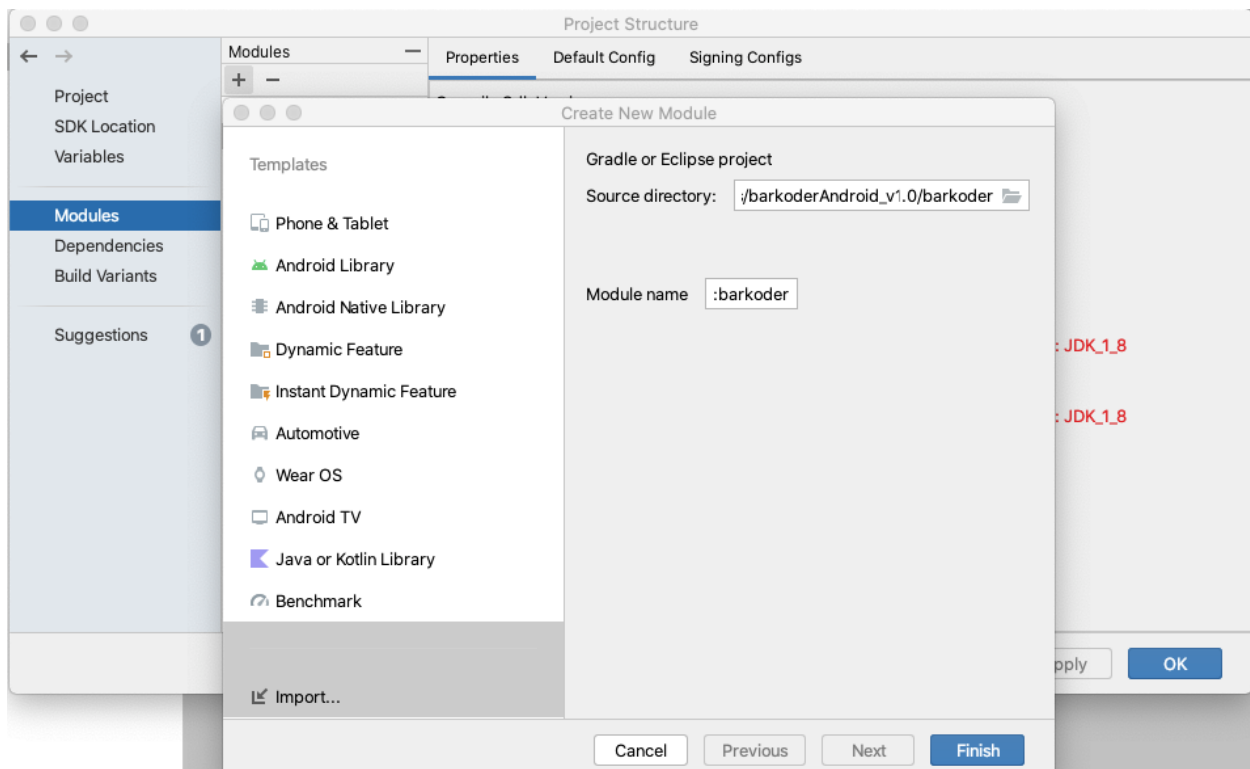


Barkoder Guide for Android v1.2.2

Installation guide

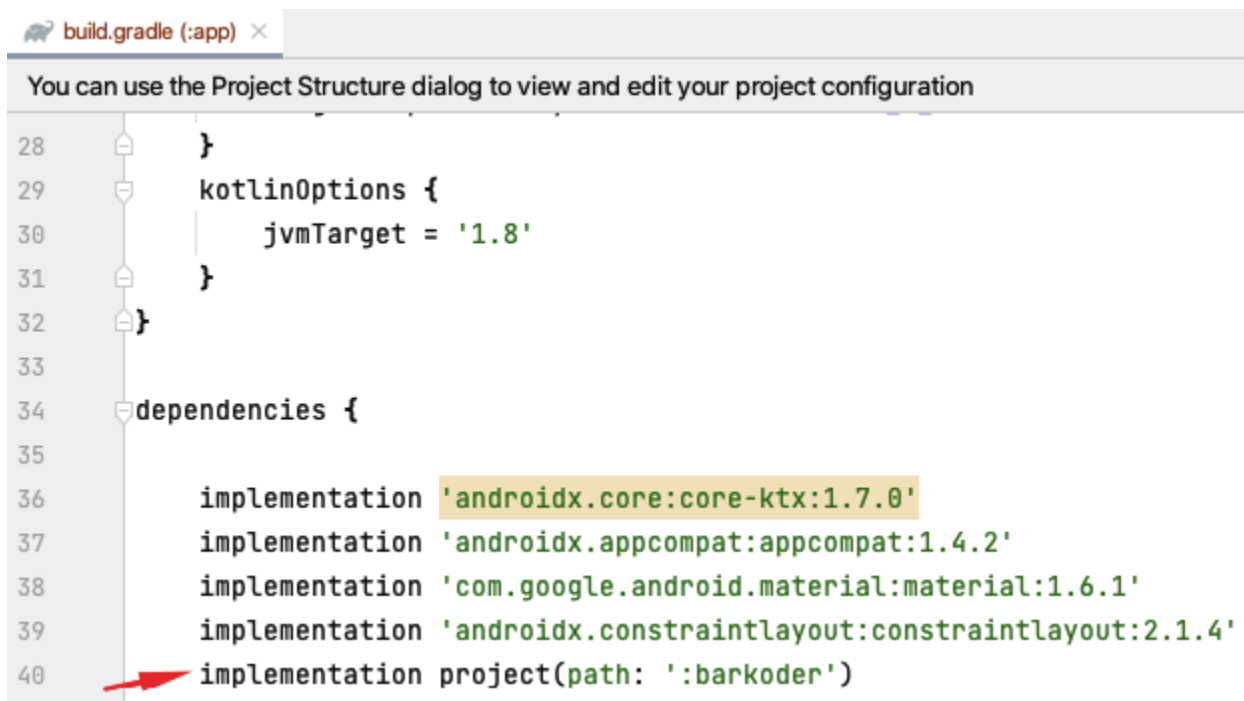
Please follow these simple steps to integrate our SDK into your Android project:

1. **Import** barkoder module in your project



Keep in mind that path in source directory should reference to the barcoder folder that contains both barkoder.aar and build.gradle, not directly to the barkoder.aar file

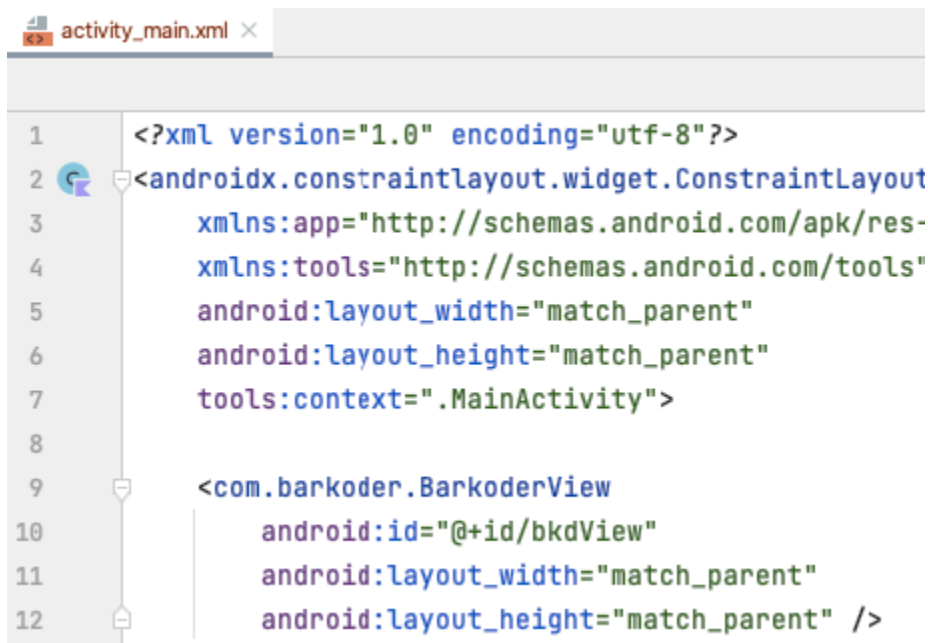
2. Add the following **dependency** in your app's build.gradle file



```
build.gradle (:app) x
You can use the Project Structure dialog to view and edit your project configuration

28     }
29     kotlinOptions {
30         jvmTarget = '1.8'
31     }
32 }
33
34 dependencies {
35
36     implementation 'androidx.core:core-ktx:1.7.0'
37     implementation 'androidx.appcompat:appcompat:1.4.2'
38     implementation 'com.google.android.material:material:1.6.1'
39     implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
40     implementation project(path: ':barkoder')
```

