EASY MOBILE Lite

User Guide

sglib

Table of Contents

GETTING STARTED

NATIVE APIs

SHARING

Easy Mobile Basic User Guide

This document is the official user guide for the Lite version of the Easy Mobile plugin for Unity by SgLib Games.

Easy Mobile Versions

- Easy Mobile Pro
- Easy Mobile Basic
- Easy Mobile Lite (Free)

Important Links

- Website
- API Reference
- Support Email
- Forum

Connect with SgLib Games

- Unity Asset Store
- Facebook
- Twitter
- YouTube

Native APIs: Introduction

The Native APIs module allows access to mobile native functionalities. The first feature available is native UI. More functionalities will be added soon.

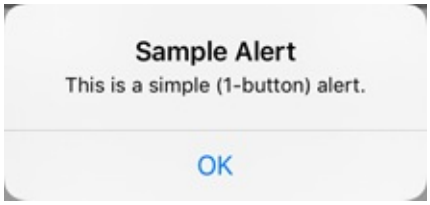
Native APIs | Native UI: Introduction

The Native UI module allows you to access native mobile UI elements such as alerts and dialogs. This module requires no configuration and all tasks can be done from script.

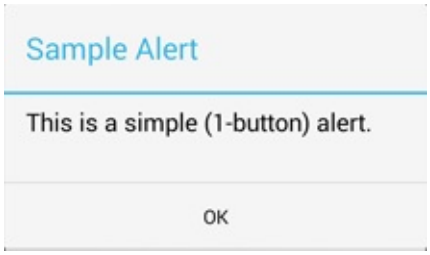
Alerts

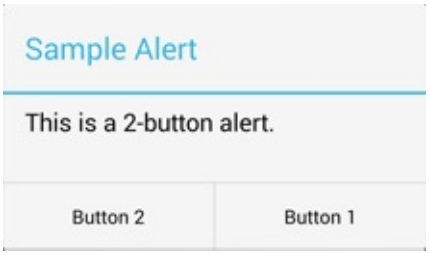
Alerts are useful in providing the users contextual information, asking for confirmation or prompting them to make a selection out of several options. An alert can have one, two or three buttons with it.

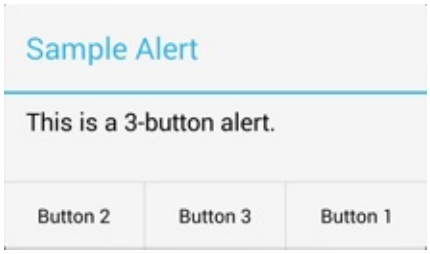
Below are the three types of alert on iOS.



And below are the three types of alert on Android.







Toasts

Toasts are short messages displayed at the bottom of the screen. They automatically disappear after a timeout. Toasts are available on Android only. Below is a sample toast message.


HOME
NATIVE UI
isFirstButtonClicked: FALSE
isSecondButtonClicked: FALSE
isThirdButtonClicked: TRUE
ALERT
2-BUTTON ALERT
3-BUTTON ALERT
A toast is a view containing a quick little message for the user. It's only available on Android.
ANDROID TOAST

This is a sample Android toast

Native APIs | Native UI: Scripting

This section provides a guide to work with Native UI scripting API.

> You can access the Native UI module API via the NativeUI class under the EasyMobile namespace.

Alerts

Alerts are available on both iOS and Android platform and can have up to three buttons.

Simple (one-button) alerts are useful in giving the user contextual information. To show a simple alert with the default OK button, you only need to provide a title and a message for the alert:

```
// Show a simple alert with OK button
NativeUI.AlertPopup alert = NativeUI.Alert("Sample Alert", "This is a sample alert with an OK button.");
```

You can also show a one-button alert with a custom button label.

```
// Show an alert with a button labeled as "Got it"
NativeUI.AlertPopup alert = NativeUI.Alert(
    "Sample Alert",
    "This is a sample alert with a custom button.",
    "Got it"
    );
```

