POWERCORE ACTIVATION PLATFORM

PowerCore SDK Integration Unity (Android/iOS) Tutorial

SDK version 1.0.2
Document Version 11.08.2016

Before You Begin

A. Software Requirements

Unity: 5.0+
Android: Android SDK 14+
iOS: iOS version 8.0+

B. Get a PowerCore Application ID

To integrate the PowerCore SDK into your application, you need to get a PowerCore Application ID for your application - this should have been given along with the SDK package - contact us if you have not been issued one!

C. Plan Out Your Activation Strategy

First, identify any PowerCore-enabled items that you want to integrate with. This will be any toys, cards, and other merchandise that contain PowerCore chips or QR codes. A sample code will be used later in this tutorial, but additional sample codes for testing can be provided if needed.

Second, establish how you want to integrate the PowerCore experience into your application, i.e what value you want to tie into each activation. Will each activation introduce a character or award a rare power-up? How many times should the application accept activations? Only once or once per day?

To get you started, a simple example of how an activation flow may go will be provided below and we're always free to brainstorm with you if you need ideas.

Third, decide how you want to handle the PowerCore activation experience UI wise. The PowerCore SDK offers two different modes to handle the activation - Result Mode and Code Adapter Mode.

> Result Mode will display the default PowerCore activation screens that make it easiest to implement and get started.
> Code Adapter Mode will enable you to dictate how the activation fits with your application's UI as it will do the activation in the background and you can display customized activation screens.

Lastly, familiarize yourself with the information that the SDK will pass back to your application.

These will be discussed in more detail below.

Integrating the PowerCore SDK

1. Install the PowerCoreSDK

The PowerCore SDK can be installed in two ways:

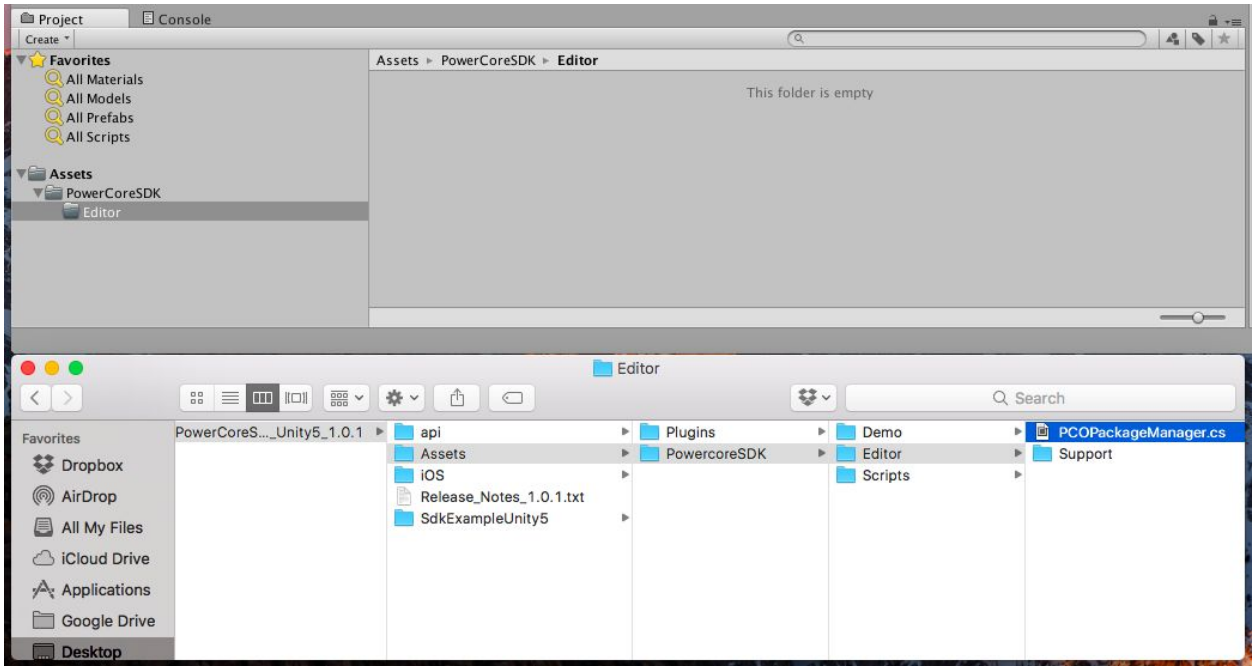
1. Via a script
2. Manually.

Using the script to install the PowerCore SDK is recommended to avoid problems with missed files or settings. The script will also automatically clear any files or settings from old PowerCore SDK versions, and can be used to easily update SDK files if any changes to settings such as 'PowerCore Application ID' or 'Bundle Identifier' are made.

If in any case you are unable to use the script, the steps to manually install the PowerCore SDK are also provided below.

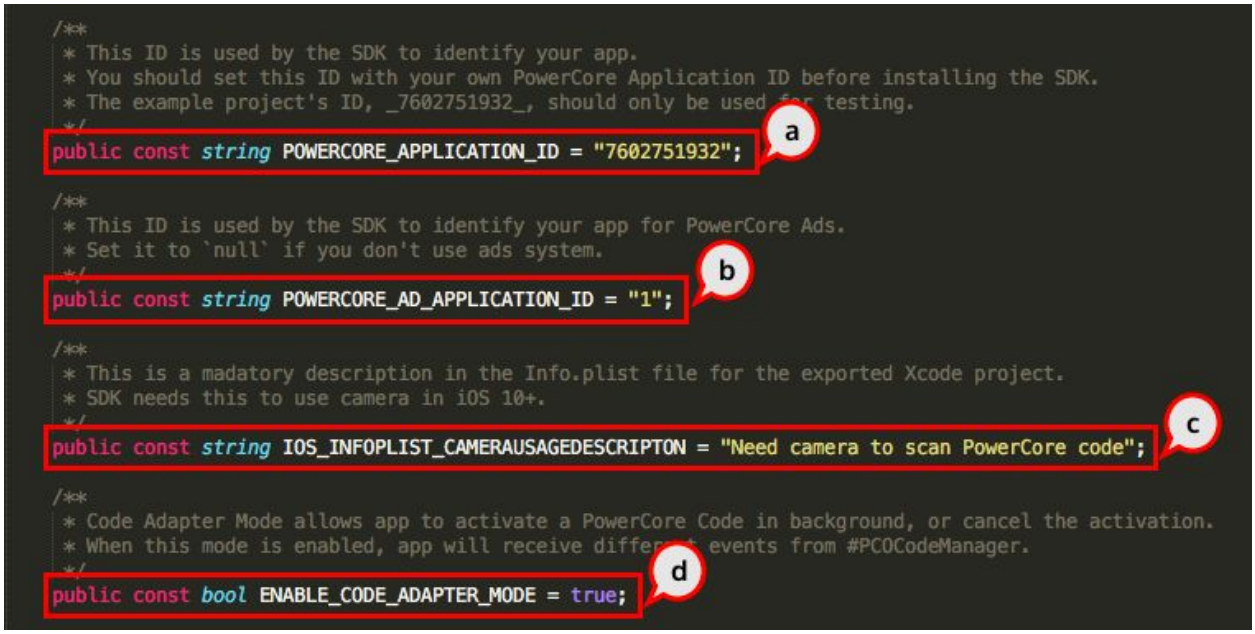
Recommended Method:
Script Installation via PCOPackageManager.cs

1. **Unzip the PowerCore SDK package.**
2. **Copy the PCOPackageManager.cs script to your project's Assets folder.**
    a. The script is located inside the PowerCore SDK package at /PowerCoreSDK_ Unity5_1.0.1/Assets/PowercoreSDK/Editor/
    b. Copy the script file to your Unity project at /<YOUR_UNITY_PROJECT_FOLDER>/Assets/PowerCoreSDK/Editor /



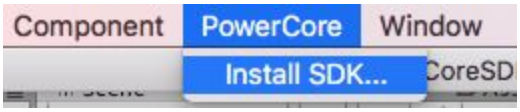
c. After copying the file, a new Menu item labeled PowerCore will appear.

