

Unity integration with Spil Games platform



spilgames

Table of Contents

Table of Contents	1
Purpose.....	2
Audience.....	2
Get the package.....	2
Unity integration for Android.....	3
Settings.....	3
Generate, build and run an Android project	6
Application configuration	7
Application signatures	8
Parameters	9
Push Notifications.....	10
Google play.....	10
Amazon.....	11
Allowed keys.....	13
Unity integration for iOS	14
Settings.....	14
Generate the Xcode project	15
Configure the certificates for your iOS application	16
Coding.....	19
Parameters	19
JSON format (Deprecated)	22
Sample format.....	22
Allowed keys.....	23
See also.....	24

Purpose

- Build applications based on [Unity](#).
- Integrate them on [Spil Games portals](#) to benefit from their feature set.

Audience

This manual targets developers who build applications using the [Unity](#) framework, and who want to integrate its rich feature set with the services and functionality available on [Spil Games portals](#).

We assume that you are familiar with the following concepts and tasks:

- Setting up a [Unity project](#).
- Adding a [plugin](#) to a Unity project.
- Setting up an [Android developer environment](#) in Unity.
- Setting up an [iOS developer environment](#) in Unity.

Get the package

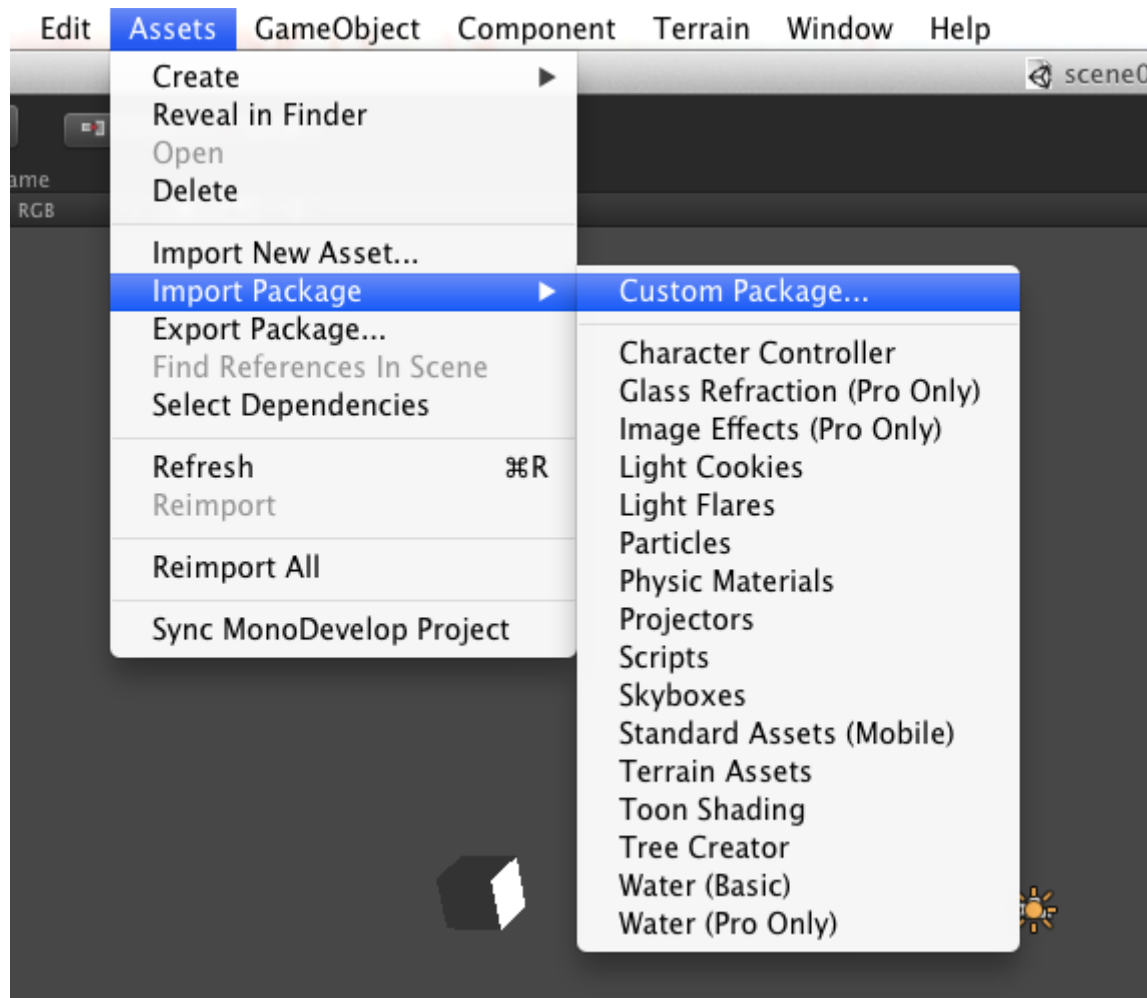
You can obtain the [Unity framework](#) and the libraries you need to set up your project from Spil Games. It contains the following components:

- **Spil.untypackage:** required if the application is [Unity](#)-based.
- *Unity sample:* example of integration with a Unity-based project using the Android plugin for the Spil framework.
- *Framework documentation:* documentation for the Android framework.

