Net Checkout

User Manual

SEMANTIC GAMES LLC

Contents

Quickstart Guide

Single Line of Code

Here is the only code you need from Net Checkout to let players buy an item in your game:

```
NetCheckout.StripeClient.Buy("My Item", "4.99", 1);
```

See Configuring Net Checkout to complete your setup.

Example Walkthrough

1. Open the Simple scene in the Demos/_Start folder.
2. Play the scene and go through the example checkout process for Stripe.
3. On the Stripe checkout page, enter any random information, except the card number must be: 4242 4242 4242 4242.
4. Complete the checkout process and follow the rest of the steps.

What is Net Checkout?

Net Checkout is a payment interface that leverages Stripe and PayPal APIs. It allows Unity developers to easily charge players for one-time or recurring purchases in their desktop, mobile and WebGL apps. Developers can avoid the hassle of interpreting payment API documentation to use with Unity's networking system. If you simply want to let users make an in-app purchase, Net Checkout lets you do it in a single line of code!

How does it work?

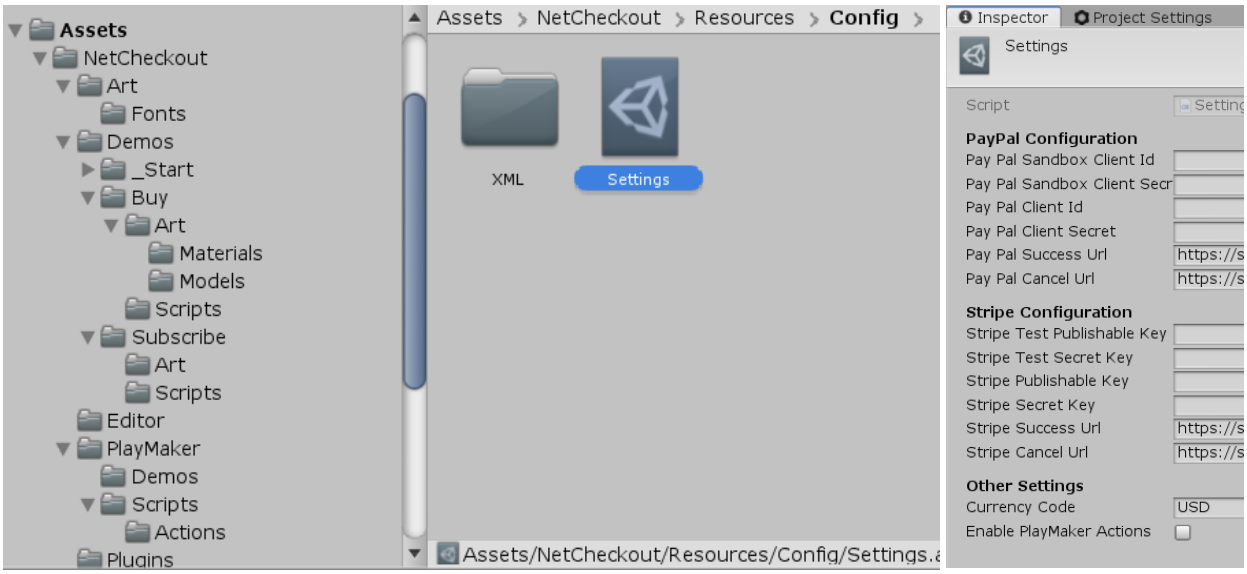
Net Checkout displays a series of messages that walks the player through the purchasing process. It handles all the window management and automatically opens the browser to let the player approve their purchases. The player can then return to the game to confirm their purchase.

As a developer, you can easily choose between Stripe or PayPal clients to handle the actual payment processing. Almost everything is the same between clients, the only difference is Stripe confirms the purchase automatically for players, but PayPal needs players to confirm the purchase after returning to the game.

Configuring Net Checkout

Settings File

Find the Settings file in the NetCheckout/Resources/Config folder. Here you will enter your PayPal or Stripe developer information. See the PayPal or Stripe section below. You can also set the currency code used for purchases. If you are using PlayMaker, enable it here to gain access to all of Net Checkout's custom Actions.