3. **Before you run the script, a couple of settings need to be manually edited within the script to customize it for your application:**

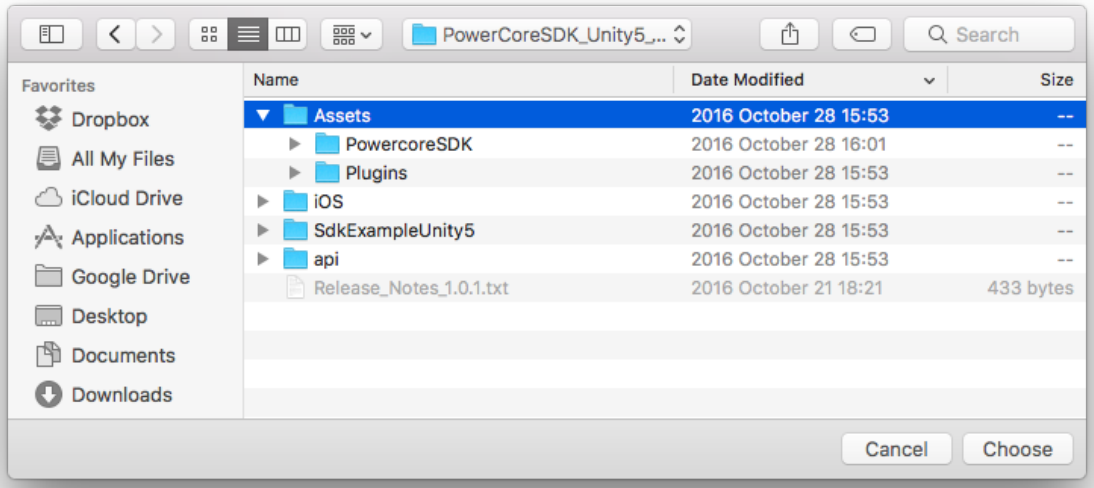


   a. **POWERCORE_APPLICATION_ID:** the application ID provided by PowerCore; this will have a default value of '**7602751932**', but will need to be updated to have your application correctly work with PowerCore codes.
   b. **POWERCORE_AD_APPLICATION_ID:** the ad application ID provided by PowerCore; this will have a default value of '**1**', but will need to be updated to have your application correctly work with the PowerCore Ads System. If case your application does not use the PowerCore Ads System, you need to set this value to **NULL**.
   c. **IOS_INFOPLIST_CAMERAUSAGEDESCRIPTION:** the PowerCore SDK will require the use of the phone's camera and will be displaying this string when asking for camera permission in iOS applications; this will have a default value of "**Need camera to scan PowerCore code**".
   d. **ENABLE_CODE_ADAPTER_MODE:** this will set whether the PowerCore SDK should use the Code Adapter Mode mentioned above and detailed below; this will have a default value of **TRUE**.

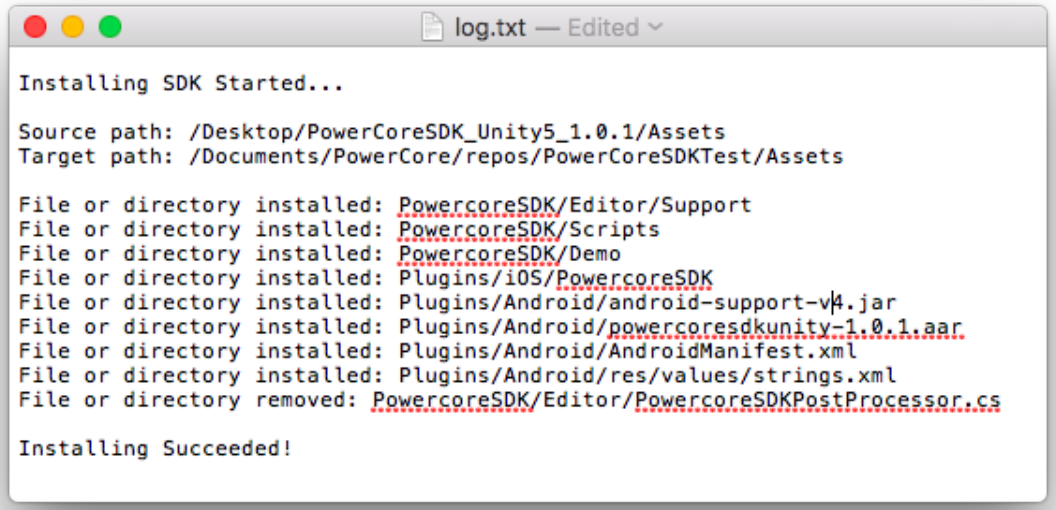
4. **Run the script via the Unity Menu item**
   a. PowerCore > Install SDK...



   b. Select the location of the Assets folder inside the unzipped PowerCore SDK package