Unity integration for Android

Settings

After setting up your Unity project, import **spil.unitypackage**, as shown below:



The **spil.unitypackage** includes a folder with the following elements:

- Unity plugin
- Post processing plugin
- Android files (**spil_core.jar** and **spil_lib.jar**).

✓ **Note:** if your Unity project already includes an **AndroidManifest.xml** file, don't override it with the one shipped with the package. Instead, add the following lines to your existing **AndroidManifest.xml** file:

- Inside the `<application>` tag, insert the following line:

```
android:name="com.spilgames.framework.SpilApplication"
```

- Insert the appropriate receiver: either the *default receiver*, or the *receiver for the Fiksu SDK* (for further details, see the *Settings section in the integration document for Android*).
- Insert the following *uses-permissions*:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

- Insert the following *receiver*:

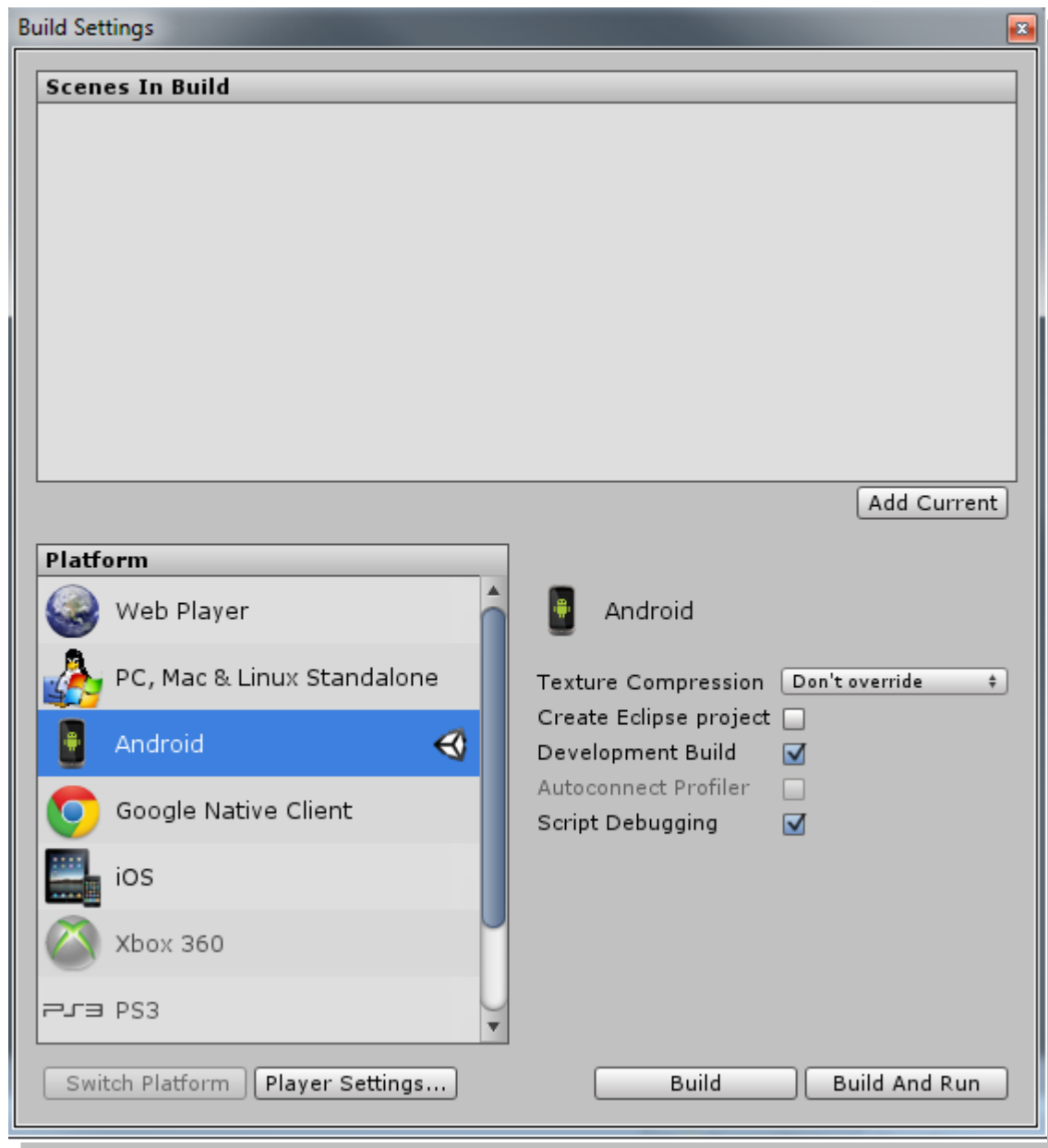
```
<receiver
  android:name="com.spilgames.framework.receivers.SpilInstallReferrerReceiver"
  android:exported="true" >
  <intent-filter>
    <action android:name="com.android.vending.INSTALL_REFERRER" />
  </intent-filter>
</receiver>
```

