning()* is called, the decoding process is started.

Based on the selected mode, additional illumination options and behaviors are set, also listed in the table.

VALUE	DESCRIPTION	ILLUMINATION	LIVE-STREAM PREVIEW
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 if the mobile device does not have an aiming accessory.	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 are ignored.	Displayed
PassiveAimer	Initializes the reader to use a passive aimer. No live-stream preview is available on the device screen in this mode, since an aiming pattern is projected.	Illumination is not available, and the live-stream preview does not have an illumination button. If commands are sent to the reader for illumination control, they are ignored because it is assumed in this mode that the built-in LED of the mobile device is being used for the aimer.	Not Displayed
ActiveAimer (for iOS only)	The live-stream preview will not be displayed when the **StartScanning()** method is called by default.	Illumination is available, if a preview option for camera preview is enabled, the illumination control button is available too . Illumination or aimer control commands are accepted .	Not Displayed
FrontCamera	Initializes the reader to use the front camera of the mobile device, if available. Use this configuration with care because most front facing cameras do not have auto focus and illumination, and provide significantly lower resolution images. Illumination is not available in this mode.	The front camera is used. Illumination is not available and the live-stream preview does not have an illumination button.	Displayed

VALUE	DESCRIPTION	ILLUMINATION	LIVE-STREAM PREVIEW
		If commands are sent to the reader for aimer or illumination control, they are ignored.	

The Scanner*PreviewOption* parameter is of type *ScannerPreviewOption* enum, and is used to change the reader's default values or override defaults derived from the selected Scanner*CameraMode*. You can specify the following options:

VALUE	DESCRIPTION
Defaults	Accept all defaults set by the CameraMode.
NoZoomButton	Hides the zoom button on the live-stream preview, preventing the user from adjusting the zoom of the mobile device camera.
NoIlluminationButton	Hides the illumination button on the live-stream preview, preventing the user from toggling the illumination.
HardwareTrigger	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 purpose-built scanner's trigger, and pressing it a second time does not stop the scanning process.
Paused	If using a live-stream preview, the preview is displayed when the *startScanning()* method is called, but the reader does not start decoding until the user presses the on-screen button to start the scanning process.
AlwaysShow	Forces a live-stream preview to be displayed even if an aiming mode is selected (for example CameraMode == PASSIVE_AIMER).
PessimisticCaching	Uses the device camera in higher resolution, changing the default 1280x720 resolution to 1920x1080 on devices that support it, and to the default resolution on devices that do not support it. This can help with scanning small barcodes, but increases the decoding time as there is more data to process in each frame.
HighResolution (for iOS only)	Use the device camera in higher resolution to help with scanning small barcodes, but slow decode time. The option sets resolution to 1920x1080 on devices that support it, and the default one on devices that do not.The default resolution is 1280x720 .
HighFrameRate	Uses the device's camera in 60 FPS instead of the default 30 FPS to provide a smoother camera preview.
ShowCloseButton	Show close button in partial view.
KeepPreviewInPausedState	Keep the preview in paused state after read or timeout.

If the *FullScreen* parameter is set **true**, a full screen preview is used, otherwise partial screen preview is in use.

The *RegistrationKey (optional)* parameter is used to license your SDK with license key that you have

Examples:

Create a reader with no aimer and a full screen live-stream preview:

```
scannerControl.GetPhoneCameraDevice(ScannerCameraMode.NoAimer, ScannerPreviewOption.Defaults, true);
```

Create a reader with no aimer, no zoom button, and using a simulated trigger:

```
scannerControl.GetPhoneCameraDevice(ScannerCameraMode.NoAimer, ScannerPreviewOption.NoZoomButton | ScannerPreviewOption.Har
```

Requesting Camera Permission for Phone Camera Scanner (for Android only)

From Android 6.0 and above you need to request permission from the user to access the built-in camera of the mobile device.