Two-button alerts can be useful when needing to ask for user confirmation. To show a two-button alert, you need to specify the labels of these two buttons.

```
// Show a two-button alert with the buttons labeled as "Button 1" & "Button 2"
NativeUI.AlertPopup alert = NativeUI.ShowTwoButtonAlert(
    "Sample Alert",
    "This is a two-button alert.",
    "Button 1",
    "Button 2"
    );
```

Three-button alerts can be used to present the user with several options, a typical usage of it is to implement the Rate Us popup. To show a three-button alert, you need to specify the labels of the three buttons.

```
// Show a three-button alert with the buttons labeled as "Button 1", "Button 2" & "Button 3"
NativeUI.AlertPopup alert = NativeUI.ShowThreeButtonAlert(
    "Sample Alert",
    "This is a three-button alert.",
    "Button 1",
    "Button 2",
    "Button 3"
    );
```

Whenever an alert is shown, a *NativeUI.AlertPopup* object is returned, when the alert is closed, this object will fire an *OnComplete* event and then destroy itself. The argument of this event is the index of the clicked button. You should listen to this event and take appropriate action depending on the button selected.

```
// Show a three button alert and handle its OnComplete event
NativeUI.AlertPopup alert = NativeUI.ShowThreeButtonAlert(
    "Sample Alert",
    "This is a three-button alert.",
```

```
    "Button 1",
    "Button 2",
    "Button 3"
    );

// Subscribe to the event
if (alert != null)
{
    alert.OnComplete += OnAlertCompleteHandler;
}

// The event handler
void OnAlertCompleteHandler(int buttonIndex)
{
    switch (buttonIndex)
    {
        case 0:
            // Button 1 was clicked
            break;
        case 1:
            // Button 2 was clicked
            break;
        case 2:
            // Button 3 was clicked
            break;
        default:
            break;
    }
}
```

Only one alert popup can be shown at a time. Any call to show an alert while another one is being displayed will be ignored. You can check if an alert is being shown using the *IsShowingAlert* method.

```
// Check if an alert is being displayed.
bool isShowingAlert = NativeUI.IsShowingAlert();
```

Toasts

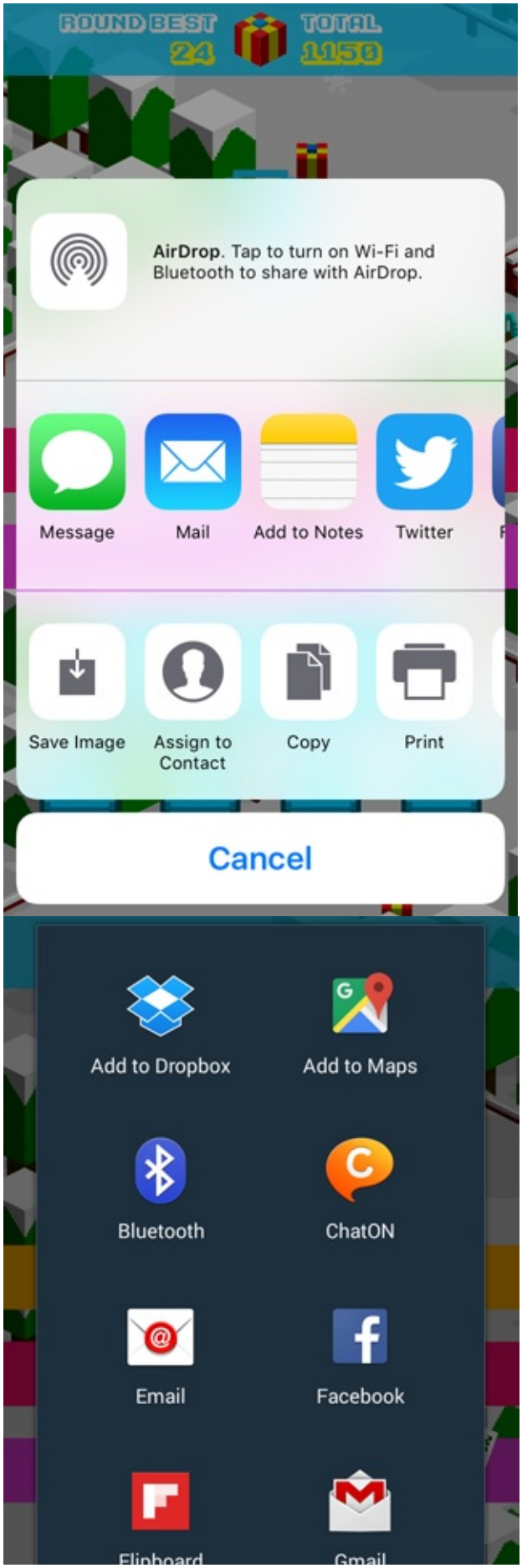
Toast is a short message displayed at the bottom of the screen and automatically disappears after a timeout. Toasts are available only on Android platform. To show a toast message:

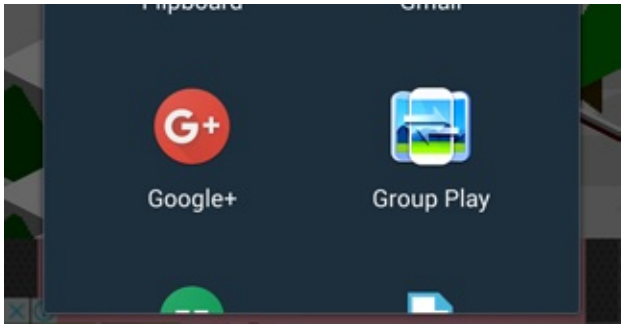
```
// Show a sample toast message
NativeUI.ShowToast("This is a sample Android toast");
```

Sharing: Introduction

The Sharing module helps you easily share texts and images to social networks including Facebook, Twitter and Google+ using the native sharing functionality. In addition, it also provides convenient methods to capture the screenshots to be shared.

Below are the sharing interfaces on iOS and Android, respectively.


ROUND BEST
24
TOTAL
1158
AirDrop. Tap to turn on Wi-Fi and Bluetooth to share with AirDrop.
Message
Mail
Add to Notes
Twitter
Save Image
Assign to Contact
Copy
Print
Cancel
Add to Dropbox
Add to Maps
Bluetooth
ChatON
Email
Facebook
Flipboard
Gmail