- ❑ **Note:** if you are implementing the [Fiksu SDK](#), inside the `<application>` tag insert the *receiver* described below instead of the one shown above:

```
<receiver
  android:name="com.fiksu.asotracking.InstallTracking"
  android:exported="true" >
  <intent-filter>
    <action android:name="com.android.vending.INSTALL_REFERRER" />
  </intent-filter>
  <meta-data
    android:name="forward.1"
    android:value="com.spilgames.framework.receivers.SpilInstallReferrerReceiver" />
</receiver>
```

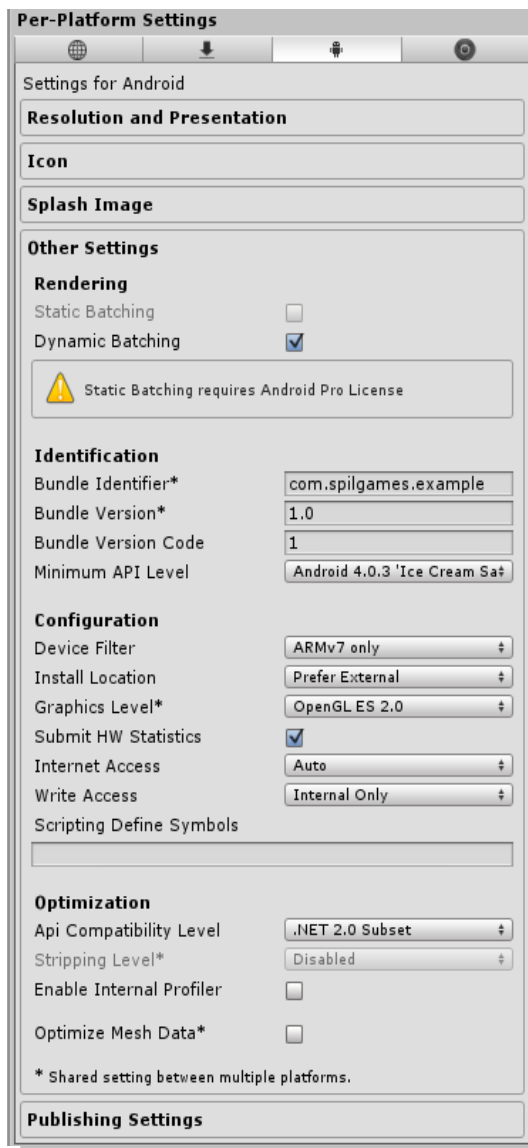
Generate, build and run an Android project

- From the menu, select the **Build settings...** command.
- In the **Build Settings** dialog window under **Platform**, select **Android**, as shown below.



Application configuration

- Add your package name to the **Bundle Identifier**. This package name needs to be *unique* because it identifies your application in the stores.
- Change **Bundle version**. This version corresponds to the application version as shown in the stores.
- Change the **Bundle Version Code**. If you are updating your application, the version number value needs to be increased. Otherwise, the stores will not consider your application as having received an update.
- The **minimum API Level** needs to be **4.0.3**: the framework does not work with previous versions.