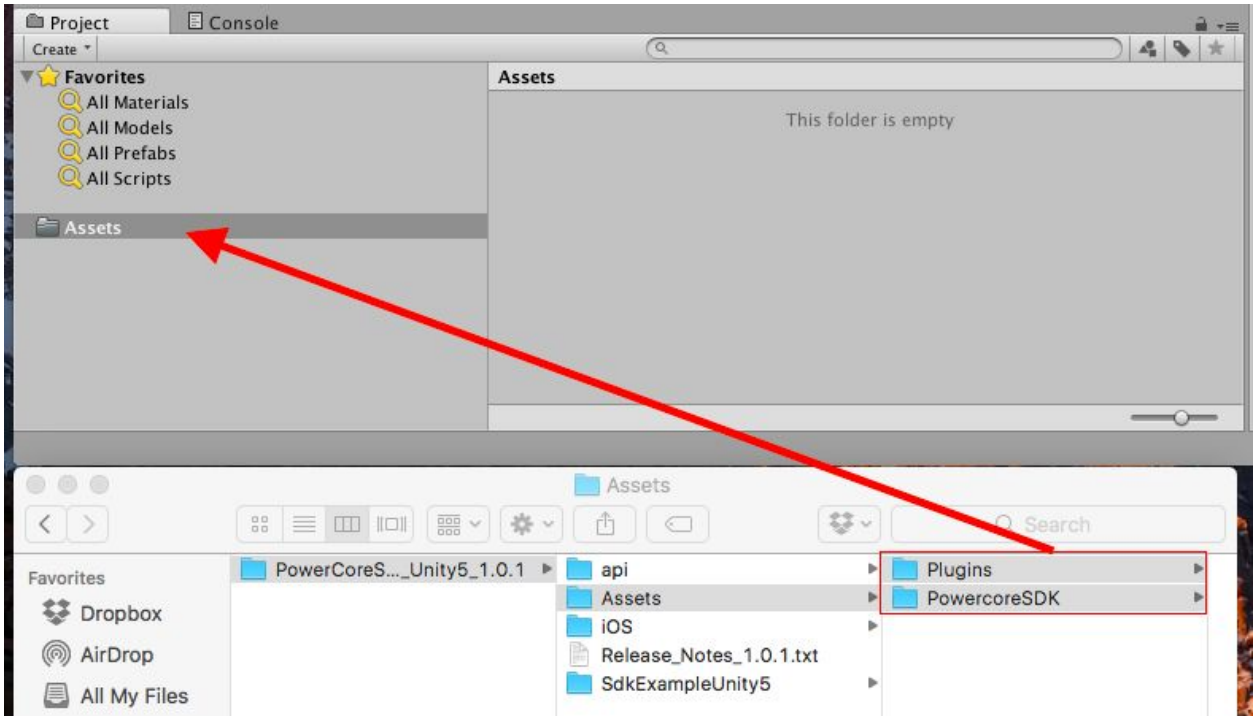


5. **Verify if the script ran successfully**
    a. Check if there are any errors in the logfile at '<YOUR_UNITY_PROJECT_FOLDER>/PowerCore/SDKInstallation/<EXECUTION_TIME>'. If there are no errors, then "Installing Succeeded!" should be the last line of the log file.

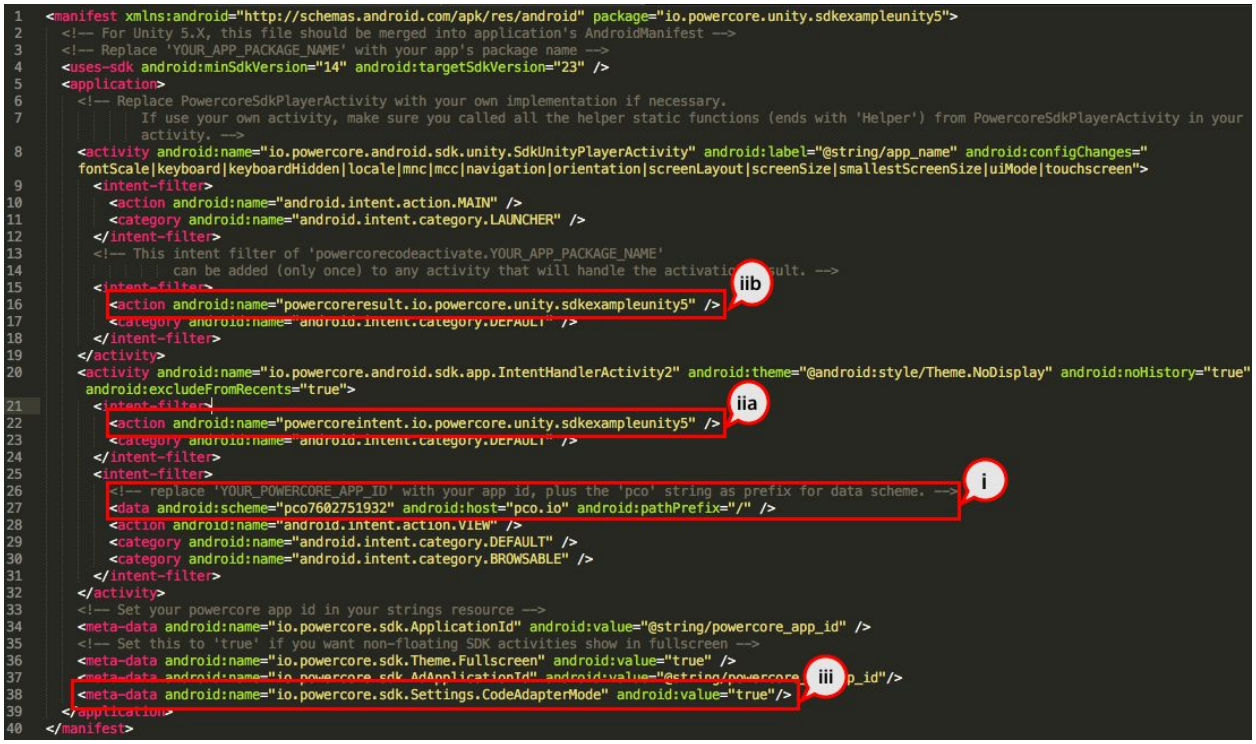


Manual Installation

1. Unzip the PowerCore SDK package.
2. Copy the **Plugins** and **PowerCoreSDK** folders to your project's **Assets** folder. Both folders are located inside /PowerCoreSDK_Unity5_1.0.1/Assets/



3. Update the values for the following items in AndroidManifest.xml



i. Update the scheme for data entry with host pco.io
   a. '**pco**' + YOUR_APPLICATION_POWERCORE_APP_ID
ii. Update **powercoreresult** and **powercoreintent** to your application's package name
   a. **powercoreresult**.YOUR_APPLICATION_PACKAGE_NAME
   b. **powercoreintent**.YOUR_APPLICATION_PACKAGE_NAME
iii. Set the value of the 'io.powercore.sdk.Settings.CodeAdapterMode' meta-data entry to TRUE or FALSE depending on whether your application will be using Result Mode (FALSE) or Code Adapter Mode (TRUE)

4. Add entries for powercore_app_id and powercore_ad_app_id in strings.xml
   a. powercore_app_id
      <string name="powercore_app_id">
          YOUR_POWERCORE_APPLICATION_ID
      </string>
   b. powercore_ad_app_id - set to NULL if you are not using the PowerCore AD platform
      <string name="powercore_app_id">
          YOUR_POWERCORE_AD_APPLICATION_ID
      </string>

2. Modify Settings For Android

The previous section will have added an 'intent-filter' element with an 'action' of 'powercoreresult.YOUR_APPLICATION_PACKAGE_NAME' to your main activity declaration in 'AndroidManifest.xml'.

You can move this 'intent-filter' to any activity in your application which will handle the Code Activation intent.

```
<!-- This intent filter of 'powercoreresult.YOUR_APP_PACKAGE_NAME'
        can be added (only once) to any activity that will handle the activation result. -->
<intent-filter>
  <action android:name="powercoreresult.io.powercore.unity.sdkexampleunity5" />
  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

The PowerCore SDK provides a subclass of '**UnityPlayerActivity**' named **'SdkUnityPlayerActivity'** where you can build your code on, but in case your application already contains a **'UnityPlayerActivity'** subclass, the following helper methods should be called from that subclass' corresponding methods.

UnityPlayerActivity	PowerCore SDK helper method
onCreate	SdkManagerUnity.playerActivityOnCreate(this, savedInstanceState);
onResume	SdkManagerUnity.playerActivityOnResume(this);
onNewIntent	SdkManagerUnity.playerActvityOnNewIntent(this, intent);

```
   @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        SdkManagerUnity.playerActivityOnCreate(this,
savedInstanceState);
        ...
    }
    @Override
    protected void onResume() {
        super.onResume();
```