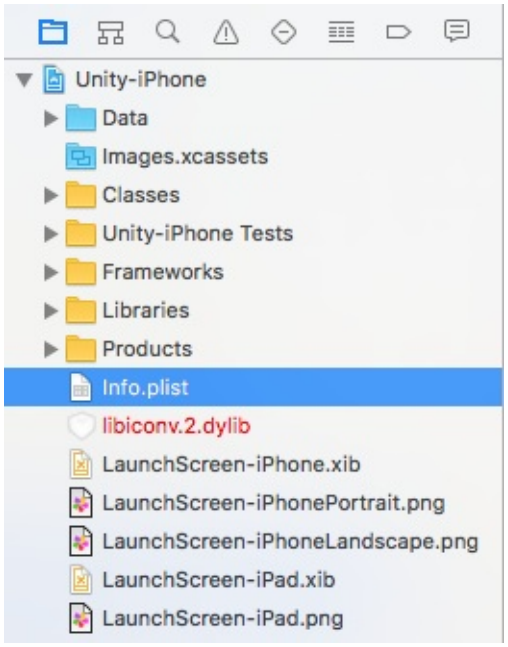


[iOS] Request Photo Library Access Permission

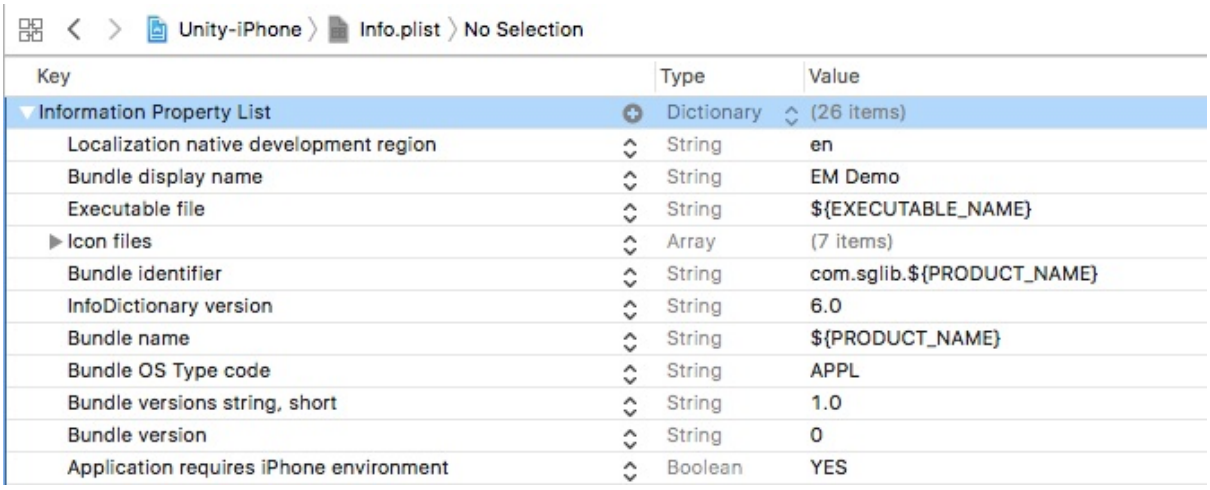
Since iOS 10, in order to use the "Save Image" feature of the sharing utility, the app needs to ask for user permission before it can access the photo library. Failure to do so will cause the app to crash as soon as the user selects the option. To request the photo library access permission, you need to add the **Privacy - Photo Library Usage Description** and **Privacy - Photo Library Additions Usage Description** properties to the Info.plist of your Xcode project.

> As of this writing, out tests show that on iOS 10, the **Privacy - Photo Library Usage Description** property is required. While iOS 11 asks for the **Privacy - Photo Library Additions Usage Description** property. Therefore it's recommended to add both properties if your target platforms including iOS 10 and above.

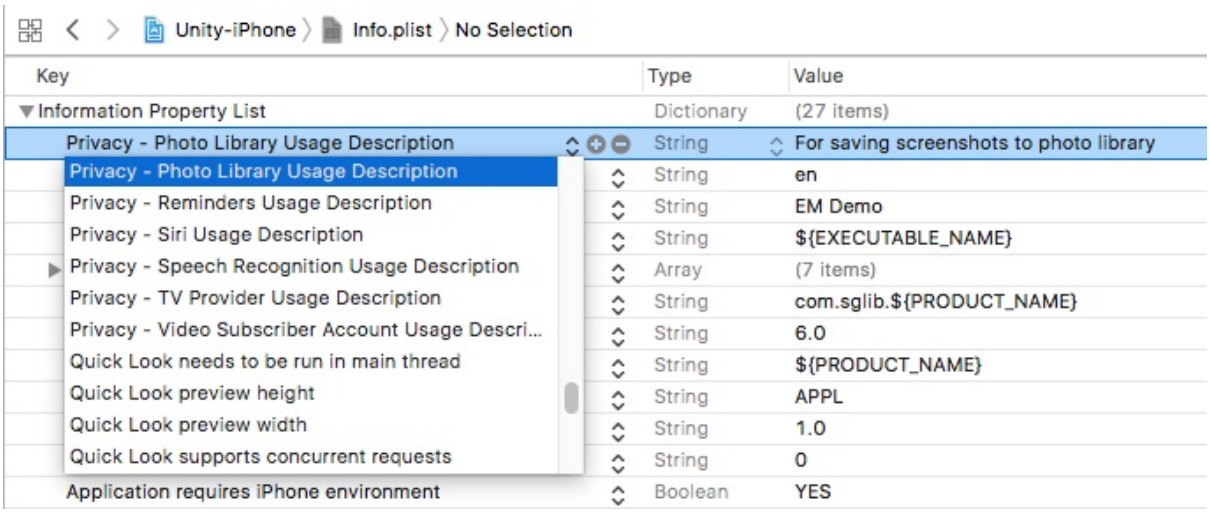
In your generated Xcode project open the Info.plist file.



