

## **PayPal Apps using PayPal REST APIs**

PayPal Apps showcase the features of PayPal's REST APIs

- Save Credit Card with PayPal for later Payments
- Make Payments using the saved Credit Card ID
- Make Payments using PayPal as the Payment Method

## PayPal Core SDK

The PayPal Core SDK is a foundational library used by all of PayPal's C# SDKs. This SDK provides functionality such as configuration, credential management, connection management, logging etc that are used by the other SDKs. The SDK is also distributed via NuGet.

### OpenId Connect Integration

- Redirect your buyer to obtain authorization
- Capture the authorization code that is available as a query parameter ("code") in the redirect URL
- Exchange the authorization code for an access token, refresh token, id token combo

```
Dictionary<string, string> configurationMap = new Dictionary<string, string>();  
configurationMap.Add("clientId", "...");  
configurationMap.Add("clientSecret", "...");  
configurationMap.Add("mode", "live");
```

```
APIContext apiContext = new APIContext();  
apiContext.Config = configurationMap;  
...
```

```
CreateFromAuthorizationCodeParameters codeParams = new  
CreateFromAuthorizationCodeParameters();  
codeParams.SetCode("code");  
TokenInfo token = TokenInfo.CreateFromAuthorizationCode(apiContext, codeParams);  
string accessToken = token.access_token;
```

- The access token is valid for a predefined duration and can be used for seamless exactly once (XO) or for retrieving user information

...

```
TokenInfo infoToken = new TokenInfo();  
infoToken.refresh_token = "refreshToken";  
UserInfoParameters infoUserParams = new UserInfoParameters();  
infoUserParams.SetAccessToken(infoToken.access_token);  
UserInfo infoUser = UserInfo.GetUserInfo(apiContext, infoUserParams);
```

- If the access token has expired, you can obtain a new access token using the refresh token from the 3'rd step

...

```
CreateFromRefreshTokenParameters params = new CreateFromRefreshTokenParameters();  
params.SetScope("openid"); // Optional  
TokenInfo info = new TokenInfo(); // Create TokenInfo object; setting the refresh token  
info.refresh_token = "refreshToken";  
info.CreateFromRefreshToken(apiContext, params);
```

# PayPal Core SDK Configuration Parameters

The PayPal .NET SDKs can be configured with information such as API credentials, integration mode etc using the configuration mechanisms listed here.

- [How to configure](#)
- [List of configuration parameters](#)
- [Logging](#)

## How to configure

The SDK can be configured using the Web.Config/App.Config files (useful for simple applications) or programmatically using a Dictionary (useful for larger applications that have complex configuration requirements or need to read account information from a database etc).

Using the Web.Config / App.Config files.

```
<configSections> <section name="paypal" type="PayPal.Manager.SDKConfigHandler, PayPalCoreSDK" /> <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" /> </configSections> <!-- PayPal SDK config --> <paypal> <settings> <add name="mode" value="sandbox"/> <add name="connectionTimeout" value="30000"/> ..... </settings> <accounts> <account apiUsername="jb-us-seller_api1.paypal.com" apiPassword="..." apiSignature="..." /> <account apiUsername="enduser_biz_api1.gmail.com" apiPassword="..." apiCertificate="..." privateKeyPassword="..." /> </accounts> </paypal>
```

If you want to use a dynamic configuration instead,

Create a dictionary with the required configuration parameters and pass the dictionary to your service object constructor. For example,

```
Dictionary<string, string> config = new Dictionary<string, string>(); config.Add("mode", "sandbox"); config.Add("account1.apiUsername", "jb-us-seller_api1.paypal.com"); config.Add("account1.apiPassword", "..."); config.Add("account1.apiSignature", "..."); PayPalAPIInterfaceService s = new PayPalAPIInterfaceService(config);
```

# List of configuration parameters

## Service Configuration

- Integration mode

PayPal provides live and a sandbox environments for API calls. The live environment moves real money while the sandbox environment allows you to test your application with mock money before you go live. You must set the mode property to either 'sandbox' or 'live'.

```
mode=sandbox
```

## API credentials

You can configure one or more API credentials using the account\* parameters. PayPal API calls can be authenticated using multiple methods

- 3-token authentication, OR
- client certificate authentication.

You can obtain your credentials by logging in to your PayPal account and navigating to the **Profile -> My Selling Tools -> API Access** page. Make sure to also go through the [Getting started guide](#). If you are using the Adaptive APIs, you will need to pass in an [Application ID](#) as well.