```
        SdkManagerUnity.playerActivityOnResume(this);
        ...
    }
    @Override
    protected void onNewIntent (Intent intent) {
        super.onNewIntent(intent);
        SdkManagerUnity.playerActivityOnNewIntent(this, intent);
        ...
    }
```

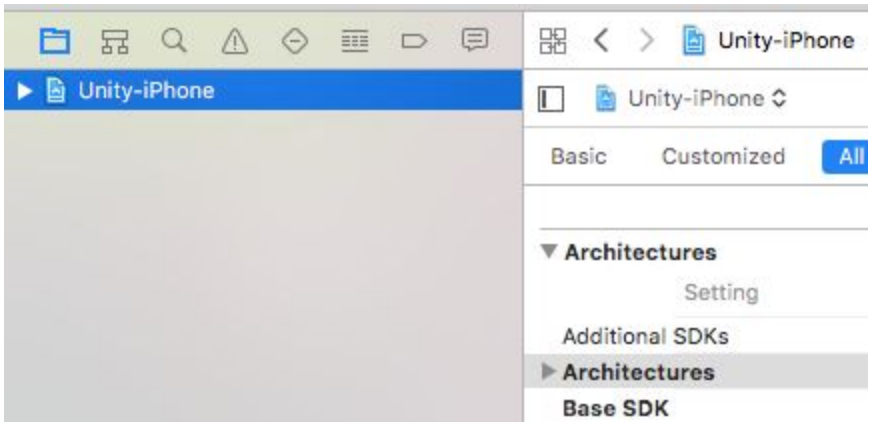
3. Modify Settings For iOS

The PowerCore SDK package contains two different binaries for iOS development

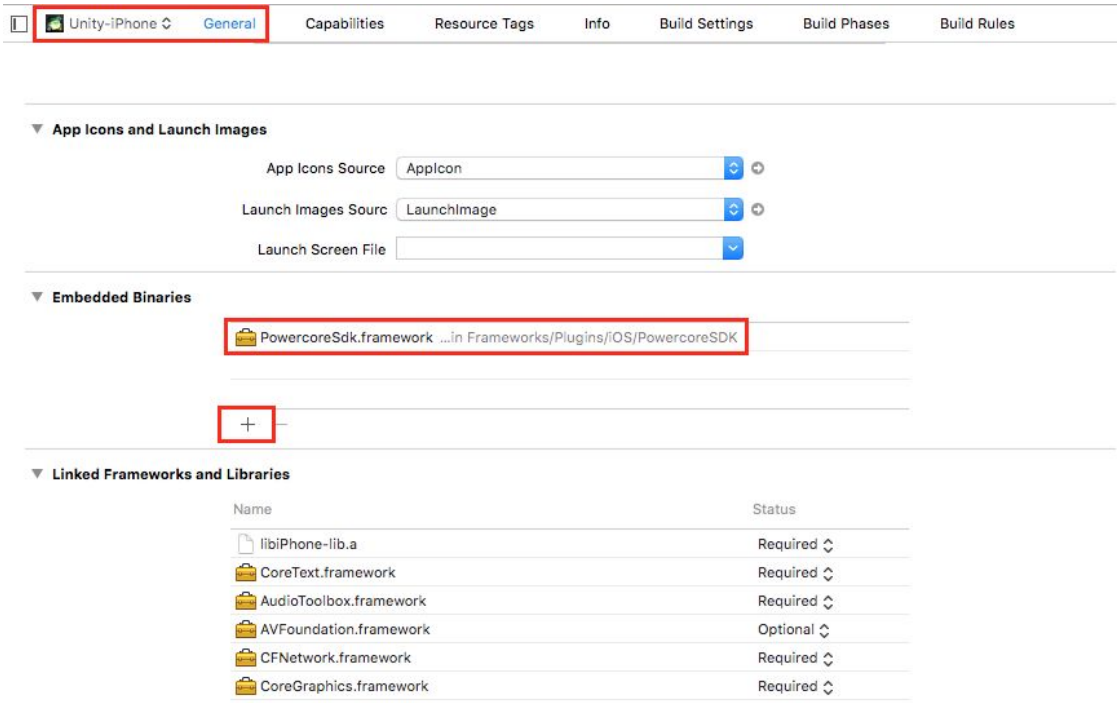
1. **PowercoreSdk-Release-universal** - this binary supports both the simulator and devices; **but cannot be used to submit to the App Store.**
2. **PowercoreSdk-Release-iphoneos** - this binary can be used to submit to the App Store; **but will not run on the simulator.**

To use either of the two binaries in XCode:

1. In **Project Navigator** > **Editor** > **Targets**, select your project and the build target which uses the PowerCore SDK.



2. In the "**General**" tab, "**Embedded Binaries**" section, click the (+) sign and choose which binary you want to use. This will add the framework to the "**Linked Frameworks**" and "**Libraries**" section as well as the content list in "**Project Navigator**".



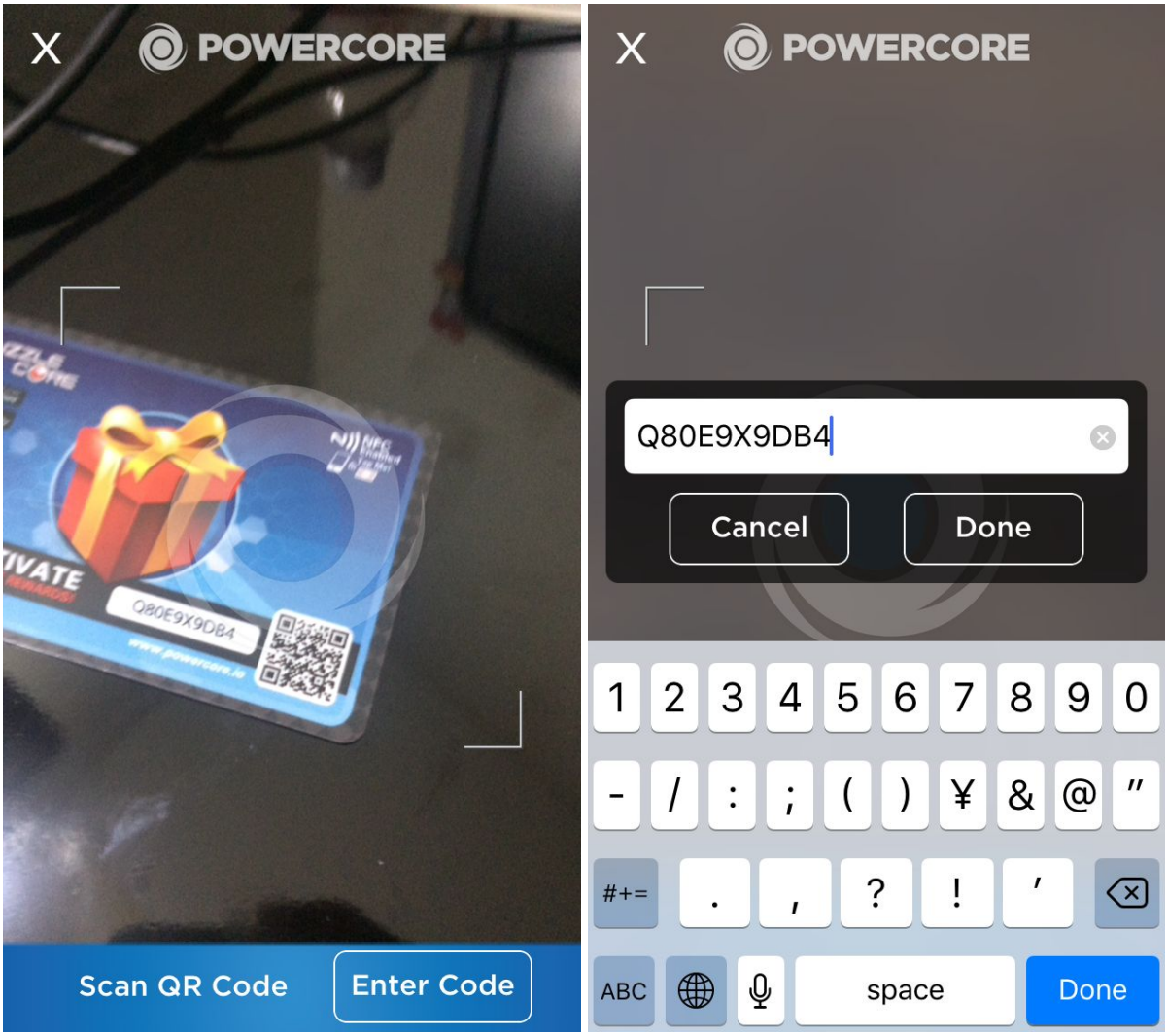
4. Handle the PowerCore Activation Experience

The PowerCore SDK activation experience consists of two portions:
Entering a code and
Displaying the result for the code entered.

The PowerCore SDK currently handles code input via NFC (Android), QR Codes, and Manual Entry. The PowerCore SDK will detect NFC codes inside any Activity of your Android application automatically. For QR Codes and Manual Entry in Android and iOS, you will need to call the Code Scanner manually.

