

## Android integration with Spil Games platform

---



**spil**games

## Table of Contents

Table of Contents .....	1
Purpose.....	2
Audience.....	2
Get the package.....	2
Spil Loader feature .....	3
Pure native Android applications .....	4
Settings.....	4
Parameters .....	6
Listeners .....	7
AdListener .....	7
Unity-based applications.....	8
Settings.....	8
Generate, build and run the Android project.....	10
Application configuration .....	11
Application signatures .....	12
Parameters .....	13
Allowed keys.....	14

## Purpose

- Build native application for Android.
- Integrate them on Spil Games portals to benefit from their feature set.

## Audience

This manual targets developers who build native applications for Android and who want to use the features and services through the Spil Games platform.

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

- Setting up an [Android project](#).
- Adding [external.jar](#) files to Android projects
- Creating a [key store](#) for signing Android applications.

## Get the package

You can obtain the Android framework and the libraries you need to set up the project from Spil Games. It contains the following components:

- **Spil.framework:** required if the application is pure Android native.
- **Spil.unityEngine:** required if the application is [Unity](#)-based.
- *Native sample:* example of a native project integration with the Android Spil framework
- *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.

## Spil Loader feature

Android Spil Framework has two libraries:

- **spil\_core.jar** is the *core* of the framework. This library dynamically loads the **spil\_lib.jar** library. The core of the framework does not change during the game lifetime (unless a major update of the framework or the game is released).
- **spil\_lib.jar** is the *dynamic* component of the framework. The core loads it during runtime. Before loading it, the core checks if a newer version of this library is available, and in case it downloads it. When a newer version of the **spil\_lib.jar** library is available and it is downloaded, any changes related to the new library become effective the following time you run the game.

This feature allows Spil Games to fix any possible bugs in the main library (**spil\_lib.jar**), and to be totally transparent for both the user and the developer. In case of a bug in the library, there is no need to update the game in the stores.

In case of a major change in the framework or when new features are implemented, it is necessary to update the game in the stores.

The Spil Loader feature works both in [native Android](#) and in [Unity-based](#) apps.

# Pure native Android applications

## Settings

To set up your Android project to use Spil Games Android framework, do the following:

- In your Android project, add the **spil\_core.jar** file to the **libs** folder.
- Add the **spil\_lib.jar** file to your **assets** folder.
- In your **AndroidManifest.xml** file, do the following:
  - Rename the application name to: **com.spilgames.framework.SpilApplication**.
  - Inside the `<application>` tag, insert the following *receiver*:

```
<receiver
android:name="com.spilgames.framework.core.receivers.SpilInstallReferrerReciever"
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:

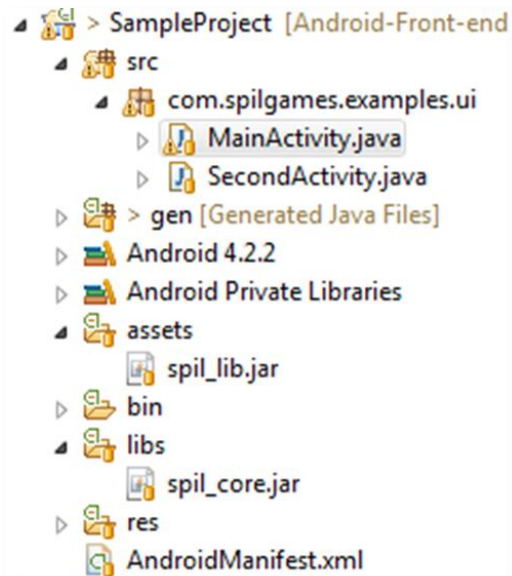
```
<receiver
  android:name="com.fiksu.asotracking.InstallTracking"
  android:exported="true" >
  <intent-filter>
    <action android:name="com.android.vending.INSTALL_REFERRER" />
  </intent-filter>
  <meta-data
    android:name="forward.1"
    android:value="com.spilgames.framework.core.receivers.SpilInstallReferrerReciever"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

