Easy ML Kit Tutorials

Easy ML Kit

Easy ML Kit gives access to top-class machine learning utilities with an easy to use api.

You don't need to be a ML expert any more. Thanks to the simple and unified API design which gets you started quickly.

Easy ML Kit gives access to top-class machine learning utilities with an easy to use api.

Access features like Barcode Scanning, Object detection and more.

Currently supports Android and iOS coming soon.

Buy Now

Setup

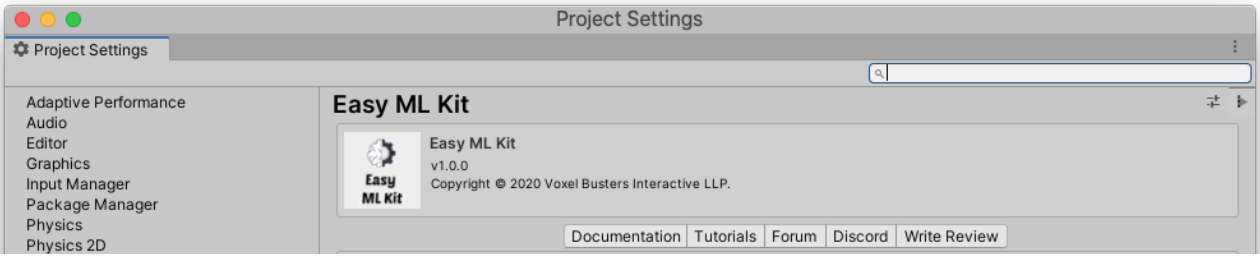
Installation

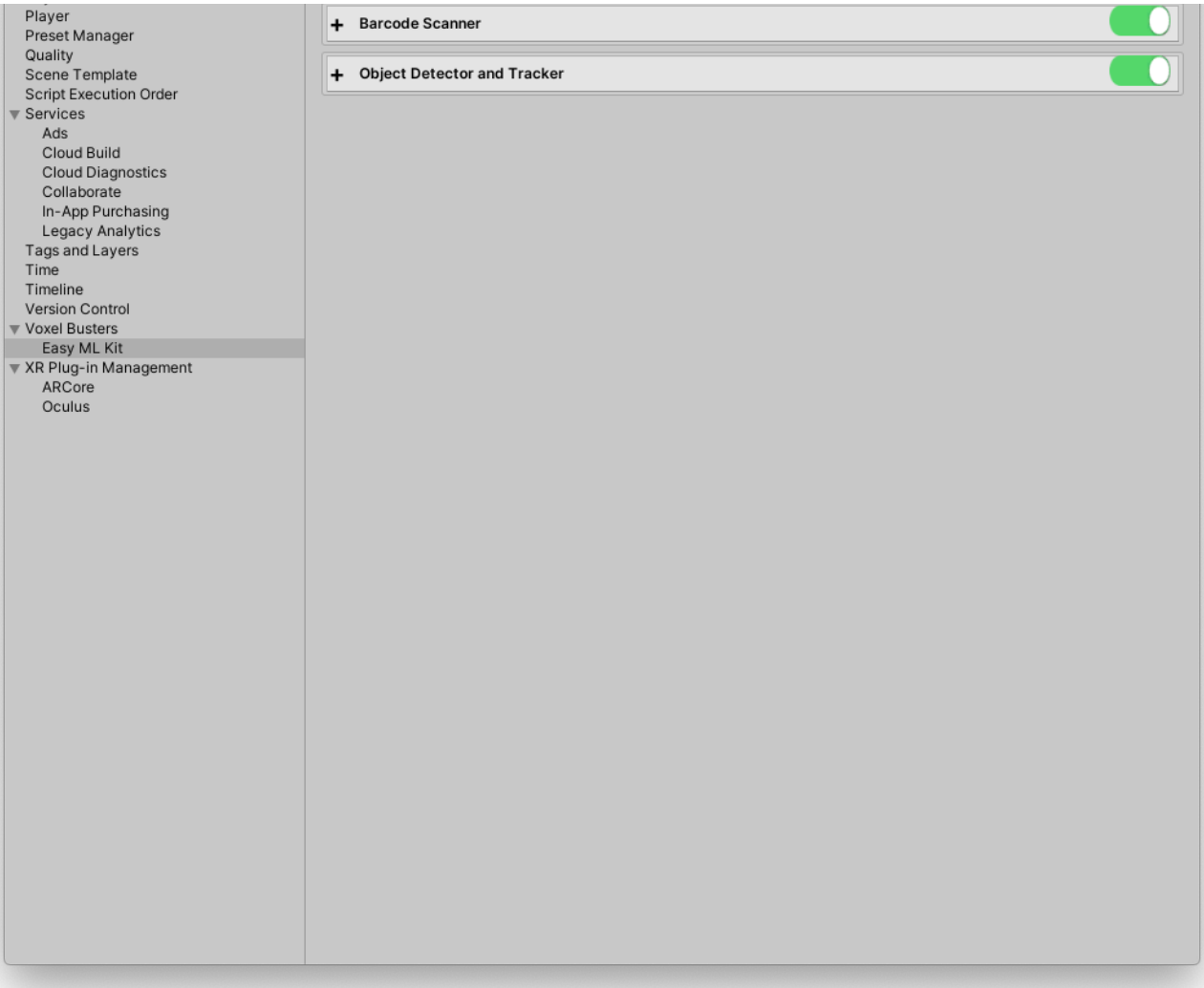
1. Uninstall any existing version from Window -> Voxel Busters -> Easy ML Kit -> Uninstall
2. Import the plugin from asset store