```
// Call PowerCore scanner
PCOCodeManager.Instance.StartCodeScanner();
```

The above call should prompt the PowerCore scanner to display - where the users can scan QR codes or input them manually.



To handle the result for the entered code, you can choose between two different modes: **Result Mode** and **Code Adapter Mode**.

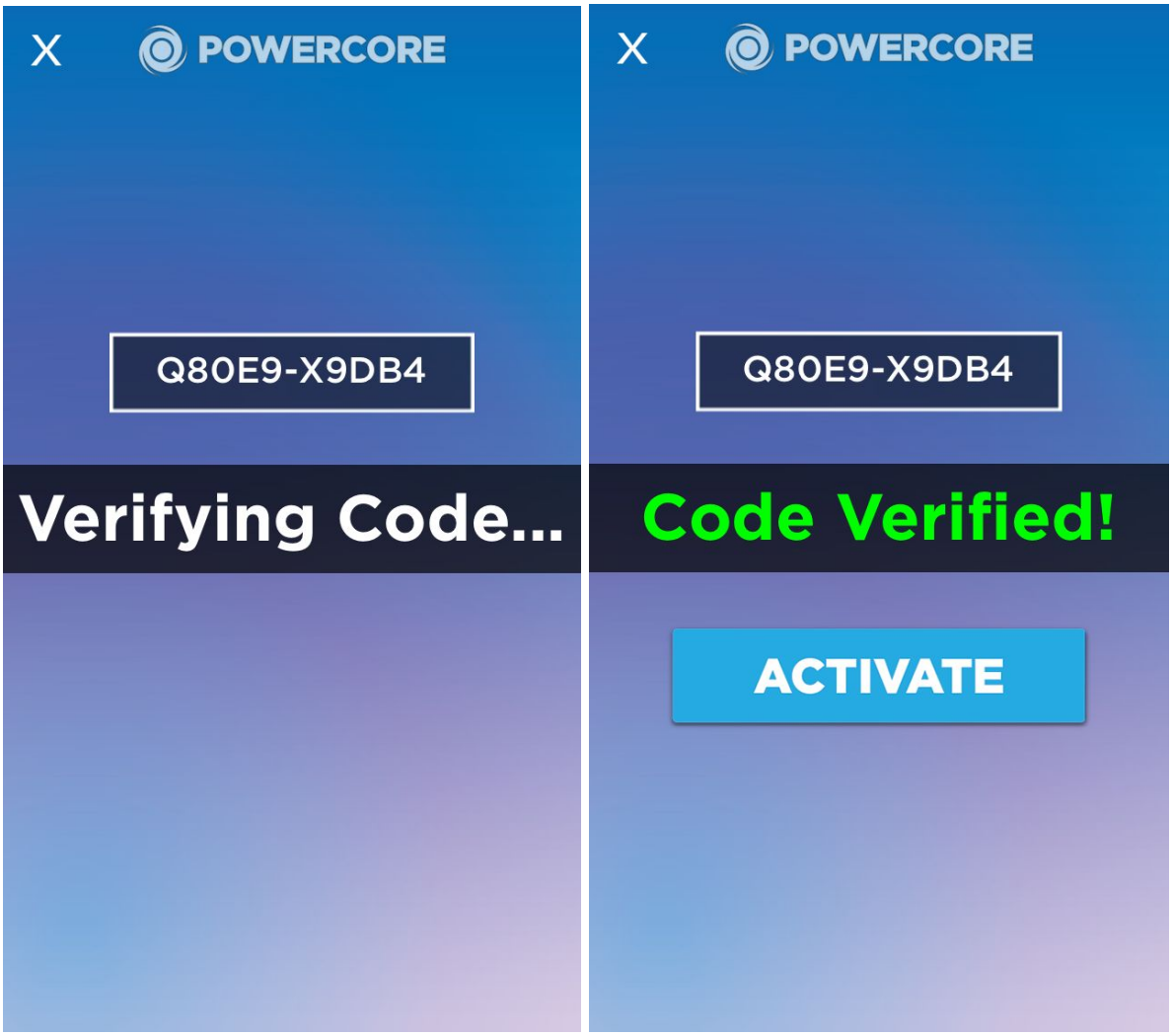
If you used the PCOPackageManager.cs script, you can switch between the two modes via the variable `ENABLE_CODE_ADAPTER_MODE`:

```
// Enable Code Adapter Mode in  PCOPackageManager.cs
public const bool ENABLE_CODE_ADAPTER_MODE =  true;
```

With **Result Mode**, your application will only need one event handler to receive the activation result, and there will be no need to define a custom UI to be displayed during the activation process, as the default PowerCore screens will appear.