- If you have not set them yet, add 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" />
```



The image on the left shows your basic Android project setup:

- The **spil\_lib.jar** file is in the **assets** folder.
- The **spil\_core.jar** file is in the **libs** folder.
- In the **AndroidManifest.xml** file, you rename the application name to **com.spilgames.framework.SpilApplication** and insert the appropriate [receiver](#).

## Parameters

After saving and setting up correctly the necessary libraries, you can initialize the framework using the following method:

```
SpilInterface spilInstance = SpilLink.spilWithAppID(context, appId, authToken, configs);
```

This method requires the following parameters:

- `context`: the current context of the application. We recommend setting it to `getApplicationContext()`.
- `appId` and `authToken`: these two parameters are provided by Spil Games when you get the package.
- `configs`: this parameter is of a `Map<String, String>` type; it needs to include a number of defined keys (see [Allowed keys](#) further on in this document).

After initializing the Spil framework for the first time, you can subsequently call it like this:

```
SpilInterface spilInstance = SpilLink.getInstance();
```

## Listeners

### AdsListener

We recommend implementing the `AdsListener` interface to obtain information about the `Ads` module. Currently the `Ads` module tracks installs in stores.

This is the `AdsListener` interface:

```
public interface AdsListener {  
    public void onAdsLoaded();  
    public void onAdsFailedToLoad(String cause);  
}
```

And this is a basic implementation showing how to set the listener:

```
public class MyActivity extends Activity implements AdsListener  
...  
// Initialization of spilframework  
...  
spilInstance.setAdsListener(this);
```

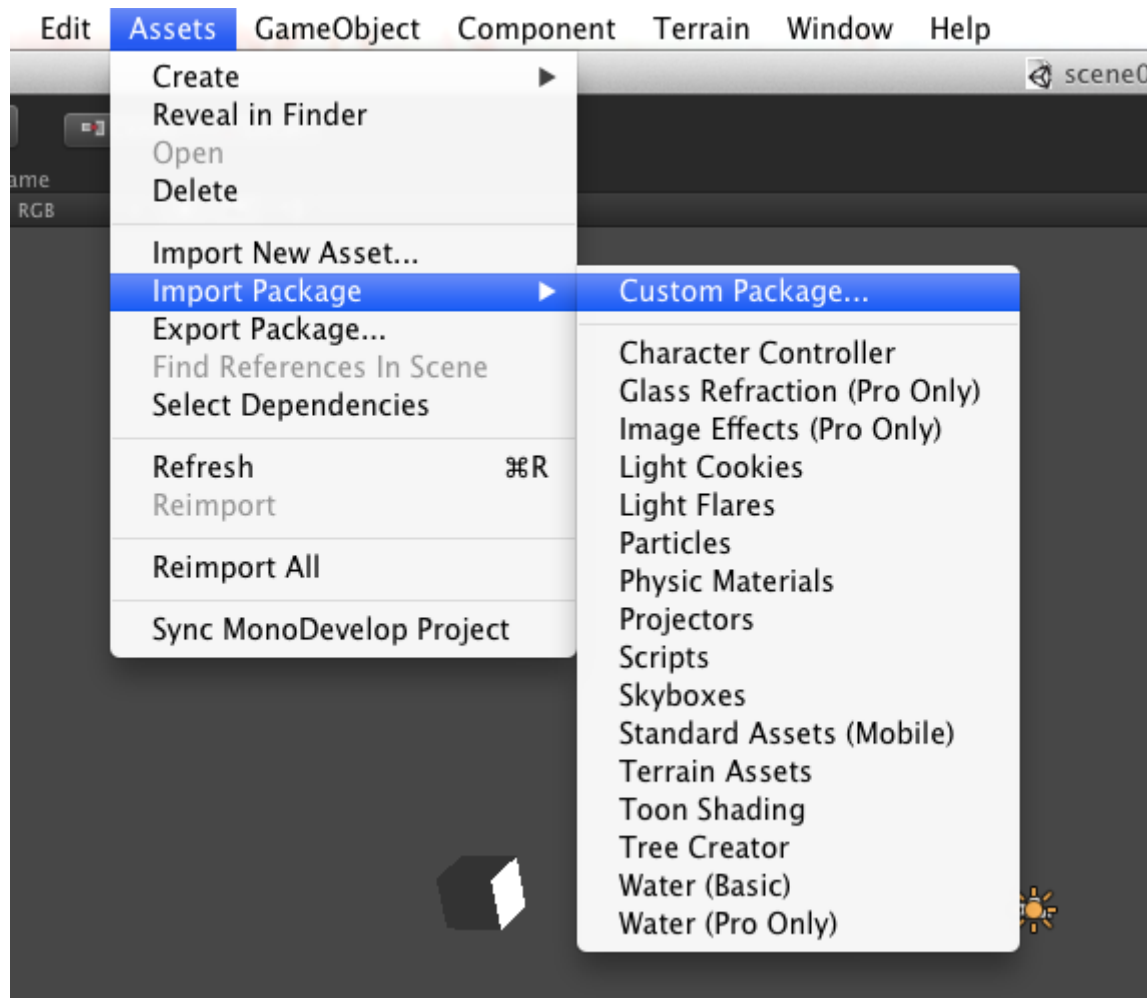
For further details about the `AdsListener` interface, see the [Doxygen documentation](#) available on [GitHub](#).



## Unity-based applications

### 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 [Settings](#) earlier in this document).
- Insert the following activity:

```
<activity android:name="com.spilgames.framework.Spil" />
```

- Insert the following permissions:

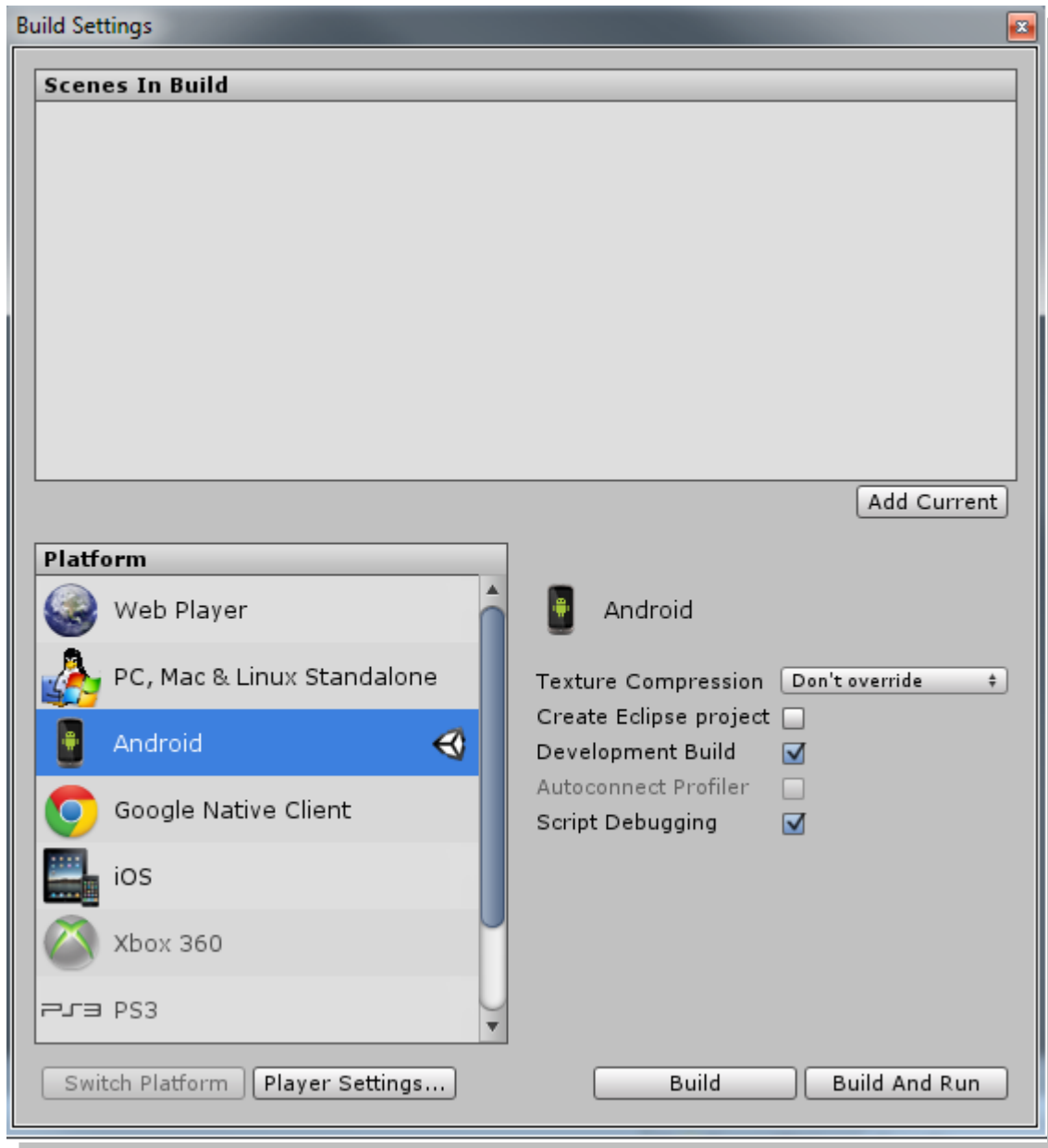
```
<uses-permission android:name="android.permission.INTERNET" />
```

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

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

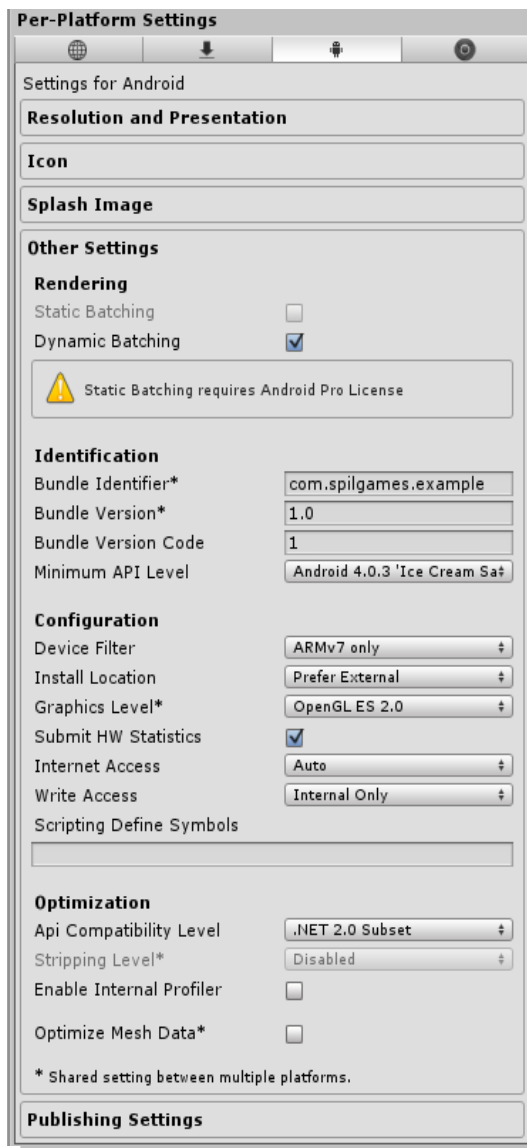
## Generate, build and run the Android project

- 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.



The screenshot shows the 'Per-Platform Settings' dialog for Android. It is divided into several sections: 'Resolution and Presentation', 'Icon', 'Splash Image', 'Other Settings', 'Identification', 'Configuration', 'Optimization', and 'Publishing Settings'. The 'Other Settings' section includes 'Rendering' options: 'Static Batching' (unchecked) and 'Dynamic Batching' (checked). A warning message states 'Static