Using PayPal

If you don't already have a developer account, you can create one here.

Log in to the developer dashboard, make sure the Sandbox button is selected, and select Create App. Choose a name (i.e. name of your game) and click Create App. Now click on the app to see your Sandbox API Credentials. Copy and paste the Client ID and Secret into Net Checkout Settings under PayPal Sandbox Client Id and PayPal Sandbox Client Secret. You can now start testing PayPal checkout in editor.

Note: PayPal will ask you to log in using a sandbox account when you try to make a purchase from the Unity editor. Credentials can be found in the dashboard sidebar by clicking Accounts under the Sandbox header. Find the Personal account, click the three dots under Manage accounts, and click View/edit account. Use the Email ID and System Generated Password to log in.

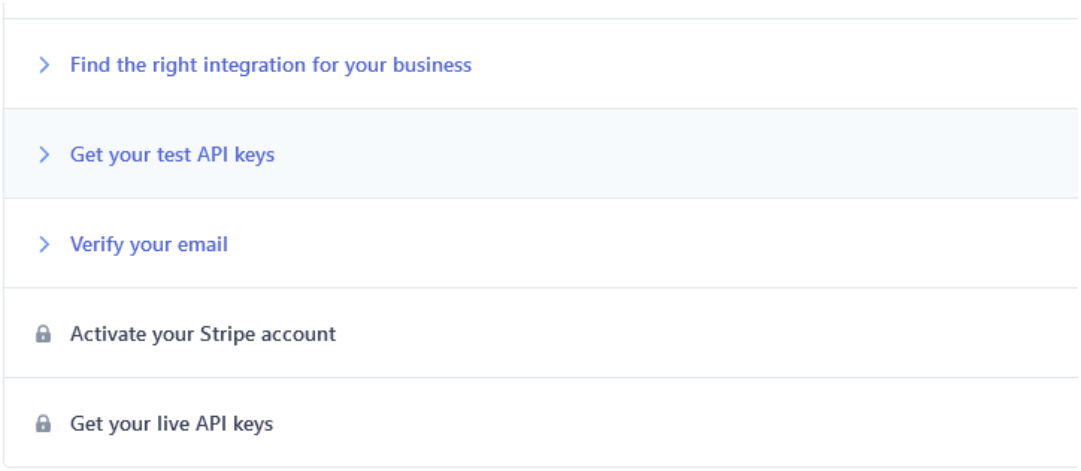
To get a live Client Id and Client Secret, you will need to upgrade your account to business (it's free). From the main dashboard page select the Live tab and upgrade your account. Once you have upgraded you can create an app for live purchases. Select the app to get your live Client Id and Secret and paste them into Net Checkout Settings under PayPal Client Id and PayPal Client Secret.

Success URL and Cancel URL are prefilled with generic pages that inform the user to return to the application. You can change these URLs so PayPal will redirect to your own site.

Using Stripe

If you don't already have a developer account, you can create one here.

Log in to Stripe's developer dashboard, and click Add a Name at the top of the sidebar if you don't already have one. This will be displayed to players on the checkout page. Next, select Get your test API keys. Copy and paste the Publishable key and Secret key into Net Checkout Settings under Stripe Test Publishable Key and Stripe Test Secret Key.



You will need to verify your email before you can activate your Stripe Account. Once you do that, you will be able to access your live API keys. Copy and paste the live Publishable key and Secret key into Net Checkout Settings under Stripe Publishable Key and Stripe Secret Key.

Success URL and Cancel URL are prefilled with generic pages that inform the user to return to the application. You can change these URLs so Stripe will redirect to your own site.