Per-Platform Settings

Settings for Android

Resolution and Presentation

Icon


Splash Image

Other Settings

Rendering

Static Batching ☐

Dynamic Batching ☒

 Static Batching requires Android Pro License

Identification

Bundle Identifier*

Bundle Version*

Bundle Version Code

Minimum API Level

Configuration

Device Filter

Install Location

Graphics Level*

Submit HW Statistics ☒

Internet Access

Write Access

Scripting Define Symbols

Optimization

Api Compatibility Level

Stripping Level*

Enable Internal Profiler ☐

Optimize Mesh Data* ☐

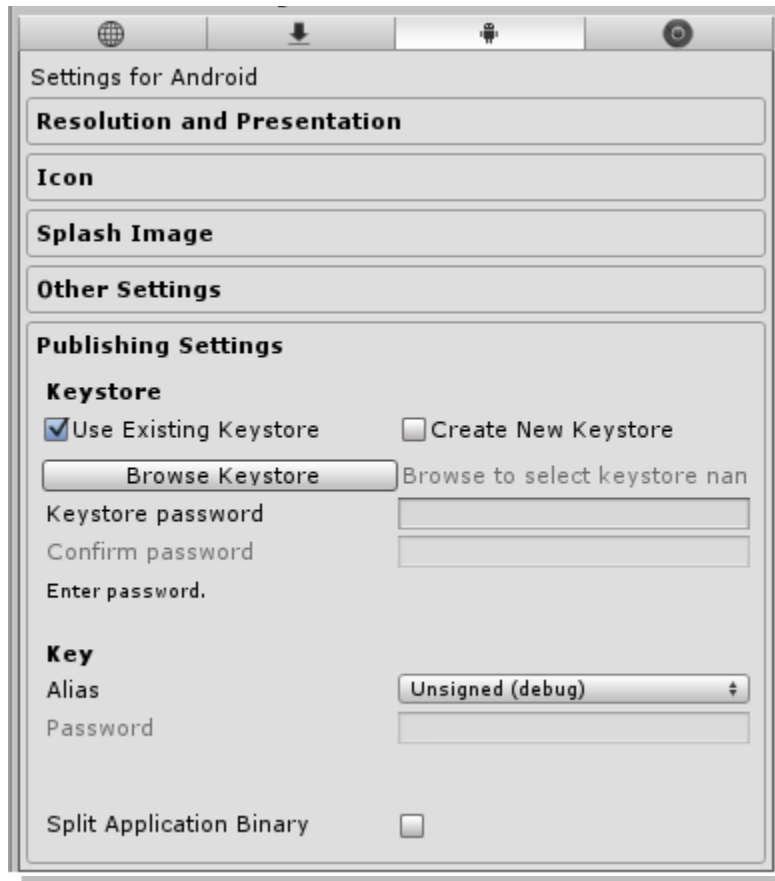
* Shared setting between multiple platforms.

Publishing Settings

Application signatures

If you are not using a development build, you need to [sign your application](#) before uploading it to the stores:

- In your file system, browse to your key store.
- Add the key store password.
- Select the key you want to sign the application with, and the corresponding password.



Parameters

Initialize the environment with the `Initialize` method. The method requires the following parameters:

- `appId` and `authToken`: these two parameters are provided by Spil Games when you get the package.
- `configs`: this parameter is a *struct* and it needs to include a number of defined keys (see [Allowed keys](#) further on in this document).

The following example shows the configuration settings to initialize the environment:

```
using Spil;
using LitJson;
public class Cube : MonoBehaviour, SpilAdsListener, SpilInGameAdsListener,
SpilAppSettingsListener {
    SpilUnity instance;
    void Start () {
        instance = (SpilUnity)GetComponent<SpilUnity>();
        SpilSettings configs;
        configs.SG_ENVIRONMENT_KEY = enviroment.SG_ENVIRONMENT_LIVE_VALUE;
        configs.SG_TRACKING_ID_KEY="<tracking-app-ids>";
        configs.SG_STORE_ID = store.SG_STORE_ANDROID;
        instance.Initialize("<spil-app-id>","<spil-auth-token>",configs);

        instance.setAdsListener(this);
        instance.setInGameAdListener(this);
        instance.setAppSettingsListener(this);
    }
}
```

- You need to implement:
 - `SpilAdsListener`,
 - `SpilInGameAdsListener`, and
 - `SpilAppSettingsListener`.

You can implement them where appropriate in your code.

- You need to pass references to the `Spil` object using `setXListener(listenerImplementation)`; you need to replace "X" with the appropriate listener.

Push Notifications

The Android framework supports push notifications for [Google play](#) and [Amazon](#). To include this feature in your game, follow the steps described below.

Google play

In the framework, set `SG_STORE_KEY` to `SG_STORE_GOOGLE_PLAY`. In this way, the framework can recognize the [Google play store](#).

- ❑ **Note:** in the following code examples you need to replace the `[YOUR_GAME_PACKAGE]` placeholder with the *actual name of your package*, i.e. the bundle identifier in the Android settings.

1. In the **AndroidManifest.xml** file, insert the following *uses-permissions*:

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<uses-permission android:name="com.spilgames.examples.permission.C2D_MESSAGE" />
```

2. In the **AndroidManifest.xml** file, insert the following *permission*:

```
<permission
    android:name="com.spilgames.examples.permission.C2D_MESSAGE"
    android:protectionLevel="normal" />
```

3. In the **AndroidManifest.xml** file, insert the following *receivers*:

```
<receiver android:name="com.spilgames.framework.receivers.NotificationsReceiver"
    android:permission="com.google.android.c2dm.permission.SEND" >
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <action android:name="com.google.android.c2dm.intent.RETRY" />
        <category android:name="[YOUR_GAME_PACKAGE]" />
    </intent-filter>
</receiver>
```

Amazon

In the framework, set `SG_STORE_KEY` to `SG_STORE_AMAZON`. In this way, the framework can recognize the [Amazon store](#).

- ✓ **Note:** in the following code examples you need to replace the `[YOUR_GAME_PACKAGE]` placeholder with the *actual name of your package*, i.e. the bundle identifier in the Android settings.

1. In the **AndroidManifest.xml** file, inside the `<manifest>` tag, insert the following *xmlns*:

```
xmlns:amazon="http://schemas.amazon.com/apk/res/android"
```

2. In the **AndroidManifest.xml** file, insert the following *uses-permissions*:

```
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.amazon.device.messaging.permission.RECEIVE" />
<uses-permission android:name="[YOUR_GAME_PACKAGE].permission.RECEIVE_ADM_MESSAGE" />
```

3. In the **AndroidManifest.xml** file, insert the following *permission*:

```
<permission
    android:name="[YOUR_GAME_PACKAGE].permission.RECEIVE_ADM_MESSAGE"
    android:protectionLevel="signature" />
```

4. In the **AndroidManifest.xml** file, inside the `<application>` tag, insert the following *enable-feature*:

```
<amazon:enable-feature
    android:name="com.amazon.device.messaging"
    android:required="false" />
```

5. In the **AndroidManifest.xml** file, insert the following *receivers*:

```
<receiver android:name="com.spilgames.framework.receivers.AmazonReceiver$Receiver"
    android:permission="com.amazon.device.messaging.permission.SEND" >
    <intent-filter>
        <action android:name="com.amazon.device.messaging.intent.RECEIVE" />
        <action android:name="com.amazon.device.messaging.intent.REGISTRATION" />
        <category android:name="[YOUR_GAME_PACKAGE]" />
    </intent-filter>
</receiver>
```

6. In the **AndroidManifest.xml** file, insert the following *service*:

```
<service android:name="com.spilgames.framework.receivers.AmazonReceiver "
    android:exported="false" />
```

7. Contact Spil Games to request an *Amazon key* for the application.
8. When you receive the Amazon key from Spil Games, create a new file in the **.../Assets/Plugins/Android/assets** folder and call it **api_key.txt**, then add the Amazon key to this file.

Allowed keys

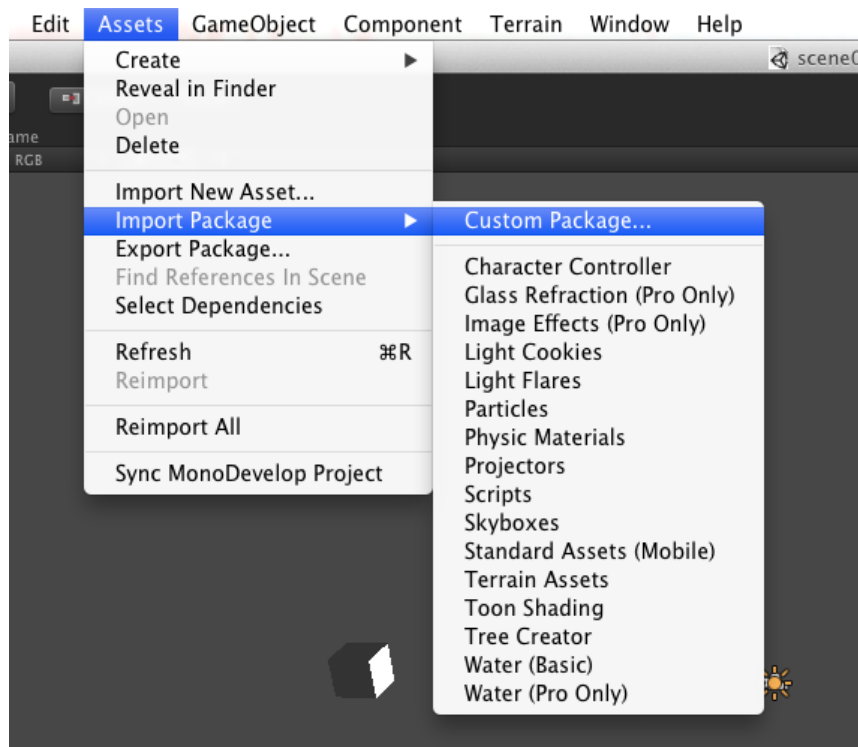
Key: `SG_ENVIRONMENT_KEY`
Description: sets the environment you want to work in: either development or production.
Values: `SG_ENVIRONMENT_DEV_VALUE, SG_ENVIRONMENT_LIVE_VALUE`
Mandatory: YES

Key: `SG_STORE_KEY`
Description: sets the store the application is deployed to.
Values: `SG_STORE_IOS, SG_STORE_AMAZON, SG_STORE_GOOGLE_PLAY`
Mandatory: YES

Unity integration for iOS

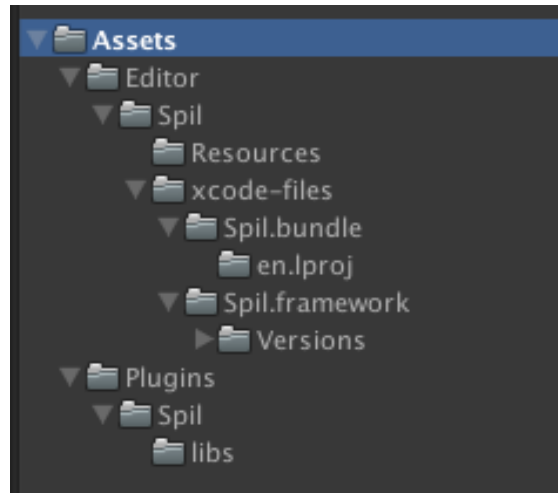
Settings

1. After setting up your Unity project, import **spil.unpackage**, as shown below:



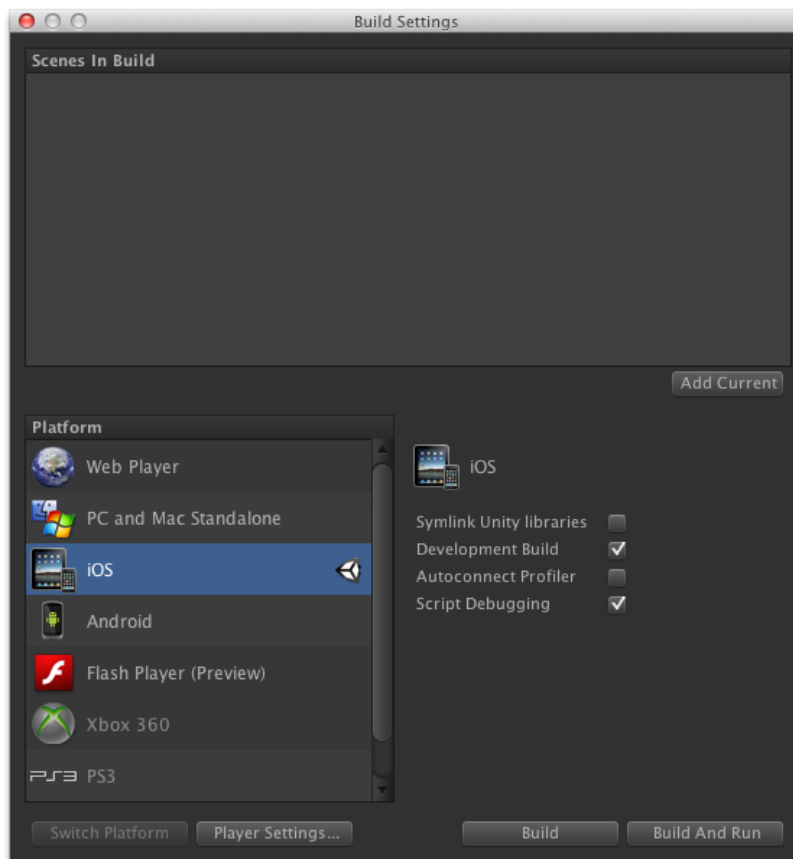
2. The **spil.unpackage** includes a folder with the following elements:
 - *Unity plug-in*
 - **spilgames_default_settings.json**
 - *Post processing plugin*
 - *Xcode files (Spil.bundle and Spil.framework)*

Now your project should look like this:



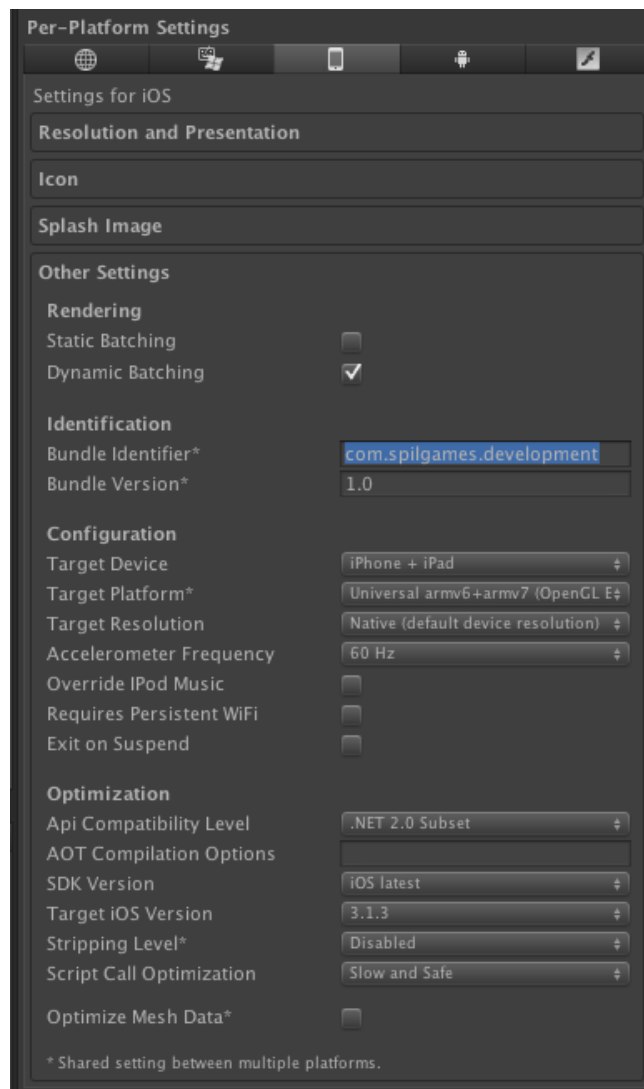
Generate the Xcode project

1. From the menu, select the **Build settings...** command.
2. Under **Platform**, select **iOS**, as shown below:



Configure the certificates for your iOS application

1. Click the **Player settings...** button. A panel opens.
2. Set up the general fields like icons, orientation, preferred, and so on.
3. Select the target platforms, the available architectures, the target devices, and the iOS level.
4. In the **Bundle Identifier** field, set the name of the application you want to build, as shown below:



✓ **Note:** the application name must be *exactly the same* as the corresponding name in the provisioning profile.

5. Set **Stripping Level** to **Strip Assemblies**, as shown below:



6. Select **Build**. This generates the Xcode project that will compile the final application.

7. In the open Xcode project, the following items should be included in the project:

- **Spil.framework**
- *Resource bundle (Spil.bundle)*