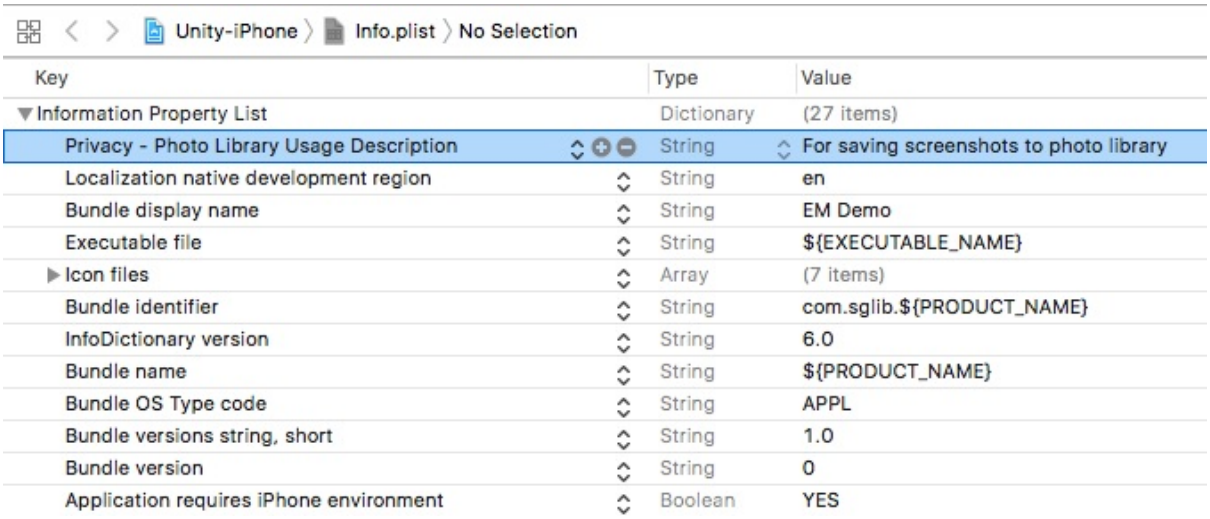
Click the **+** button on the right of **Information Property List** to add a new key.



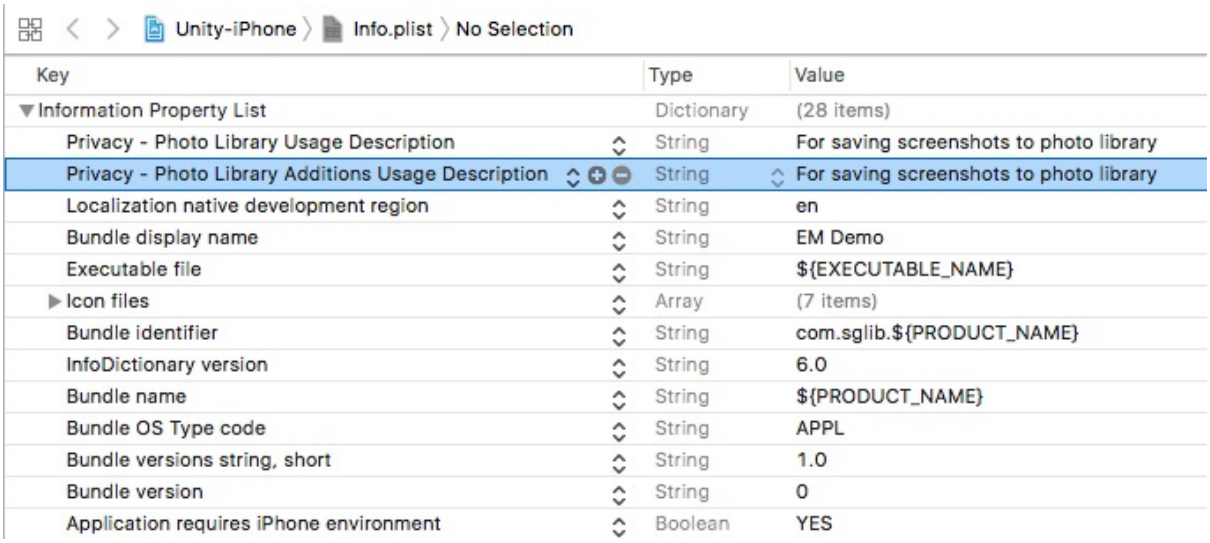
Scroll down to find the **Privacy - Photo Library Usage Description** key.



