using System;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Purchasing;

// Placing the Purchaser class in the CompleteProject namespace allows it to interact with ScoreManager,

// one of the existing Survival Shooter scripts.

namespace CompleteProject

{

 // Deriving the Purchaser class from IStoreListener enables it to receive messages from Unity Purchasing.

 public class Purchaser : MonoBehaviour, IStoreListener

 {

 private static IStoreController m\_StoreController; // The Unity Purchasing system.

 private static IExtensionProvider m\_StoreExtensionProvider; // The store-specific Purchasing subsystems.

 // Product identifiers for all products capable of being purchased:

 // "convenience" general identifiers for use with Purchasing, and their store-specific identifier

 // counterparts for use with and outside of Unity Purchasing. Define store-specific identifiers

 // also on each platform's publisher dashboard (iTunes Connect, Google Play Developer Console, etc.)

 // General product identifiers for the consumable, non-consumable, and subscription products.

 // Use these handles in the code to reference which product to purchase. Also use these values

 // when defining the Product Identifiers on the store. Except, for illustration purposes, the

 // kProductIDSubscription - it has custom Apple and Google identifiers. We declare their store-

 // specific mapping to Unity Purchasing's AddProduct, below.

 public static string PRODUCT\_Ad\_Free = "Duck\_Ad\_Free";

 // Apple App Store-specific product identifier for the subscription product.

 //private static string kProductNameAppleSubscription = "com.unity3d.subscription.new";

 // Google Play Store-specific product identifier subscription product.

 //private static string kProductNameGooglePlaySubscription = "com.unity3d.subscription.original";

 void Start()

 {

 // If we haven't set up the Unity Purchasing reference

 if (m\_StoreController == null)

 {

 // Begin to configure our connection to Purchasing

 InitializePurchasing();

 }

 }

 public void InitializePurchasing()

 {

 // If we have already connected to Purchasing ...

 if (IsInitialized())

 {

 // ... we are done here.

 return;

 }

 // Create a builder, first passing in a suite of Unity provided stores.

 var builder = ConfigurationBuilder.Instance(StandardPurchasingModule.Instance());

 builder.AddProduct(PRODUCT\_Ad\_Free, ProductType.NonConsumable);

 // Kick off the remainder of the set-up with an asynchrounous call, passing the configuration

 // and this class' instance. Expect a response either in OnInitialized or OnInitializeFailed.

 UnityPurchasing.Initialize(this, builder);

 }

 private bool IsInitialized()

 {

 // Only say we are initialized if both the Purchasing references are set.

 return m\_StoreController != null && m\_StoreExtensionProvider != null;

 }

 public void BuyAdFree()

 {

 // Buy the non-consumable product using its general identifier. Expect a response either

 // through ProcessPurchase or OnPurchaseFailed asynchronously.

 BuyProductID(PRODUCT\_Ad\_Free);

 }

 void BuyProductID(string productId)

 {

 // If Purchasing has been initialized ...

 if (IsInitialized())

 {

 // ... look up the Product reference with the general product identifier and the Purchasing

 // system's products collection.

 Product product = m\_StoreController.products.WithID(productId);

 // If the look up found a product for this device's store and that product is ready to be sold ...

 if (product != null && product.availableToPurchase)

 {

 Debug.Log(string.Format("Purchasing product asychronously: '{0}'", product.definition.id));

 // ... buy the product. Expect a response either through ProcessPurchase or OnPurchaseFailed

 // asynchronously.

 m\_StoreController.InitiatePurchase(product);

 }

 // Otherwise ...

 else

 {

 // ... report the product look-up failure situation

 Debug.Log("BuyProductID: FAIL. Not purchasing product, either is not found or is not available for purchase");

 }

 }

 // Otherwise ...

 else

 {

 // ... report the fact Purchasing has not succeeded initializing yet. Consider waiting longer or

 // retrying initiailization.

 Debug.Log("BuyProductID FAIL. Not initialized.");

 }

 }

 // Restore purchases previously made by this customer. Some platforms automatically restore purchases, like Google.

 // Apple currently requires explicit purchase restoration for IAP, conditionally displaying a password prompt.

 public void RestorePurchases()

 {

 // If Purchasing has not yet been set up ...

 if (!IsInitialized())

 {

 // ... report the situation and stop restoring. Consider either waiting longer, or retrying initialization.