```
// Result Mode required handlers
if (!PCOCodeManager.Instance.IsCodeAdapterModeEnabled())
{
 PCOCodeManager.Instance.OnSdkResultReturned +=
    OnSdkResultReturned;
}
```

Below are the default PowerCore screens that will display when using **Result Mode**.



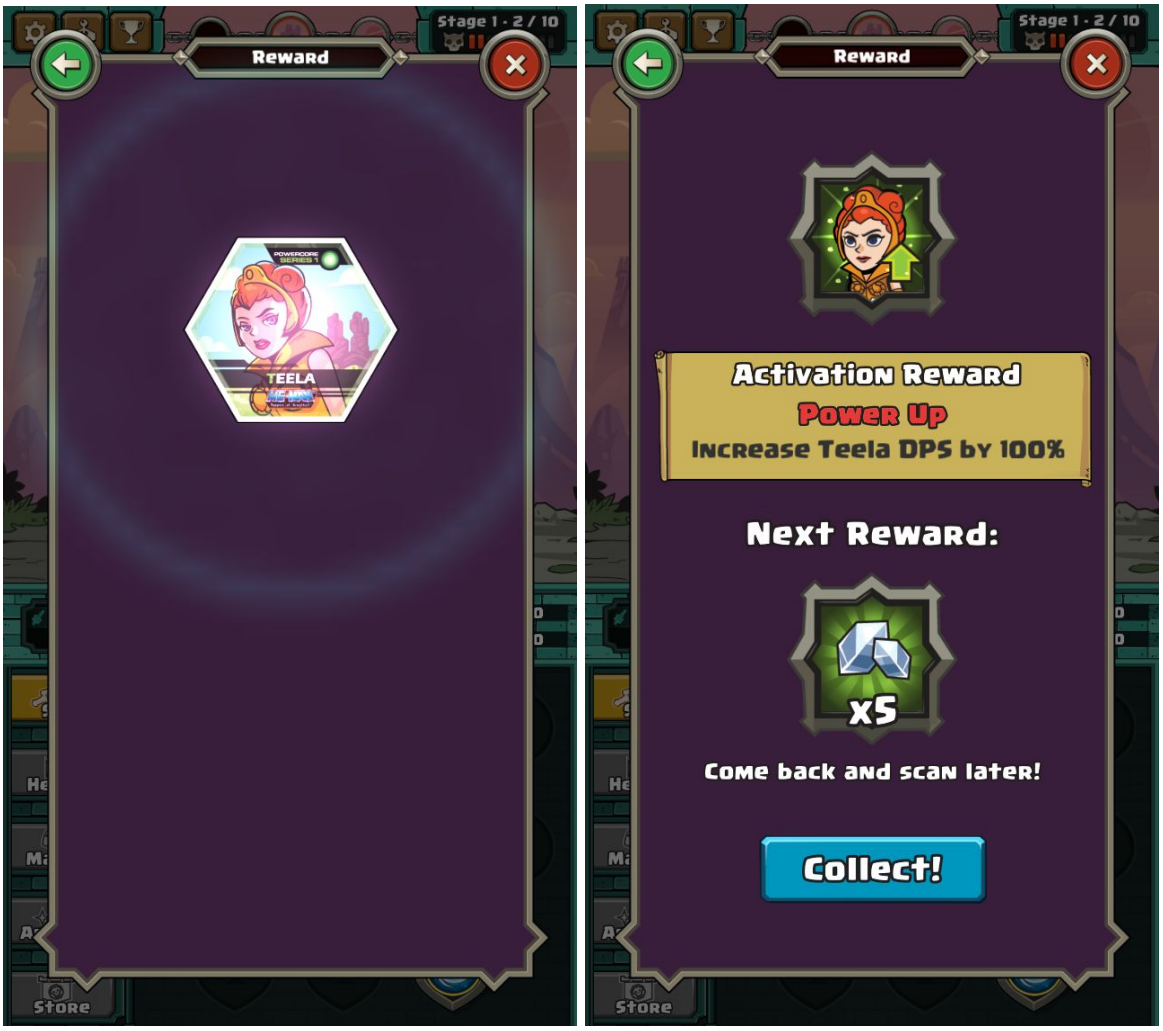
With **Code Adapter Mode**, your application will need to define a handler for the different phases of the activation process, as well as a custom UI to be displayed during the activation process.