3. Add the following code to the layout.xml of the activity/fragment where you want the scanner to be shown



```
activity_main.xml x
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout
3     xmlns:app="http://schemas.android.com/apk/res-
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <com.barkoder.BarkoderView
10         android:id="@+id/bkdView"
11         android:layout_width="match_parent"
12         android:layout_height="match_parent" />
```

4. Create **barkoder config** per your needs or from some of our templates

```

bkdView.config = BarkoderConfig(this, "LICENSE_KEY") {
    Log.i("LicenseInfo", it.message)
}
BarkoderHelper.applyConfigSettingsFromTemplate(
    this,
    bkdView.config,
    BarkoderConfigTemplate.INDUSTRIAL_1D,
    null
)

```

5. Implement **BarkoderResultCallback** interface where you will receive scanned results

```

class MainActivity : AppCompatActivity(), BarkoderResultCallback {
    ....

    override fun scanningFinished(results: Array<Barkoder.Result>,
                                   thumbnails: Array<Bitmap>,
                                   imageResult: Bitmap?) {
        Log.i("Scanned result", results[0].textualData)
    }
}

```

6. Start the **scanning** process

```

bkdView.startScanning(this)

```

API description

BarkoderView

```

/*
 * Set camera frames callback if you want to receive the frames/images only without decoding
 * them and do your own work with the frames.
 * Don't forget to close the image at the end
 * @param previewFramesCallback
 */
fun setPreviewFramesCallback(previewFramesCallback: BarkoderPreviewFramesCallback)

/*
 * Get maximum zoom factor that can be set for the camera preview
 * Zoom factor is received in MaxZoomAvailableCallback
 * @param callback
 */

```

```

fun getMaxZoomFactor(callback: MaxZoomAvailableCallback)

/*
 \* Set zoom factor for the camera preview. If preview session is already active this zoom factor
 will be set only for this session,
 \* otherwise initial zoom will be set. Every next preview session will be started with this zoom
 factor.
 \* @param zoomFactor [1, maxZoomFactor]. Default value is 1
 */
fun setZoomFactor(zoomFactor: Int)

/*
 \* Check if current mobile device has flash available
 \* Result is received in FlashAvailableCallback.
 \* @param callback
 */
fun isFlashAvailable(callback: FlashAvailableCallback)

/*
 \* Turn flash ON/OFF
 \* If preview session is already active this state be set only for active session,
 \* otherwise the initial flash state is set. Every next preview session will be started with this
 state.
 \* @param enabled [true, false]. Default value is false.
 */
fun setFlashEnabled(enabled: Boolean)

/*
 \* Start the camera preview only, without decoding
 */
fun startCamera()

/*
 \* Stop the camera preview session and decoding if active
 */
fun stopCamera()

/*
 \* Start the camera preview (if is not already running) and the scanning process
 \* @param resultCallback
 \* @throw NullPointerException if BarkoderView config is not set
 */
fun startScanning(resultCallback: BarkoderResultCallback)

/*
 \* Stop the scanning process and the camera preview
 */

```

```

fun stopScanning()

/*
\* Pause only the scanning process. Camera preview is still running
*/
fun pauseScanning()