Batching requires Android Pro License'. The 'Identification' section contains fields for 'Bundle Identifier\*' (com.spilgames.example), 'Bundle Version\*' (1.0), 'Bundle Version Code' (1), and 'Minimum API Level' (Android 4.0.3 'Ice Cream Sa'). The 'Configuration' section includes 'Device Filter' (ARMv7 only), 'Install Location' (Prefer External), 'Graphics Level\*' (OpenGL ES 2.0), 'Submit HW Statistics' (checked), 'Internet Access' (Auto), 'Write Access' (Internal Only), and 'Scripting Define Symbols'. The 'Optimization' section includes 'Api Compatibility Level' (.NET 2.0 Subset), 'Stripping Level\*' (Disabled), 'Enable Internal Profiler' (unchecked), and 'Optimize Mesh Data\*' (unchecked). A note at the bottom states '\* Shared setting between multiple platforms.'

**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\* com.spilgames.example

Bundle Version\* 1.0

Bundle Version Code 1

Minimum API Level Android 4.0.3 'Ice Cream Sa'

**Configuration**

Device Filter ARMv7 only

Install Location Prefer External

Graphics Level\* OpenGL ES 2.0

Submit HW Statistics ☒

Internet Access Auto

Write Access Internal Only

Scripting Define Symbols

**Optimization**

Api Compatibility Level .NET 2.0 Subset

Stripping Level\* Disabled

Enable Internal Profiler ☐

Optimize Mesh Data\* ☐

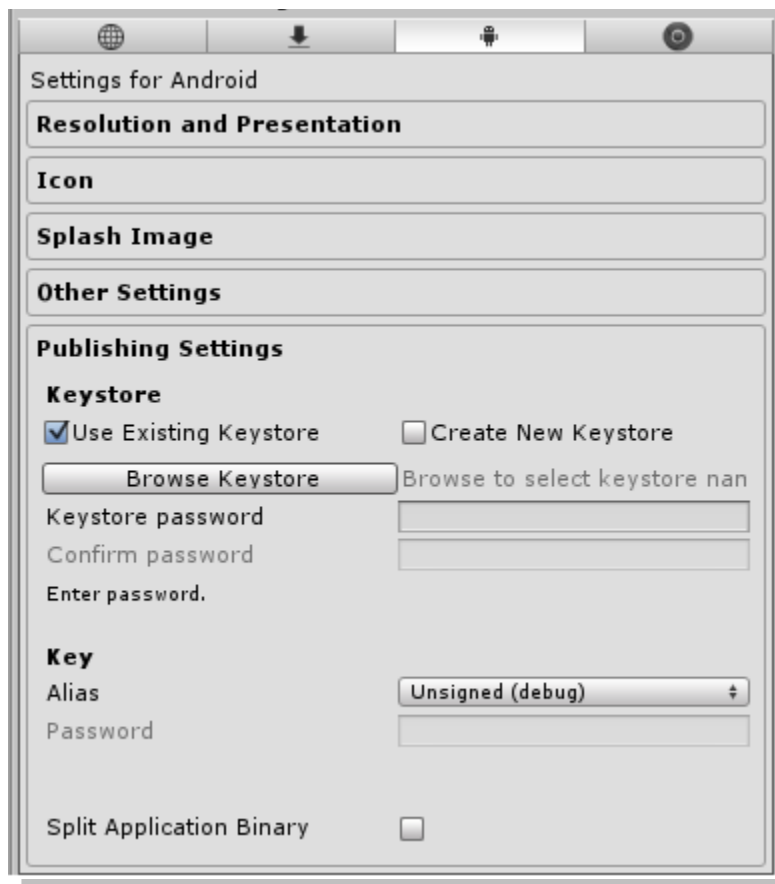
\* Shared setting between multiple platforms.

**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 {
    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.GetAds(this);
    }
}
```

- You need to implement `SpilAdsListener`. You can implement it wherever it is appropriate.
- You need to pass references to the `Spil` object using `GetAds(listenerImplementation)`.

## 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