```
// Code Adapter Mode required handlers
if (PCOCodeManager.Instance.IsCodeAdapterModeEnabled())
{
    PCOCodeManager.Instance.OnCodeDetected +=
         OnCodeDetected;
    PCOCodeManager.Instance.OnCodeActivationStarting +=
         OnCodeActivationStarting;
    PCOCodeManager.Instance.OnCodeActivationEnded +=
         OnCodeActivationEnded;
}
```

Handler	Description
OnCodeDetected	Triggered when an NFC is detected
OnCodeActivationStarting	Triggered on QR code scan or Done is selected in the Camera/Manual Entry screen
OnCodeActivationEnded	Triggered when the verification process for the code ends

When using the **Code Adapter Mode**, you can also choose to ignore NFC scans inside the OnCodeDetected handler.

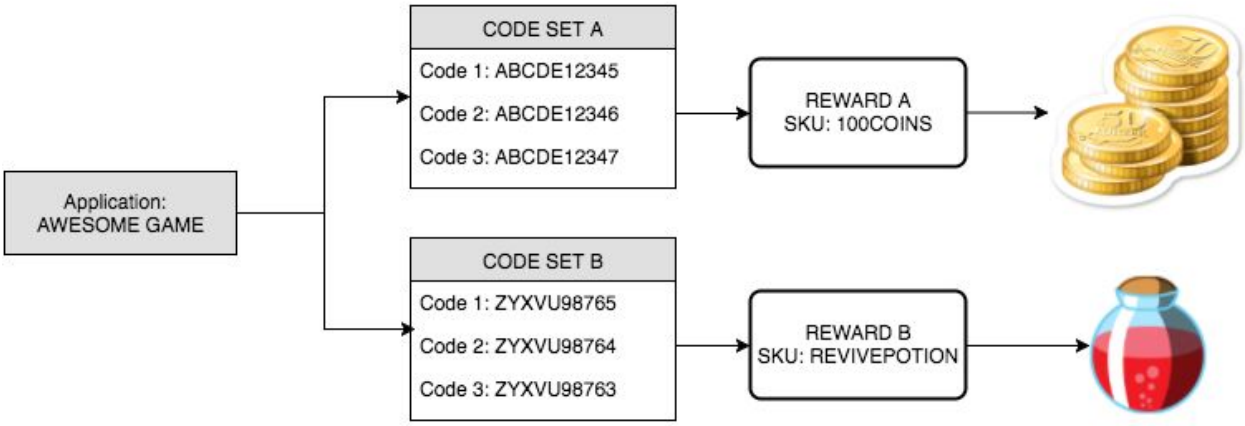
Below is a sample of how a customized UI would look when using **Code Adapter Mode**.



For both **Result** and **Code Adapter** modes, adding the event handlers as early as possible in your code is recommended so that your application will not miss any codes that are scanned via the PowerCore SDK.

5. Read Activation Data

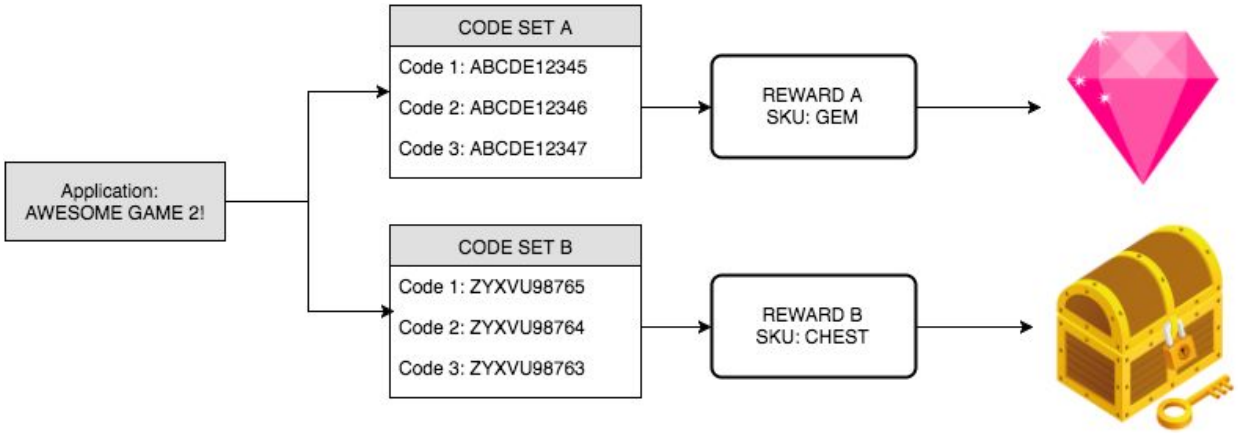
The PowerCore SDK is used to validate against the PowerCore Server whether an activation is valid or not, and will return based on the application a set of information on what the activation should enable.



In the diagram above, we have an application named AWESOME GAME and for this application, we have allocated two different code sets - CODE SET A and CODE SET B to correspond to two different rewards.

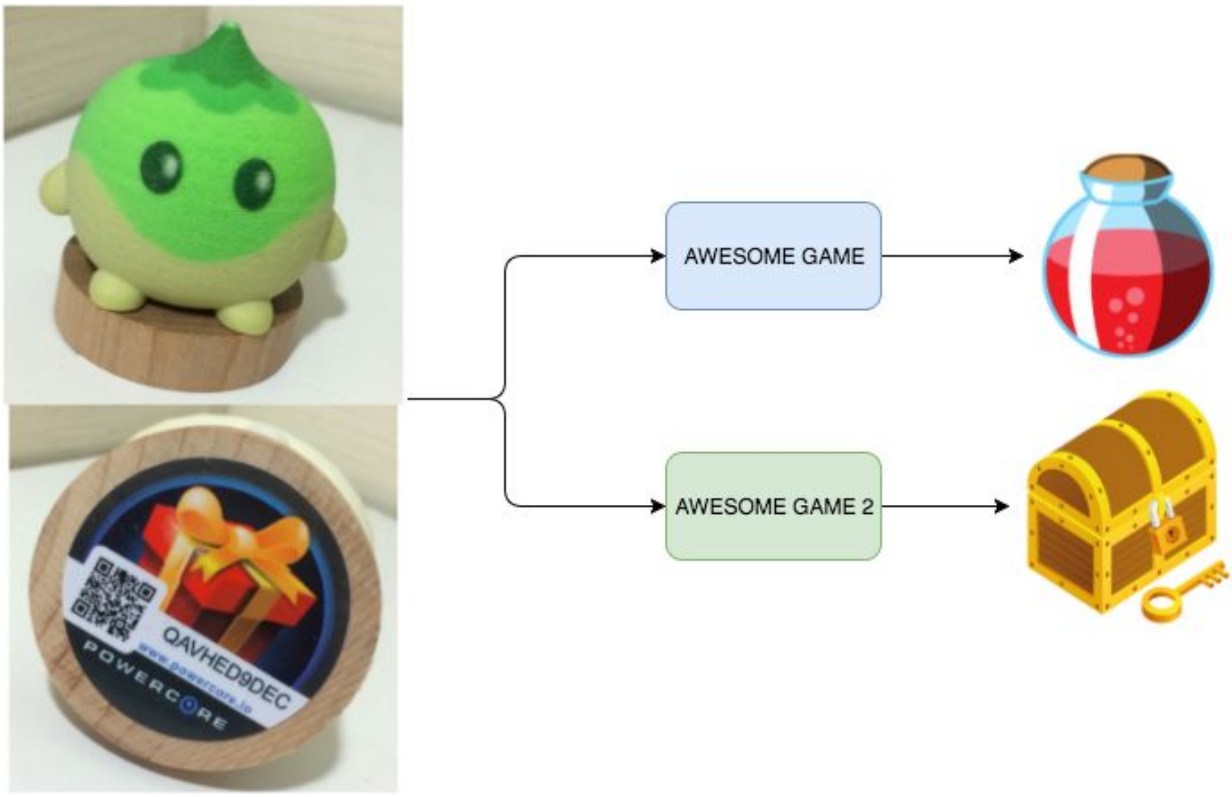
This means that whenever a code is scanned from CODE SET A, it will return data for AWESOME GAME, in this case Reward A with a SKU of 100COINS. Similarly, if a code is scanned from CODE SET B, it will return data for Reward B, which in this case has a SKU of REVIVEPOTION.

In case you have multiple applications that use the PowerCore SDK, the PowerCore System can be set up so that the same CODE SETS activate a different batch of rewards across your applications



In the diagram above, we now have an application named AWESOME GAME 2 and for this application, we are using the same code sets from AWESOME GAME - CODE SET A and CODE SET B but attaching it to different rewards. This time they are tied to a GEM (CODE SET A) and a CHEST (CODE SET B).

This means that whenever a code is scanned from CODE SET A, it will return data for AWESOME GAME 2 - in this case Reward A, but this time with a SKU of GEM. Similarly, if a code is scanned from CODE SET B, it will return data for Reward B, which in this case has a SKU of CHEST.



Below is an example of the result string after a code from CODE SET B is scanned inside AWESOME GAME and it is a valid activation. As mentioned above, this result string will be available via

(a) PCOCodeResultEventArgs.Result in Result Mode

(b) PCOCodeAdapterEventArgs.Code.ActivationResult in Code Adapter Mode