If the camera cannot be opened due to permission issues, the *OnConnectionCompleted(sender, args)* callback contains a *ScannerException* in the args parameter. You can check for this exception type and request permission.

```
public void OnConnectionCompleted(object sender, object[] args)
{
    // If we have valid connection error param will be null,
    // otherwise here is error that inform us about issue that we have while connecting to scanner
    if ((ScannerExceptions)args[0] != ScannerExceptions.NoException)
    {
        // ask for Camera Permission if necessary (android only, for iOS we handle permission from SDK)
        if ((ScannerExceptions)args[0] == ScannerExceptions.CameraPermissionException)
            RequestCameraPermission();
...
```

If camera permission is granted you can try to connect on scanner device again:

```
private async void RequestCameraPermission()
{
    var result = await Permissions.RequestAsync<Permissions.Camera>();
    // Check result from permission request. If it is allowed by the user, connect to scanner
    if (result == PermissionStatus.Granted)
    {
        scannerControl.Connect();
    }
    else
    {
        if (Permissions.ShouldShowRationale<Permissions.Camera>())
        {
            if (await DisplayAlert(null, "You need to allow access to the Camera", "OK", "Cancel"))
                RequestCameraPermission();
        }
    }
}
```

Connecting to the Device

Before connecting, set the *OnAvailabilityChanged*, *ConnectionStateChanged* and *ConnectionCompleted* event handlers.

```
public void OnAvailabilityChanged(object sender, ScannerAvailability availability)
public void OnConnectionStateChanged(object sender, ScannerConnectionStatus status)
public void OnConnectionCompleted(object sender, object[] args)
```

Invoke the connect method after initializing the *ScannerDevice*.

```
// Before the scanner can be configured or used, a connection needs to be established
private void ConnectToScannerDevice()
{
    scannerControl.Connect();
}
```

Event handlers that we set before will be called with new *ScannerDevice* status information.

Configuring MX Mobile Terminals

The MX family of mobile terminals provides sophisticated device configuration and management including saved configurations on the device. MX devices come Cognex preconfigured with most symbologies and features ready to use.

If you would like a custom configuration, reconfigure through DataMan Setup Tool, or the Cognex Quick Setup. Both tools distribute saved configurations easily to multiple devices for simple configuration management.

The mobile application is able to configure the MX device giving you the option to:

- have multiple scanning applications, each of which requiring a different set of device settings
- create your own options in a “known” state, even though the device has been pre-configured correctly

Built-in Camera

The cmbSDK employs a default set of options for barcode reading with the built-in camera of the mobile device. However, there are two important differences to keep in mind:

- The cmbSDK does not implement saved configurations for the built-in camera reader. Every time an application using the camera reader starts defaults are used automatically.
- The cmbSDK does not enable symbologies by default. The application programmer enables all barcode symbologies to scan in your application. The requisite for enabling only the needed symbologies explicitly, the application achieves most optimal scanning performance on the mobile device.

Enabling Symbologies

CmbSDK does not enable any *symbologies* by default for barcode reading with the built-in camera of the mobile device. You must enable all barcode symbologies your application needs to scan to achieve optimal scanning performance.

Individual symbologies can be enabled using the following method of the *ScannerControl* class:

```
public void SetSymbologyEnabled(Symbology symbology, bool enable)
```

All symbologies used for the symbology parameter in this method can be found in **Symbology** enum.

Examples

```
// Explicitly enable the symbologies we need
scannerControl.SetSymbologyEnabled(Symbology.Datamatrix, true);
scannerControl.SetSymbologyEnabled(Symbology.C128, true);
```

You can also use the same method to disable symbologies:

```
// Explicitly disable symbologies we know we don't need
scannerControl.SetSymbologyEnabled(Symbology.Codabar, false);
```

You can implement SymbologyEnabled event handler to check the result of the symbology change:

```
// SymbologyEnabled listener (args: Symbology symbology, bool isEnabled, string error)
public void OnSymbologyEnabled(object sender, object[] args)
{
    if ((string)args[2] != null)
        Debug.WriteLine("Failed to enable/disable " + args[0].ToString());
    else
        Debug.WriteLine(((Symbology)args[0]).ToString() + ((bool)args[1] ? " enabled" : " disabled"));
}
```

Illumination Control

If your scanner device is equipped with illumination lights, you can control them using this DMCC:

```
scannerControl.SendCommand("SET LIGHT.INTERNAL-ENABLE ON");
```

You can control lights while scanning preview is active, or if you send this DMCC before starting the scanner you will set the initial value for lights.

> Not all devices and device modes support illumination control.

Camera Zoom Settings

If the built-in camera of a mobile device is used as the reader device, you can configure zoom levels and how they are used. There are three zoom levels:

- normal: not zoomed (100%)
- level 1 zoom (150% on Android by default)
- level 2 zoom (300% on Android by default)

The *SET CAMERA.ZOOM-PERCENT [100-MAX] [100-MAX]* command is for configuring how far the two levels zoom in percentage. 100 is not zoomed and MAX (goes up to 1000) zooms as far as the device is capable of. The first argument is used for setting level 1 zoom, and the second for level 2 zoom.