```

BarkoderConfig

```

/*
\* Get the decoder config object. With this object you can enable/disable decoders (barcode
types) or configure each one of them
\* @return Barkoder.Config object
*/
fun getDecoderConfig(): Barkoder.Config

/*
\* Get the location line color as integer
\* @return location line color
*/
fun getLocationLineColor(): Int

/*
\* Set the location line color as integer.
\* This line is used for marking the scanned barcode in the image result
\* @param locationLineColor [valid color representation as integer]. Default value is
Color.GREEN
*/
fun setLocationLineColor(locationLineColor: Int)

/*
\* Get the location line width as float.
\* @return location line width
*/
fun getLocationLineWidth(): Float

/*
\* Set the location line width as float.
\* This line is used for marking the scanned barcode in the image result
\* @param locationLineWidth. Default value is 4.0
*/
fun setLocationLineWidth(locationLineWidth: Float)

/*
\* Get the region of interest line color as integer

```

```

\* @return roi line color
*/
fun getRoiLineColor(): Int

/*
\* Set the region of interest line color as integer
\* @param roiLineColor [valid color representation as integer]. Default value is Color.RED
*/
fun setRoiLineColor(roiLineColor: Int)

/*
\* Get the region of interest line width as float
\* @return roi line width
*/
fun getRoiLineWidth(): Float

/*
\* Set the region of interest line width as float
\* @param roiLineWidth. Default value is 3.0
*/
fun setRoiLineWidth(roiLineWidth: Float)

/*
\* Get the region of interest background color as integer
\* @return roi overlay background color
*/
fun getRoiOverlayBackgroundColor(): Int

/*
\* Set the region of interest line color as integer
\* @param roiOverlayBackgroundColor [valid color representation as integer]. Default value is
40% transparency
*/
fun setRoiOverlayBackgroundColor(roiOverlayBackgroundColor: Int)

/*
\* Check if camera preview session will be closed when barcode is scanned
\* @return true if preview will be closed, false otherwise
*/
fun isCloseSessionOnResultEnabled(): Boolean

/*
\* Set if camera preview session should be closed when barcode is scanned
\* @param closeSessionOnResultEnabled [false, true]. Default is true
*/
fun setCloseSessionOnResultEnabled(closeSessionOnResultEnabled: Boolean)

```

```

/*
\* Check if the image result is enabled.
\* Image result is received in BarkoderResultCallback as Bitmap
\* @return true if enabled or false if is not enabled
*/
fun isImageResultEnabled(): Boolean

/*
\* Set if image result is enabled, otherwise null will be received
\* @param imageResultEnabled [false, true]. Default is false
*/
fun setImageResultEnabled(imageResultEnabled: Boolean)

/*
\* Check if barcode location in the image result is enabled.
\* If enabled, barcode in the result image will be marked
\* @return true if enabled or false if is not enabled
*/
fun isLocationInImageResultEnabled()

/*
\* Set if scanned barcode in the image result should be marked
\* @param locationInImageResultEnabled [false, true]. Default is false
*/
fun setLocationInImageResultEnabled(locationInImageResultEnabled: Boolean)

/*
\* Get active region of interest
\* @return Barkoder.BKRect object
*/
fun getRegionOfInterest(): Barkoder.BKRect

/*
\* Set region of interest in percentage
\* @param left. Default 3%
\* @param top. Default 20%
\* @param width. Default 94%
\* @param height. Default 60%
\* @throw IllegalArgumentException if input params are not valid
*/
fun setRegionOfInterest(left: Float, top: Float, width: Float, height: Float)

/*
\* Get maximum threads that are used for the decoding process
\* @return threads number as integer
*/
fun GetThreadsLimit(): Int