Settings

Settings provide a one-stop place for configuring the plugin. It allows you to enable or disable features of the plugin.

1. Open settings from Window -> Voxel Busters -> Easy ML Kit -> Open Settings
2. Toggle the on/off switches for each feature as per your requirement





Easy ML Kit Settings : Window -> Voxel Busters -> Easy ML Kit -> Open Settings

Usage Overview

> Create Input Source -> Create Feature -> Prepare with options -> Process -> Close

1. Import **VoxelBusters.EasyMLKit** and **VoxelBusters.CoreLibrary** namespaces
2. Create one of the Input Sources
3. Pass Input Source to the feature instance
4. Call Prepare with options
5. Call Process for getting the result callback
6. Close once processing is done

>

Input Sources

Image Input Source

Use **ImageInputSource** if you want to process the input in texture format.

```
using VoxelBusters.EasyMLKit;
private IInputSource CreateImageInputSource(Texture2D texture)
{
    return new ImageInputSource(texture);
}
```

Live Camera Input Source

Use **LiveCameraInputSource** if you want to process the input directly from live camera feed.

```
using VoxelBusters.EasyMLKit;
private IInputSource CreateLiveCameraInputSource()
{
    IInputSource inputSource = new LiveCameraInputSource()
    {
        EnableFlash = false,
        IsFrontFacing = false
    };

    return inputSource;
}
```

Options	Description
Enable Flash	Pass true to switch on the device flash
Is Front Facing	Pass true to enable front facing camera feed as input

AR Foundation Camera Input Source

Use **ARFoundationCameraInputSource** if you want to process the input directly from ar camera feed.

```
using VoxelBusters.EasyMLKit;
private IInputSource CreateARCameraInputSource()
{
    IInputSource inputSource = new ARFoundationCameraInputSource(arSession, arCameraManage
    return inputSource;
}
```

Pass ArSession and ArCameraManager instances for creating the **ARFoundationCameraInputSource** instance.

> To use ARFoundation supported input source, you need to add **EASY_ML_KIT_SUPPORT_AR_FOUNDATION** scripting define symbol in in player settings for each platform.