- **spilgames_default_settings.json file.**

✓ Notes:

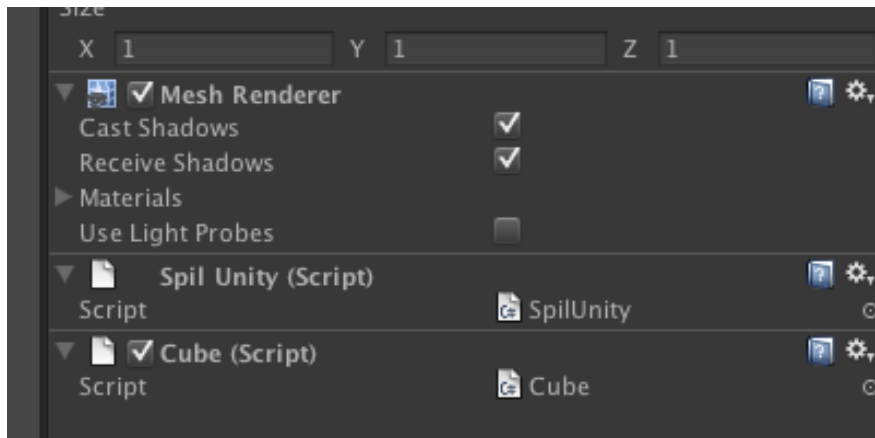
- Use **spilgames_default_settings.json** and the resource bundle as a reference to the corresponding files in the Unity folder.
 - Make sure the following frameworks are included as well, as shown below:
 - **UIKit.framework**
 - **QuartzCore.framework**
 - **SystemConfiguration.framework**
 - **CoreLocation.framework [weak link]**
 - **libsqlite3.0.dylib**
 - **CFNetwork.framework**
 - **CoreGraphics.framework**
 - **AdSupport.framework [weak link]**
 - **AssetsLibrary.framework**
 - **CoreVideo.framework**
 - **CoreMedia.framework**
 - **MobileCoreServices.framework**
 - **StoreKit.framework [weak link]**
 - **AVFoundation.framework**
 - **CoreTelephony.framework**
 - **CoreData.framework**
8. Set the flags for the compiler and specify them, if they are not defined:
- 8.1 In the project **Build Settings** tab, under the **Setting** category, expand the **Other C Flags** node.
 - 8.2 Add the C flag `-mno-thumb` to the list.
 - 8.3 In the project **Build Settings** tab, under the **Setting** category, expand the **User-defined** node.
 - 8.4 Add the user-defined setting `GCC_THUMB_SUPPORT` and set its value to `NO`.

Coding

1. Include the Spil namespace to your project and to the JSON parsing API. This works in the same way as in pure native applications:

- `using Spil;`
- `using LitJson;`

2. Add the **SpilUnity.cs** script to an object.



The camera is an ideal object because it behaves like a [singleton](#), but any other object will work as well.

Parameters

Initialize the environment with the `Initialize` method in the same way as you do for pure native applications (see the *Pure native applications* and *Coding for pure native applications* sections in the iOS integration document).

- `appID` and `authToken`: the first two parameters are provided by Spil Games when you get the package.
- `configs`: this is the last parameter. It is a struct, and it needs to include a number of defined keys. The keys are the same you use for pure native applications (see [Allowed keys](#) further on in this document).

The following is an example of configuration settings used to initialize the environment (see following page):