Enter a value for the key, this message will be displayed as the app requests access permission when the user selects the "Save Image" option.

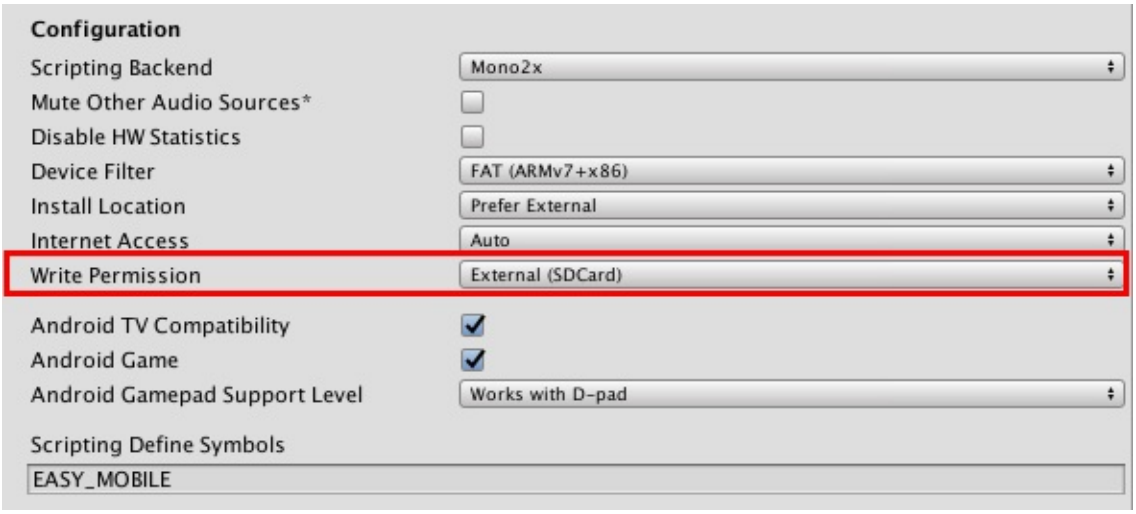


Repeat the process to add the **Privacy - Photo Library Additions Usage Description** property.



[Android] Enable External Write Permission

For this module to function on Android, it is necessary to enable the permission to write to external storage. To do so, go to *Edit* > *Project Settings* > *Player*, select *Android settings* tab, then locate the **Configuration** section and set the *Write Permission* to *External (SDCard)*.



Sharing: Scripting

This section provides a guide to work with Sharing module scripting API.

> You can access the Sharing module API via the Sharing class under the EasyMobile namespace.

Screenshot Capturing

To capture the device's screenshot, you have a few options.