You can check the current zoom setting with the *GET CAMERA.ZOOM-PERCENT* command, which returns two values: level 1 and level 2 zoom.

Example

```
scannerControl.SendCommand("SET CAMERA.ZOOM-PERCENT 250 500");
```

Note: The camera needs to be started within cmbSDK at least once to have a valid maximum zoom level. It means that if you set the zoom level to 1000 and the device can only go up to 600, the *GET CAMERA.ZOOM-PERCENT* command returns 1000 as long as camera is not opened, but it returns 600 afterwards.

GET/SET CAMERA.ZOOM 0-2 is another command that sets the zoom level or returns the actual setting. Possible values for the SET command are:

- 0 - normal (not zoomed)
- 1 - level 1 zoom
- 2 - level 2 zoom

You can call this command before or even during scanning, and the zoom goes up to the configured level. If scanning is finished, the value is reset to normal behavior (0).

Example

```
scannerControl.SendCommand("SET CAMERA.ZOOM 2");
```

Camera Overlay Customization

When using the mobile device's camera, cmbSDK allows you to see the camera preview inside a preview container or in full screen. This preview also contains a customizable overlay. The cmbSDK camera overlay features buttons for zooming, flashing and closing the scanner, and a progress bar indicating the scan timeout.

To use the legacy camera overlay originally used in cmbSDK v2.0.x and ManateeWorks SDK, use this before initializing the ScannerDevice:

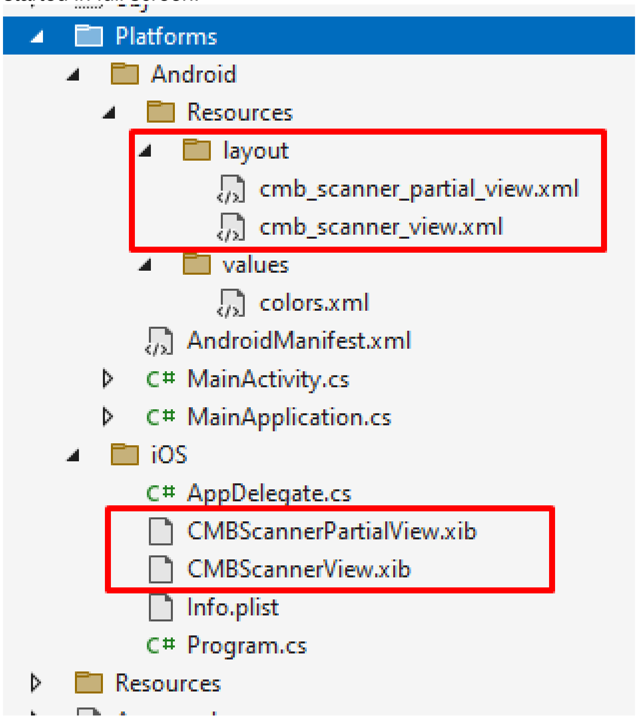
```
private void CreateScannerDevice()
{

    if (param_deviceClass == ScannerDevice.PhoneCamera)
    {
        scannerControl.SetOverlay(ScannerOverlay.LEGACY);

        scannerControl.GetPhoneCameraDevice(ScannerCameraMode.NoAimer, ScannerPreviewOption.Defaults, true);
    }
```

The customization of the legacy camera overlay is limited, so it is recommended to use the cmbSDK overlay.

When using the cmbSDK overlay:

1. Copy the layout files from the Resources/Android/layout directory into your Android platform resources and modify them. Use **cmb_scanner_partial_view.xml** if scanning is started inside a container (partial view), and use **cmb_scanner_view.xml** if scanning is started in full screen. Same for iOS, copy the xib files from the Resources/iOS directory into your iOS platform folder and modify them. Use **CMBScannerPartialView.xib** if scanning is started inside a container (partial view), and use **CMBScannerView.xib** if scanning is

started in full screen.



2. Modify the layout and xib according to your needs. For example, you can change the sizes, positions or color of the views, remove views and add your own views, like an overlay image.

> CmbSDK accesses the views it uses (zoom, flash, close buttons, the view used for drawing lines on the corners, and the progress bar) with the tag attribute. Do not change the tag attribute, otherwise cmbSDK cannot recognize the views and continues to function as if they are removed.