```
using Spil;
using LitJson;
public class Cube : MonoBehaviour, SpilAppSettingsListener, SpilAdsListener {
    SpilUnity instance;
    Vector3 rotation = Vector3.zero;
    // Use this for initialization
    void Start () {
        instance = (SpilUnity)GetComponent<SpilUnity>();
        SpilSettings configs;
        configs.SG_ENVIRONMENT_KEY = enviroment.SG_ENVIRONMENT_LIVE_VALUE;
        configs.SG_TRACKING_ID_KEY="<tracking-app-ids>";

        instance.Initialize("<spil-app-id>","<spil-auth-token>",configs);
        instance.GetSettings(this);
        instance.StartAds(this);
        instance.GetABTest(this);
    }
}
```

✓ Notes:

- You need to implement the `SpilAppSettingsListener` and `SpilAdsListener` interfaces. Optionally, you can implement also the `SpilABTestListener` interface. You can implement them where appropriate in your code.
- You need to pass references to the Spil object using the `GetSettings(listenerImplementation)`, `GetAds(listenerImplementation)` and `GetABTest(listenerImplementation)` methods, respectively.

Refer to the following example:

```
// App settings listener
public void AppSettingsDidLoad(JsonData data){
    renderer.material.color = new Color(
        (((int)data["color"]>>16)&0x000000ff)/255.0f,
        (((int)data["color"]>>8)&0x000000ff)/255.0f,
        (((int)data["color"])&0x000000ff)/255.0f);

    rotation = new Vector3((int)data["rotation"]["x"],
                           (int)data["rotation"]["y"],
                           (int)data["rotation"]["z"]);
}
public void AppSettingsDidFailWithError(string error){
    Debug.LogError(error);
}
// Ads listener
public void AdDidStart(){
    Debug.Log("started");
}
public void AdDidFailToStart(string error){
    Debug.LogError(error);
}
public void AdWillAppear(){
    Debug.Log("will appear");
}
public void AdDidAppear(){
    Debug.Log("appeared");
}
public void AdDidFailToAppear(string error){
    Debug.LogError(error);
}
public void AdPopupDidDismiss(){
    Debug.Log("popup was dismissed");
}

public void AdMoreGamesWillAppear(){
    Debug.Log("more games will appear");
}
public void AdMoreGamesDidAppear(){
    Debug.Log("more games appeared");
}
public void AdMoreGamesDidFailToAppear(string error){
    Debug.LogError(error);
}
public void AdMoreGamesDidDismiss(){
    Debug.Log("more games were dismissed");
}

public void ABTestSessionDidStart(){
    Debug.Log("A/B test session started");
    instance.ABTestGetTestDiff();
}
public void ABTestSessionDidEnd(){
    Debug.Log("A/B test session ended");
}
public void ABTestSessionDiffReceived(JsonData diffs){
    Debug.Log("A/B test differences received");
}
}
```



Now you can compile the project in [Xcode](#) and run it directly in the devices. The example above includes parsing the default settings provided to modify color and rotation of the object.

For further details, refer to the Unity sample available in package you received from Spil Games.

JSON format (Deprecated)

App Settings require additional metadata in the JSON format to render the front-end admin. This makes the format slightly more verbose than usual.

Below is the [BNF](#) definition of the format.

```
<JSON> ::= { <pair> [, <pair>]* }

<pair> ::= "key": <obj>

<obj> ::= { "type":"string", "value": <string> } |
          { "type":"number", "value": [+|-]<number> } |
          { "type":"bool", "value": true|false } |
          { "type":"array", "value": <obj> }
```

Sample format

```
{
  "color":{"type":"number","value":"0x00ff00"},
  "rotation":{
    "type":"array",
    "value":{
      "x":{"type":"number","value":"0"},
      "y":{"type":"number","value":"0"},
      "z":{"type":"number","value":"0"}
    }
  },
  "hide":{"type":"bool","value": false}
}
```

✔ **Note:** when the delegate/listener is called back, you need to define the following data in both native and Unity-based apps to correctly parse the settings:

- Format
- Name
- Types.

Allowed keys

Key: `SG_ENVIRONMENT_KEY`

Description: sets the environment to work in: *development* or *production*.

Values: `SG_ENVIRONMENT_DEV_VALUE`, `SG_ENVIRONMENT_LIVE_VALUE`

Mandatory: YES

Key: `SG_ENVIRONMENT_SETTINGS_URL_GET`

Description: the URL the app settings are stored in. It must point to a JSON file.

Values: a `NSURL` object.

Mandatory: YES, if `SG_ENVIRONMENT_KEY` is set to `SG_ENVIRONMENT_DEV_VALUE`

Key: `SG_APP_SETTINGS_POLL_TIME_KEY`

Description: the refresh interval for the app settings. Values are in seconds.

Used only if `SG_ENVIRONMENT_KEY` is set to `SG_ENVIRONMENT_DEV_VALUE`

If no value is specified, the default setting is one (1) second.

Values: float

Mandatory: *No*

See also

- [Doxygen](#)-generated documentation, available on [GitHub](#): further explanations of the methods used to set the delegates for the App Settings and Ads sub-systems.
- [Samples](#) available on GitHub: further details about setup and configuration parameters for Unity-based application projects.