```
{
   "result":"succeeded",
   "results_array":[
```

```
      {
         "application":"Awesome Game",
         "reward":"Revive Potion",
         "reward_type":"Digital",
         "sku":"REVIVEPOTION",
         "points":0
      }
   ]
}
```

Field Name	Description
result	(string) result of code validation request; this can have a value of (a) succeeded (b) failed (c) cancelled
application	(string) name of application requesting the code activation
reward	(string) name of reward connected to the activated code
reward_type	(string) type of reward connected to the activated code; the reward_type can be Physical or Digital
sku	(string) the sku (stock keeping unit) value for the activated code; this will be unique for each set of codes generated for your application
points	(int) number of points connected to the activated code

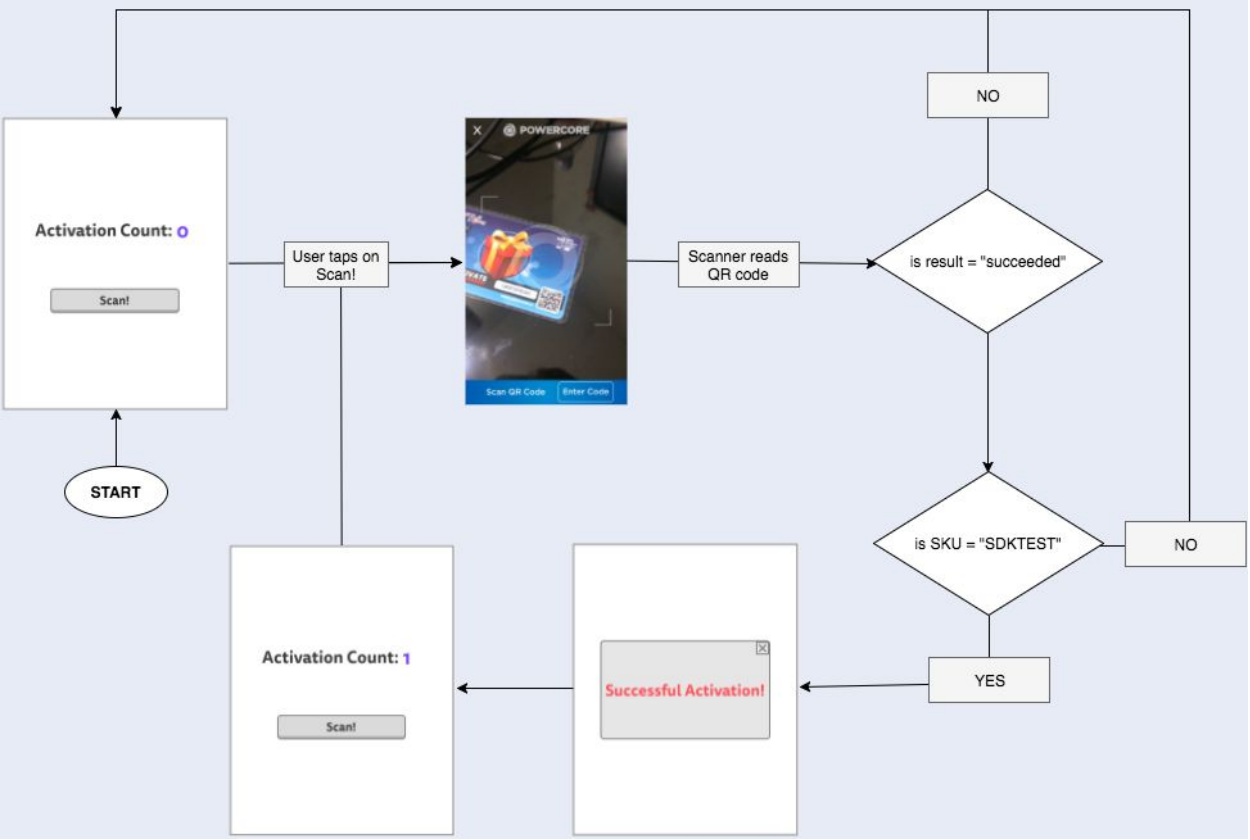
So in the sample result above, the PowerCore SDK has successfully activated a code for the application named Awesome Game. This code is tied to a reward named Revive Potion. This reward has SKU value of REVIVEPOTION. The application can then use this SKU inside the game to award the user a Revive Potion.

Next Steps - Using the PowerCore SDK

Now that you've integrated the PowerCore SDK into your project, you can now start working with PowerCore codes!

The following section layouts a very simple activation flow which you can build on. This can also be used to verify if you've successfully integrated the PowerCore SDK into your application.

For this flow we will be using the Result Mode which will display the default PowerCore activation screens. We will also only be using the SKU to dictate what the application will do after successfully activating a code.

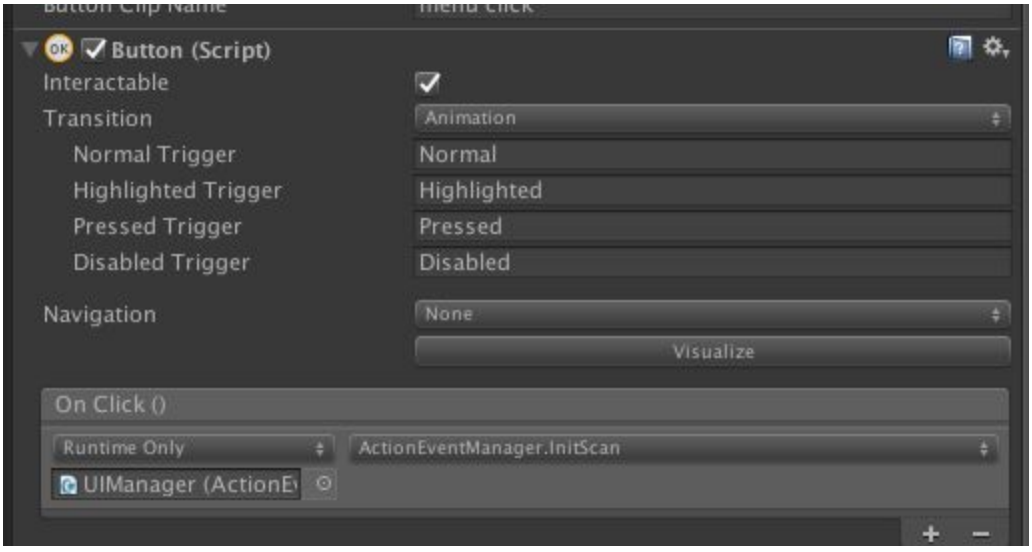


In this sample flow, our main screen will be displaying the number of times you have activated a code, as well as contain a button to call the SDK scanner so you can scan QR codes or enter codes manually.

The SKU value we will compare with has a value of **SDKTEST**. This SKU will be unique to a specific set of codes inside the PowerCore code system, meaning that other codes will be tied to a different SKU value.

For the button, we just need to tie it to a method which will call the PowerCoreSDK method `PCOCodeManager.Instance .StartCodeScanner();`

A sample implementation is below - here we have ActionEventManager which has a method named InitScan(), which we have tied to the **Scan!** Button.



And then inside **InitScan()**, we have the following code:

```
public void InitScan()
{
```

```
      PowercoreSDKCodeManager.instance.StartCodeScanner();
    }
```

This should bring up the PowerCore code scanner on click of the button.

For the results of the scan, as we are using the ResultMode , we need to set a delegate to handle it - in this case, we're setting it to a method named **CheckPowerCoreResult.**

```
PowercoreSDKCodeManager.instance.OnPowercoreResultDelegate +=
CheckPowerCoreResult;
```

Inside the delegate method, we can access the **_pcoResult** object detailed in section 5 above, and we can read and check against the values in that JSON

```
public void CheckPowerCoreResult(string _pcoResult) {
    // parse JSON and check the value for SKU
    // you may use your preferred JSON library
    // below is just pseudo code

    if(parsedJSON.getValueFor("SKU") == "SDKTEST") {
      activationCount++;
      // other specific actions for the SKU
      // introduce a new character
      // give out coins
    }

    updateUI();
}
```

Based on how you designed your activation rewards, you can customize what is inside the **CheckPowerCoreResult** method to how you want to reward each activation, may it be a new character or redeem coins inside your application.

Try this flow out!

Set the PowerCore application ID as '**7602751932**' and then try entering the code **'RXY6NA9DC4'**. The PowerCore SDK should return a successful result with the SKU value being returned is **'SDKTEST'.**

The following QR code can also be scanned and should return a successful result with the same SKU value of '**SDKTEST**'.

RXY6NA9DC4

The PowerCore SDK package also contains a sample project which demonstrates the PowerCore activation process which you can base and build on.

Troubleshooting the PowerCore SDK

If you have any questions or problems with compiling or running your project after integrating the PowerCore SDK, please reach out to us at support@powercore.io.