```

```

/*
\* Set maximum threads that will be used for the decoding process
\* @param threadsLimit [1, max threads available]
\* @throw IllegalArgumentException if input param is greater than maximum threads available
on that device
*/
fun SetThreadsLimit(threadsLimit: Int)

/*
\* Check if barcode location in preview is enabled.
\* If enabled, scanned barcode will be marked on the preview screen for short time
\* @return true if enabled or false if is not enabled
*/
fun isLocationInPreviewEnabled(): Boolean

/*
\* Set if scanned barcode should be marked on the preview screen for short time
\* @param locationInPreviewEnabled [true, false]. Default is true
*/
fun setLocationInPreviewEnabled(locationInPreviewEnabled: Boolean)

/*
\* Check if the camera preview can be zoomed with pinch
\* @return true if enabled or false if is not enabled
*/
fun isPinchToZoomEnabled(): Boolean

/*
\* Enable or disable pinch to zoom on the camera preview
\* @param pinchToZoomEnabled [true, false]. Default is false
*/
fun setPinchToZoomEnabled(pinchToZoomEnabled: Boolean)

/*
\* Check if ROI is visible on the preview screen
\* @return true if visible or false otherwise
*/
fun isRegionOfInterestVisible(): Boolean

/*
\* Set if ROI should be visible on the preview screen
\* @param regionOfInterestVisible [true, false]. Default is true
*/
fun setRegionOfInterestVisible(regionOfInterestVisible: Boolean)

/*

```



```

/* Get the active resolution. It can be Normal(HD), or HIGH(Full HD)
/* @return BarkoderResolution object
*/
fun getBarkoderResolution(): BarkoderResolution

/*
/* Set the camera resolution that will be used while scanning
/* @param barkoderResolution [BarkoderResolution.NORMAL, BarkoderResolution.HIGH].
Default is BarkoderResolution.NORMAL
*/
fun setBarkoderResolution(barkoderResolution: BarkoderResolution)

/*
/* Check if device will beep on successful scan
/* @return true if enabled or false if is not enabled
*/
fun isBeepOnSuccessEnabled(): Boolean

/*
/* Set if device should beep on successful scan
/* @param beepOnSuccess [true, false]. Default is true
*/
fun setBeepOnSuccessEnabled(beepOnSuccess: Boolean)

/*
/* Check if device will vibrate on successful scan
/* @return true if enabled or false if is not enabled
*/
fun isVibrateOnSuccessEnabled(): Boolean

/*
/* Set if device should vibrate on successful scan
/* @param vibrateOnSuccess [true, false]. Default is true
*/
fun setVibrateOnSuccessEnabled(vibrateOnSuccess: Boolean)

/*
/* Set barcode thumbnail on result
/* @param enabled. Default is true
*/
fun setThumbnailOnResultEnabled(boolean enabled)

/*
/* Set threshold between duplicates scans
/* @param thresholdBetweenDuplicatesScans. Default is 5
*/
fun setTresholdBetweenDuplicatesScans(int thresholdBetweenDuplicatesScans)

```

BarkoderHelper

```
/*
\* Scan barcode from bitmap image
\* @param image Image that you want to be scanned as Bitmap
\* @param config that will be used for scanning process
\* @param resultCallback where you will receive scanned result
\* @throw NullPointerException if config param is null
*/
fun ScanImage(image: Bitmap, config: BarkoderConfig, resultCallback:
BarkoderResultCallback)

/*
\* Apply config params from predefined template
\* @param context from where this function is called
\* @param config that will be configured
\* @param template that will be applied on config
\* @param callback that will be executed when this function is finished
*/
fun ApplyConfigSettingsFromTemplate(context: Context, config: BarkoderConfig, template:
BarkoderConfigTemplate, callback: BarkoderConfigCallback)

/*
\* Retrieve config properties from the URL and apply them in the config that is send as input
param
\* @param config that will be configured
\* @param url to the JSON file
\* @param callback that will be executed when this function is finished
*/
fun ApplyConfigSettingsFromURL(config: BarkoderConfig, url: String, callback:
BarkoderConfigCallback)

/*
\* Retrieve config properties from the URL and apply them in the config that is send as input
param
\* @param config that will be configured
\* @param filePath to the JSON file
\* @param callback that will be executed when this function is finished
*/
fun ApplyConfigSettingsFromFile(config: BarkoderConfig, filePath: String, callback:
BarkoderConfigCallback)

/*
\* Export config that is send as input param to JSON string
```

```
\* @param config that will be exported
*/
fun ConfigToJSONString(config: BarkoderConfig): String
```