Screenshot as PNG Image

To capture and save a screenshot of the whole device screen, simply specify the file name to be saved. This screenshot will be saved as a PNG image in the directory pointed by Application.persistentDataPath. Note that this method, as well as other screenshot capturing methods, needs to be called at the end of a frame (when the rendering has done) for it to produce a proper image. Therefore you should call it within a coroutine after *WaitForEndOfFrame()*.

```
// Coroutine that captures and saves a screenshot
IEnumerator SaveScreenshot()
{
    // Wait until the end of frame
    yield return new WaitForEndOfFrame();

    // The SaveScreenshot() method returns the path of the saved image
    // The provided file name will be added a ".png" extension automatically
    string path = Sharing.SaveScreenshot("screenshot");
}
```

You can also captures and saves just a portion of the screen:

```
// Coroutine that captures and saves a portion of the screen
IEnumerator SaveScreenshot()
{
    // Wait until the end of frame
    yield return new WaitForEndOfFrame();

    // Capture the portion of the screen starting at (50, 50),
    // has a width of 200 and a height of 400 pixels.
    string path = Sharing.SaveScreenshot(50, 50, 200, 400, "screenshot");
}
```

Screenshot as Texture2D

In some cases you may want to capture a screenshot and obtain a Texture2D object of it instead of saving to disk, e.g. to create a sprite from the texture and display it in-game.

```
// Coroutine that captures a screenshot and generates a Texture2D object of it
IEnumerator CaptureScreenshot()
{
    // Wait until the end of frame
    yield return new WaitForEndOfFrame();

    // Create a Texture2D object of the screenshot using the CaptureScreenshot() method
    Texture2D texture = Sharing.CaptureScreenshot();
}
```

Similar to the case above, you can also capture only a portion of the screen.

```
// Coroutine that captures a portion of the screenshot and generates a Texture2D object of it
IEnumerator CaptureScreenshot()
{
    // Wait until the end of frame
    yield return new WaitForEndOfFrame();

    // Create a Texture2D object of the screenshot using the CaptureScreenshot() method
    // The captured portion starts at (50, 50) and has a width of 200, a height of 400 pixels.
    Texture2D texture = Sharing.CaptureScreenshot(50, 50, 200, 400);
}
```

> Note that screenshot capturing should be done at the end of the frame.

Sharing

To share an image you also have a few options. You can also attach a message to be shared with the image.

> Due to Facebook policy, pre-filled messages will be ignored when sharing to this network, i.e. sharing messages must be written by the user.

Share a Saved Image

You can share a saved image by specifying its path.

```
// Share a saved image
// Suppose we have a "screenshot.png" image stored in the persistentDataPath,
// we'll construct its path first
string path = System.IO.Path.Combine(Application.persistentDataPath, "screenshot.png");

// Share the image with the path, a sample message and an empty subject
Sharing.ShareImage(path, "This is a sample message");
```

Share a Texture2D

You can also share a Texture2D object obtained some point before the sharing time. Internally, this method will also create a PNG image from the Texture2D, save it to the persistentDataPath, and finally share that image.

```
// Share a Texture2D
// sampleTexture is a Texture2D object captured some time before
// This method saves the texture as a PNG image named "screenshot.png" in persistentDataPath,
// then shares it with a sample message and an empty subject
Sharing.ShareTexture2D(sampleTexture, "screenshot", "This is a sample message");
```

Share a Text

You can share a text-only message using the *ShareText* method. Note that Facebook doesn't allow pre-filled sharing messages, so the text will be discarded when sharing to this particular network.

```
// Share a text
Sharing.ShareText("Hello from Easy Mobile!");
```

Share a URL

To share a URL, use the *ShareURL* method. On networks like Facebook or Twitter, a summary of the page will be shown if the shared URL points to a website. URLs are also useful to share GIF images hosted on sites like Giphy (see the *GIF* > *Scripting* section).

```
// Share a URL
Sharing.ShareURL("www.sglibgames.com");
```

Release Notes

Version 2.0.0

Changes

- **Editor:**
  - Updated User Guide link.
  - Upgraded to Google Play Services Resolver version 1.2.89.
  - The minimum required version of Unity is now 5.5.5f1.

Version 1.1.0

Changes

- Revised demo app.
- Revised scripting APIs.

Bug Fixes

- Fixed sharing issues on Android Nougat.

Version 1.0.0

First release.

Upgrade Guide

This section describes the required actions you may need to take when upgrading to a certain version of Easy Mobile. Please visit this place before upgrading Easy Mobile to avoid unnecessary issues.