 Debug.Log("RestorePurchases FAIL. Not initialized.");

 return;

 }

 // If we are running on an Apple device ...

 if (Application.platform == RuntimePlatform.IPhonePlayer ||

 Application.platform == RuntimePlatform.OSXPlayer)

 {

 // ... begin restoring purchases

 Debug.Log("RestorePurchases started ...");

 // Fetch the Apple store-specific subsystem.

 var apple = m\_StoreExtensionProvider.GetExtension<IAppleExtensions>();

 // Begin the asynchronous process of restoring purchases. Expect a confirmation response in

 // the Action<bool> below, and ProcessPurchase if there are previously purchased products to restore.

 apple.RestoreTransactions((result) => {

 // The first phase of restoration. If no more responses are received on ProcessPurchase then

 // no purchases are available to be restored.

 Debug.Log("RestorePurchases continuing: " + result + ". If no further messages, no purchases available to restore.");

 });

 GameControl.instance.RestoreButton.image.sprite = GameControl.instance.OnSprite;

 }

 // Otherwise ...

 else

 {

 // We are not running on an Apple device. No work is necessary to restore purchases.

 Debug.Log("RestorePurchases FAIL. Not supported on this platform. Current = " + Application.platform);

 }

 }

 //

 // --- IStoreListener

 //

 public void OnInitialized(IStoreController controller, IExtensionProvider extensions)

 {

 // Purchasing has succeeded initializing. Collect our Purchasing references.

 Debug.Log("OnInitialized: PASS");

 // Overall Purchasing system, configured with products for this application.

 m\_StoreController = controller;

 // Store specific subsystem, for accessing device-specific store features.

 m\_StoreExtensionProvider = extensions;

 }

 public void OnInitializeFailed(InitializationFailureReason error)

 {

 // Purchasing set-up has not succeeded. Check error for reason. Consider sharing this reason with the user.

 Debug.Log("OnInitializeFailed InitializationFailureReason:" + error);

 }

 public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)

 {

 if (String.Equals(args.purchasedProduct.definition.id, PRODUCT\_Ad\_Free, StringComparison.Ordinal))

 {

 Debug.Log(string.Format("ProcessPurchase: PASS. Product: '{0}'", args.purchasedProduct.definition.id));

 // TODO: The non-consumable item has been successfully purchased, grant this item to the player.

 GlobalControl.Instance.gamePurchased = 1;

 PlayerPrefs.SetInt ("gamePurchased", GlobalControl.Instance.gamePurchased);

 GlobalControl.Instance.purchasedMusicBool = true;

 PlayerPrefs.SetInt ("purchasedMusic", 1);

 GameControl.instance.DefaultMusicToggle.image.sprite = GameControl.instance.OffSprite;

 GameControl.instance.PurchasedMusicToggle.image.sprite = GameControl.instance.OnSprite;

 GameControl.instance.DefaultMustPurchaseText.gameObject.SetActive(false);

 GameControl.instance.Song2MustPurchaseText.gameObject.SetActive(false);

 GameControl.instance.DefaultSongText.gameObject.SetActive (true);

 GameControl.instance.DefaultSong2Text.gameObject.SetActive (true);

 }

 else

 {

 Debug.Log(string.Format("ProcessPurchase: FAIL. Unrecognized product: '{0}'", args.purchasedProduct.definition.id));

 }

 // Return a flag indicating whether this product has completely been received, or if the application needs

 // to be reminded of this purchase at next app launch. Use PurchaseProcessingResult.Pending when still

 // saving purchased products to the cloud, and when that save is delayed.

 return PurchaseProcessingResult.Complete;

 }

 public void OnPurchaseFailed(Product product, PurchaseFailureReason failureReason)

 {

 // A product purchase attempt did not succeed. Check failureReason for more detail. Consider sharing

 // this reason with the user to guide their troubleshooting actions.

 Debug.Log(string.Format("OnPurchaseFailed: FAIL. Product: '{0}', PurchaseFailureReason: {1}", product.definition.storeSpecificId, failureReason));

 }

 }

}