Features

Barcode Scanner

With Easy ML Kit's barcode scanning API, you can read data encoded using most standard barcode formats. Barcode scanning happens on the device, and doesn't require a network connection.

Barcodes are a convenient way to pass information from the real world to your app. In particular, when using 2D formats such as QR code, you can encode structured data such as contact information or WiFi network credentials. Because Easy ML Kit can automatically recognize and parse this data, your app can respond intelligently when a user scans a barcode.

> Import VoxelBusters.EasyMLKit and VoxelBusters.CoreLibrary namespaces

```
using VoxelBusters.EasyMLKit;
using VoxelBusters.CoreLibrary;
```

Create Instance

Create an instance of the **BarcodeScanner** instance by passing one of the input sources.

```
private BarcodeScanner CreateBarcodeScanner()
{
    IInputSource inputSource = CreateImageInputSource(TEXTURE);
    scanner = new BarcodeScanner(inputSource);
```

```
5
6       return scanner;
7   }
8
9   private IInputSource CreateImageInputSource(Texture2D texture)
10  {
11      return new ImageInputSource(texture);
12  }
13
14  private IInputSource CreateLiveCameraInputSource()
15  {
16      IInputSource inputSource = new LiveCameraInputSource()
17      {
18          EnableFlash = false,
19          IsFrontFacing = false
20      };
21
22      return inputSource;
23  }
```

Prepare

For preparing, you need to pass on **BarcodeScannerOptions** and a callback to know when prepare is complete.

```
1   private BarcodeScannerOptions CreateBarcodeScannerOptions()
2   {
3       BarcodeScannerOptions.Builder builder = new BarcodeScannerOptions.Builder();
4       builder.SetScannableFormats(BarcodeFormat.QR_CODE);//BarcodeFormat.QR_ALL;
5       return builder.Build();
6   }
7
8   private void Prepare()
9   {
10      BarcodeScannerOptions options = CreateBarcodeScannerOptions();
11      Debug.Log("Starting prepare...");
12      scanner.Prepare(options, OnPrepareComplete);
13  }
14
15  private void OnPrepareComplete(BarcodeScanner scanner, Error error)
16  {
17          Debug.Log("Prepare complete..." + error);
18          if (error == null)
19          {
20              Debug.Log("Prepare completed successfully!");
21          }
22          else
23          {
24              Debug.Log("Failed preparing Barcode scanner : " + error.Description);
25          }
26  }
```

Process

Once prepare is complete, you can start processing which gives the result in a callback.

```
private void OnPrepareComplete(BarcodeScanner scanner, Error error)
{
        Debug.Log("Prepare complete..." + error);
        if (error == null)
        {
            Debug.Log("Prepare completed successfully!");
            scanner.Process(OnProcessUpdate);
        }
        else
        {
            Debug.Log("Failed preparing Barcode scanner : " + error.Description);
        }
}

private void OnProcessUpdate(BarcodeScanner scanner, BarcodeScannerResult result)
{
    if (!result.HasError())
    {
        Debug.Log(string.Format("Received {0} Barcodes", result.Barcodes.Count));

        foreach (Barcode each in result.Barcodes)
        {
            Debug.Log(string.Format("Format : {0}, Value Type : {1}, Raw Value : {2}, Bound
        }
        if(result.Barcodes.Count > 0)
        {
            scanner.Close(null);
        }
    }
    else
    {
        Debug.Log("Barcode scanner failed processing : " + result.Error.Description, false)
    }
});
```

Close

Close the scanner once you are done processing.

```
scanner.Close(null);//Or pass a callback to know once its complete
```

Object Detector and Tracker

With ML Kit's on-device Object Detection and Tracking API, you can detect and track objects in an image or live camera feed.

Optionally, you can classify detected objects, either by using the coarse classifier built into the API.

Because object detection and tracking happens on the device, it works well as the front end of the visual search pipeline.