Using the Checkout Class

The Checkout class is Net Checkout's primary class to access its major features. To use it, first include the **using NetCheckout** namespace. Next, create a new Checkout instance and pass in a PayPalClient or StripeClient instance. If you are using PlayMaker, the features are almost identical, just use the SetClient Action to set a PayPal or Stripe client before using the other Actions.

```
PayPalClient client = new PayPalClient();
Checkout checkout = new Checkout(client);
```

Major Functions

Buy(*string itemName, string price, int quantity, action onComplete*)

Charge users a one-time fee. Passes an order id string to the onComplete callback.

```
checkout.Buy("My Item", "2.99", 1, OnBuy);
```

Subscribe(*string planName, string price, PaymentPeriod period, int intervals, action onComplete*)

Charge users a recurring fee. For example, to charge a monthly fee, set the period to month and intervals to 1. Passes a subscription id string to the onComplete callback.

```
checkout.Subscribe("My Plan", "2.99", month, 1, OnSubscribe);
```

GetOrderData(*string orderId, action onComplete*)

Gets data on a purchased item. Passes an OrderData instance to the onComplete callback, which includes item name, price, total price, quantity, and currency.

```
checkout.GetOrderData("order-id", OnComplete);
```

GetSubscriptionData(*string subscriptionId, action onComplete*)

Gets data on a purchased subscription. Passes a SubscriptionData instance to the onComplete callback, which includes plan name, price, period, intervals, active state, and status.

```
checkout.GetSubscriptionData("subscription-id", OnComplete);
```

ActivateSubscription(*string subscriptionId, action onComplete*)

Unpauses the current subscription, so users will start to be charged again.

```
checkout.ActivateSubscription("subscription-id", OnComplete);
```

DeactivateSubscription(*string subscriptionId, action onComplete*)

Pauses the current subscription, so users will not be charged anymore.

```
checkout.DeactivateSubscription("subscription-id", OnComplete);
```

Action Callback

Each of the major functions makes asynchronous web requests to PayPal or Stripe API endpoints. Once the response is received, it passes the results in an action callback. To read the results, you will need to implement a function that accepts two parameters: a bool and an object. Pass that function as an onComplete argument, and Net Checkout will call it once it completes the request. The bool function parameter will be true if the response contains a

successful status code, and the object parameter will typically contain an id or error message. The object could also contain order or subscription data.

```
void OnBuy(bool success, object data) {
    Debug.Log("my order id: " + data.ToString());
}
```

Configuring the Checkout Windows

Net Checkout's built-in window controller allows you to easily customize the windows and messages displayed during the checkout process.

Message Config

To customize the window messages, go to the MessageConfig.xml file located in NetCheckout/Resources/Config/XML.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Default messages for each window in the NetCheckout process. -->
<MessageConfig>
  <OrderWindow>
    <Header>Buy Now</Header>
    <Body>Click below to open the payment service window and authorize your purchase.</Body>
    <ButtonTitle>Pay Now</ButtonTitle>
  </OrderWindow>

  <WaitWindow>
    <Header>Waiting for Approval</Header>
    <Body>Approve your payment in the payment service window.</Body>
    <ButtonTitle></ButtonTitle>
  </WaitWindow>