Both the cmbSDK and the legacy overlay allow you to change the images used on the zoom and flash buttons if your images have the same name as the names cmbSDK uses. You can find the images and names used in cmbSDK in the Resources/**drawable-mdpi** and **drawable-hdpi** directories for Android and Resources/**MWBScannerImages.xcassets** for iOS.

Both the cmbSDK and the legacy overlay allow you to change the color and width of the rectangle that is displayed when a barcode is detected. These changes need to be done inside ConfigureScannerDevice() method (after we have valid connection to scanner device).

Example:

```
private void ConfigureScannerDevice()
{
    scannerControl.SetLegacyOverlayLocationLine(255, System.Drawing.Color.Yellow, 6.0f, true);

...
```

Advanced Configuration using DataMan Control Commands

Cognex scanning devices implement DataMan Control Commands (DMCC) for configuring and controlling the device. Every feature of the device can be controlled using this text-based language. The API provides a method for sending DMC commands to the device. Commands exist both for setting and querying configuration properties. DMC commands are same for all platforms and frameworks.

The Appendix includes the complete DMCC reference for the camera reader.

> The DMCCs for MX mobile terminals and other supported devices can be found in their respective manuals available through Setup Tool.

The following examples show different DMCC sent to the device for more advanced configuration.

Examples

```
//Change the scan direction to omnidirectional
scannerControl.SendCommand("SET DECODER.1D-SYMBOLORIENTATION 0");
//Change live-stream preview's scanning timeout to 10 seconds
scannerControl.SendCommand("SET DECODER.MAX-SCAN-TIMEOUT 10");
```

You can also set ResponseReceived event handler to receive response from the send command:

```
// ResponseReceived listener after we send DMCC command (args: string payload, string error, string dmcc)
public void OnResponseReceived(object sender, object[] args)
{
    if ((string)args[1] != null)
        Debug.WriteLine("Failed to execute DMCC");
    else
        Debug.WriteLine("Response for " + (string)args[2] + ": " + (string)args[0]);
}
```

Resetting the Configuration

NOTE: This section includes resetting to CmbSDK defaults and does not include instruction on resetting to factory defaults.

CmbSDK includes a method for resetting the device to its default settings. In case of an MX mobile terminal, the default setting are the saved configurations. In case of a built-in camera, the default settings are the defaults identified in the Appendix, where no symbologies are enabled.

To reset the device send this DMCC:

```
scannerControl.SendCommand("CONFIG.DEFAULT");
```

When using an MX mobile terminal, there are three states that we can distinguish:

- Factory defaults
- Saved configuration: when there were different configurations set on the device and CONFIG.SAVE DMCC was called.
- Session configuration: when you make changes on the saved configuration, the changes are valid until the MX Mobile Terminal is rebooted. If it is rebooted, it has the saved configuration state.

Scanning Barcodes

With a properly configured reader, you are ready to scan barcodes. This is simply accomplished by calling the **StartScanning()** method from your *scannerControl*. What happens next is based on the type of *ScannerDevice* and how it has been configured. Generally:

- If using an MX terminal, 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 program calls the **StopScanning()** method.

When a barcode is decoded successfully, you will receive a ScannedResult list in your ReadResultReceived event:

```
// This is called after scanning has completed, either by detecting a barcode, canceling the scan by using the on-screen bu
public void OnReadResultReceived(object sender, List<ScannedResult> results)
```

To simply display a *ScannedResult* after scanning a barcode:

```
public void OnReadResultReceived(object sender, List<ScannedResult> results)
{
    if (results?.Count > 0)
    {
       lblResultText.Text = results[0].ResultCode;
       lblResultType.Text = results[0].ResultSymbology;
    }
}
```

Handling Disconnection

1. Disconnection:

There may be cases when a device disconnects due to low battery condition or manual cable disconnection. These cases can be detected in the *ConnectionStateChanged* event handler.

Note: The **OnAvailabilityChanged** 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 *scanner device* object before trying to call the **Connect** method.

2. Re-Connection:

After returning to your application from inactive state, the reader device remains initialized but not connected. There is no need for reinitializing the SDK but you need to re-connect.

```
//----------------------------------------------------------------------------
// When an page is disappeared, the connection to the scanning device needs
// to be closed; thus when we are resumed (Appear this page again) we
// have to restore the connection (assuming we had one).
//----------------------------------------------------------------------------
if (scannerControl.IsScannerControlReady
              && await Permissions.CheckStatusAsync<Permissions.Camera>() == PermissionStatus.Granted) {
            ConnectToScannerDevice();
}
```