- **Fast object detection and tracking** Detect objects and get their locations in the image. Track objects across successive image frames.
- **Optimized on-device model** The object detection and tracking model is optimized for mobile devices and intended for use in real-time applications, even on lower-end devices.
- **Prominent object detection** Automatically determine the most prominent object in an image.
- **Coarse classification** Classify objects into broad categories, which you can use to filter out objects you're not interested in. The following categories are supported: home goods, fashion goods, food, plants, and places.
- **Classification with a custom model (Coming Soon)** Use your own custom image classification model to identify or filter specific object categories. Make your custom model perform better by leaving out background of the image.

> Import VoxelBusters.EasyMLKit and VoxelBusters.CoreLibrary namespaces

```
using VoxelBusters.EasyMLKit;
using VoxelBusters.CoreLibrary;
```

Create Instance

Create an instance of the **ObjectDetectorAndTracker** instance by passing one of the input sources.

```
ObjectDetectorAndTracker detector;
private ObjectDetectorAndTracker CreateObjectDetectorAndTracker()
{
    IInputSource inputSource = CreateImageInputSource(YOUR_TEXTURE_HERE);
    detector = new ObjectDetectorAndTracker(inputSource);
    return detector;
}

private IInputSource CreateImageInputSource(Texture2D texture)
{
    return new ImageInputSource(texture);
}

private IInputSource CreateLiveCameraInputSource()
{

    IInputSource inputSource = new LiveCameraInputSource()
```

```
    {   EnableFlash = false,
        IsFrontFacing = false
    };

    return inputSource;
}
```

Prepare

For preparing, you need to pass on **ObjectDetectorAndTrackerOptions** and a callback to know when prepare is complete.

```
private ObjectDetectorAndTrackerOptions CreateObjectDetectorAndTrackerOptions(IInputSource
{
    ObjectDetectorAndTrackerOptions.Builder builder = new ObjectDetectorAndTrackerOptions.
    builder.EnableClassification(true);
    builder.EnableMultipleObjectDetection(true);
    builder.SetClassificationConfidenceThreshold(0.5f);
    builder.SetCustomModelPath(null);
    return builder.Build();
}

private void Prepare()
{
    ObjectDetectorAndTrackerOptions options = CreateObjectDetectorAndTrackerOptions();
    Debug.Log("Starting prepare...");
    detector.Prepare(options, OnPrepareComplete);
}

private void OnPrepareComplete(ObjectDetectorAndTracker detector, Error error)
{
        Debug.Log("Prepare complete..." + error);
        if (error == null)
        {
            Debug.Log("Prepare completed successfully!");
        }
        else
        {
            Debug.Log("Failed preparing Object Detector and Scanner : " + error.Descriptio
        }
}
```

Process

Once prepare is complete, you can start processing which gives the result in a callback.

```
private void OnPrepareComplete(ObjectDetectorAndTracker detector, Error error)

{
```

```
        Debug.Log("Prepare complete..." + error);
        if (error == null)
        {
            Debug.Log("Prepare completed successfully!");
            detector.Process(OnProcessUpdate);
        }
        else
        {
            Debug.Log("Failed preparing Object detector and tracker : " + error.Description
        }
}

private void OnProcessUpdate(ObjectDetectorAndTracker detector, ObjectDetectorAndTrackerRes
{
    if (!result.HasError())
    {
        Debug.Log(string.Format("Received {0} detected objects", result.DetectedObjects.Cou

        foreach (DetectedObject each in result.DetectedObjects)
        {
            Debug.Log(string.Format("Tracking Id : {0}, Labels : {1}, Bounding Box : {2}",
        }

        if(result.DetectedObjects.Count > 0)
        {
            detector.Close(null);
        }
    }
    else
    {
        Debug.Log("Object detector failed processing : " + result.Error.Description, false)
    }
});
```

Close

Close the detector once you are done processing.

```
detector.Close(null);//Or pass a callback to know once its complete
```