  <ConfirmWindow>
    <Header>Confirm Purchase</Header>
    <Body>Click below to complete your purchase.</Body>
    <ButtonTitle>Confirm Purchase</ButtonTitle>
  </ConfirmWindow>
```

There are six types of window messages that can be configured. The OrderWindow is the first window to be displayed when the Buy method is called. The WaitWindow is the second window to be displayed when a player clicks the payment button in the OrderWindow to open PayPal or

Stripe checkout in the browser. This window regularly checks if the player has approved or confirmed their purchase. Afterward, for PayPal, the ConfirmWindow is displayed to confirm the player's purchase. Stripe automatically confirms the payment from the browser, so it skips this window. Finally, the CompleteWindow is displayed to inform players whether the purchase was successful. For subscriptions, the SubscribeWindow acts like the OrderWindow, and the SubscribeCompleteWindow acts like the CompleteWindow.

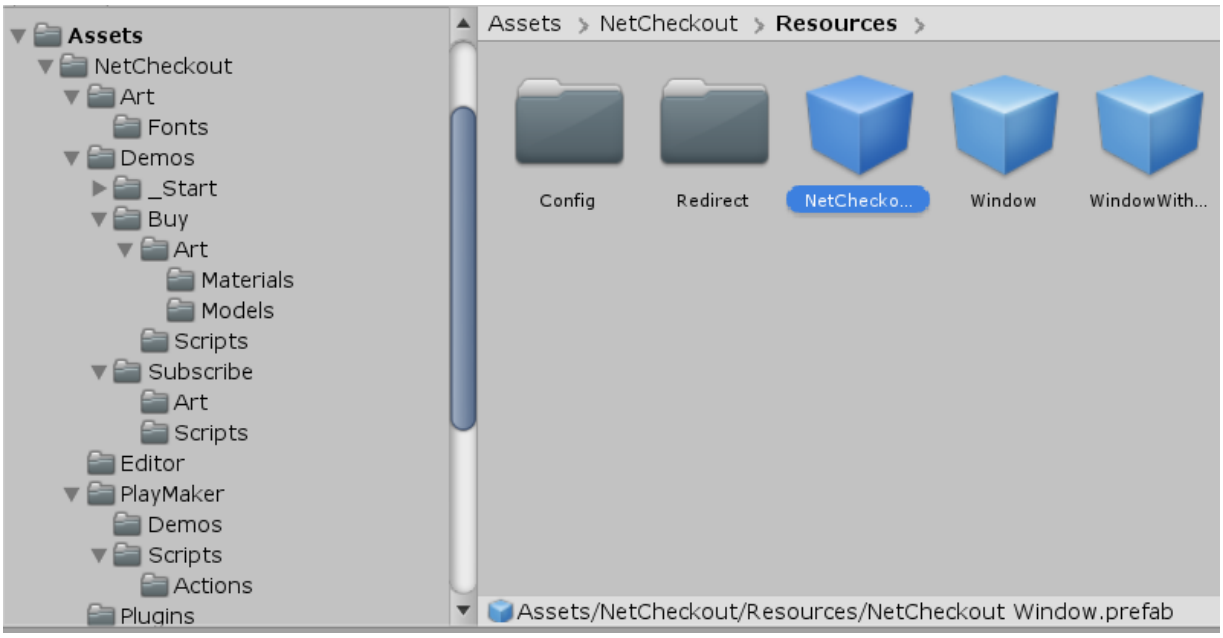
Note: There is no way to determine if a player canceled their purchase in the browser, so the WaitWindow will remain on screen until they close it.

To change the messages dynamically at runtime, you can change the MessageConfig variable from the Checkout instance:

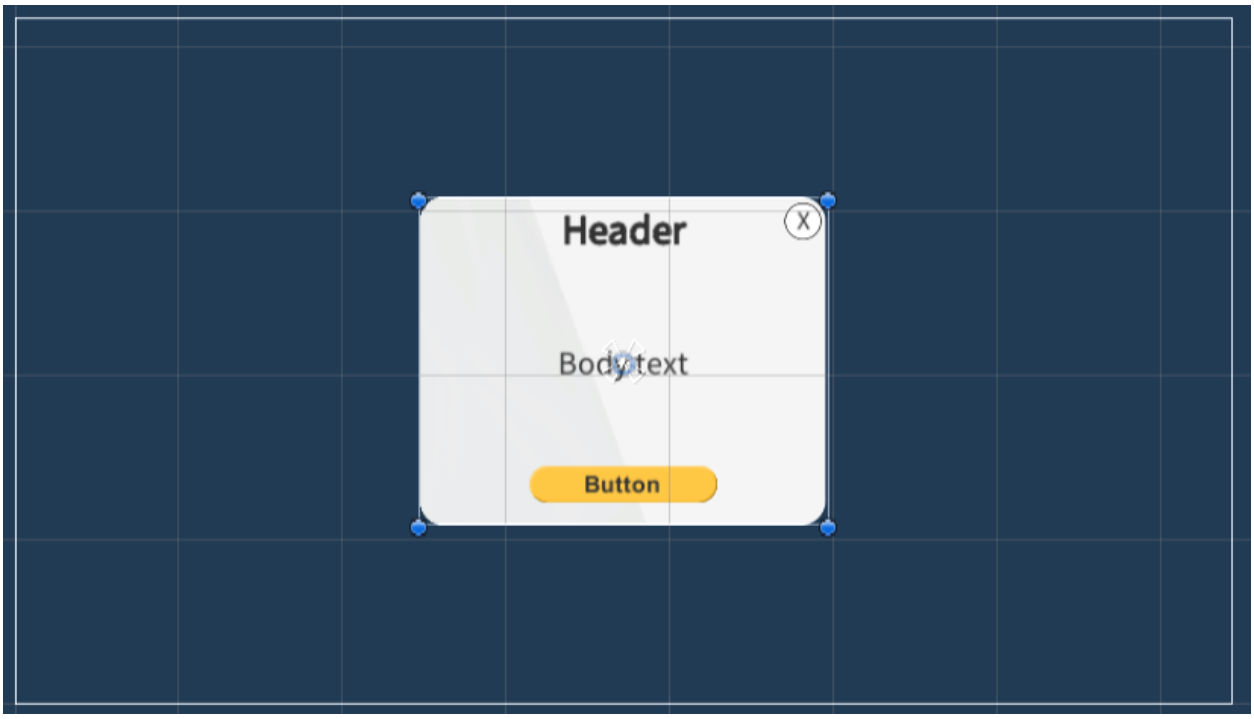
```
checkout.MessageConfig.orderWindow.header = "My Dynamic Header";
```

Window Prefabs

To change the look of the UI, you can change the Window prefab in the NetCheckout/ Resources folder. In addition, the NetCheckout Window prefab has a WindowController component which contains an array of all window prefabs that can be displayed by the controller.



You could use a different prefab for each window in the checkout process if you wanted to. The prefabs must contain a Window component to be added to the controller's array. To easily create a new custom window, simply duplicate the existing Window prefab by selecting it and pressing Ctrl + D (or Cmd + D).



There are two ways to assign the prefab to the appropriate window: (1) Using the MessageConfig.xml file or (2) setting it in code. The MessageConfig file is located in the NetCheckout/Resources/Config/XML folder. Between a window's tags, simply add a new tag called PrefabIndex and set the index from the WindowController prefab array.

```
<PrefabIndex>2</PrefabIndex>
```

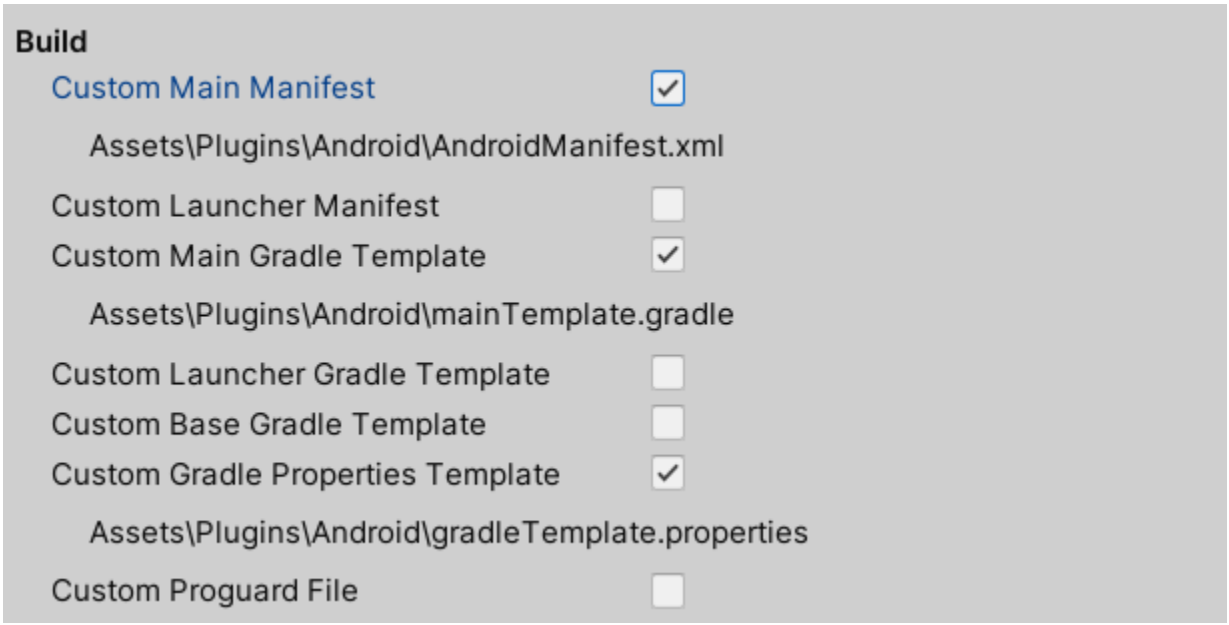
Note: for the CompleteWindow, the tag must go under the SuccessState and/or the FailureState.

To set the prefab index in code, set it in the MessageConfig variable:

```
checkout.MessageConfig.orderWindow.prefabIndex = 2;
```

Android Setup

Go to Project Settings -> Player -> Publishing Settings, and make sure these boxes are checked:



Next, open the *mainTemplate.gradle* file and add these lines of code to the end:

```
dependencies {
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.browser:browser:1.0.0'
    implementation 'com.android.volley:volley:1.2.1'
}
```

Open the *gradleTemplate.properties* file and add this line of code:

```
android.useAndroidX=true
```

Open the *AndroidManifest.xml* file, and at the end of the file, right before the last </manifest> tag, add the following code:

```
<queries>
    <intent>
        <action
android:name="android.support.customtabs.action.CustomTabsService” />
    </intent>
</queries>
```

Browser Callback

Net Checkout uses deep links to return users to your app after completing a payment. Let’s set this up now. You will need a unique URI for your app, otherwise another app on the user’s device could have the same URI and Android won’t know which app to send users back to. Open the *AndroidManifest.xml* file again and find this exact line *</intent-filter>*. Below that line, add these lines of code:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="netcheckout"
android:host="semanticgamesllc.payment_callback" />
</intent-filter>
```

Change the *android:scheme* from “netcheckout” to your own scheme (such as your app’s unique name). Change the *android:host* from “semanticgamesllc.payment_callback” to your own host (such as your own domain name). When Android calls the URI, it will use this format: *scheme://host* (for example, netcheckout://semanticgamesllc.payment_callback).

Next, in the Settings window, under the **Other Settings** header, change the Android App Uri to your custom URI following the *scheme://host* format.

WebGL Builds (Stripe)

Stripe needs an html file to redirect properly to the browser. For WebGL builds, you need to include this html file in your build folder. Net Checkout includes a template for building with Stripe that already contains this html file. The template is located under NetCheckout/WebGLTemplates/NetCheckout-Stripe. Unity will only recognize the template if you move the WebGLTemplates folder into your Assets folder. If you have your own template, you can simply copy the stripe-redirect.html file to your own template. Make sure the redirect file is located at the same relative path as the index file.

Security

If you are concerned about embedding your PayPal or Stripe secret key into your desktop game, you can always leave it blank and set it manually from your client instance with *SetClientSecret(key)*. You will need to store your secret key on a remote server and use a login system to retrieve the secret key once players have been authenticated. Hackers could technically still read the memory in your running application, but it is more difficult to achieve. Since WebGL games are hosted on a server, this is not an issue.

Tips

Make sure to check out the Demos folder to see many of Net Checkout's functionalities already implemented for you. For PlayMaker demos, go to NetCheckout/PlayMaker/Demos.

Use another tool such as Easy Save to store your players' purchases in the cloud. If their local data gets corrupted or deleted, you don't want them to lose their purchased items.

Don't Forget to uncheck the *Build With Test Keys* option when you're ready for release. This will use the production API keys in builds to allow live transactions. You can find this option in Net Checkout's settings (Net Checkout/Resources/Config/Settings) in the Inspector.

(Mobile only) Google and Apple are very particular about accepting apps that use third-party payment systems, so please make sure your app follows the policies they have outlined for Google Play Store and App Store.

Questions

If you have any questions, suggestions, or issues, contact us at semanticgamesllc@gmail.com.