### Sample 3-token configuration

```
account1.apiUsername = jb-us-seller_api1.paypal.com account1.apiPassword = WX4WTU3S8MY44S7F
account1.apiSignature = AFcWxV21C7fd0v3bYYRCpSSRl31A7yDhhsPUU2XhtMoZXsWHF Xu-RWy
account1.applicationId = APP-80W284485P519543T
```

### Sample certificate configuration

```
account2.apiUsername = certuser_biz_api1.paypal.com account2.apiPassword = D6JNKKULHN3G5B8A
account2.apiCertificate = resource/sdk-cert.p12 account2.privateKeyPassword = password
account2.applicationId = APP-80W284485P519543T
```

## Third party API calls

You can execute third party API calls on behalf of other paypal accounts. You can use the [Permissions API](#) to get an authorization token and secret for a third party.

### Sample token based third party configuration

```
IThirdPartyAuthorization thirdPartyAuth = new TokenAuthorization("accessToken",  
"tokenSecret"); SignatureCredential cred = new SignatureCredential("jb-us-  
seller_api1.paypal.com", "WX4WTU3S8MY44S7F",  
"AFcWxV21C7fd0v3bYYRCpSSR131A7yDhhsPUU2XhtMoZXsWHF Xu-RWy"); cred.ApplicationID = "APP-  
80W284485P519543T"; cred.ThirdPartyAuthorization = thirdPartyAuth; MassPayResponse response =  
service.MassPay(massPayReq, cred);
```

## Connection Information

- defaults to 3000 if not specified

```
connectionTimeout=3000
```

- defaults to 1 if not specified

```
requestRetries=1
```

- defaults to 127.0.0.1 if not specified

```
IPAddress=127.0.0.1
```

## HTTP Proxy configuration

- If you are using proxy replace the following values with your proxy parameters

```
proxyAddress=proxyIP[:port] proxyCredentials=domain\\user:password
```

## Developer Sandbox Email Address

- Developer email address is required only by the AdaptiveAccounts API for creating accounts on sandbox environment

```
sandboxEmailAddress=pp.devtools@gmail.com
```

## Logging

PayPal's SDKs provide out of the box support for logging using log4net and the Systems.Diagnostics based trace system. To configure a logging mechanism of your choice, add the following to your Web/App.Config file.

```
<appSettings> <add key="PayPalLogger" value="PayPal.Log.DiagnosticsLogger,  
PayPal.Log.Log4netLogger"/> </appSettings>
```

Notice that you can configure multiple loggers. Also remember to add the logging framework specific configuration section as applicable.

## PayPal REST SDK

### To make an API call

- Create 'accesstoken' from 'ClientID' and 'ClientSecret' using OAuthTokenCredential and set the same in resource as follows,  

```
// Retrieve the access token from
// OAuthTokenCredential by passing in
// ClientID and ClientSecret
// It is not mandatory to generate Access Token on a per call basis.
// Typically the access token can be generated once and
// reused within the expiry window
string accessToken = new
OAuthTokenCredential(ConfigManager.Instance.GetProperties()["ClientID"],
ConfigManager.Instance.GetProperties()["ClientSecret"]).GetAccessToken();
```
- Depending on the context of API calls, calling method may be static or non-static (For example, most 'GET' http methods are created as static methods within the resource).  
If it is static, invoke it as a class method as seen here  

```
// Retrieve the payment object by calling the
// static `Get` method
// on the Payment class by passing a valid
// AccessToken and Payment ID
Payment pymnt = Payment.Get(accessToken, "PAY-0XL713371A312273YKE2GCNI");
```
- If it is non-static, invoke it using resource object using the following syntax  

```
// A Payment Resource; create one using
// the above types and intent as `sale`
Payment pymnt = new Payment();
pymnt.intent = "sale";
...
...
pymnt.Create(accessToken);
```

# Windows 8 Checkout SDK

The *PayPal Windows 8 Checkout SDK* gives you the ability to integrate PayPal payment functionality into the apps you create for the Windows 8 Store and the Windows 8 Phone.

PayPal is the most trusted payment system for online transactions, and offering PayPal as a payment method lets your customers be secure with the transactions they complete with your app. In addition to offering secure payments, PayPal also gives you the ability to sell digital goods from within your app, which is a unique offering among Microsoft platform payment methods. Accepting PayPal can open new revenue streams because you can sell additional program content and software updates from within your Windows app.

Windows 8 Checkout SDK uses **Mobile Express Checkout** for its underlying technology, and it offers the same functionality with the same pricing model. Before integrating the SDK requests into your Windows 8 app, make sure you understand how to complete a Mobile Express Checkout transaction as described in the [Mobile Express Checkout Getting Started Guide](#).

## Overview

To integrate PayPal payments into your Windows 8 applications:

1. **Configure** Visual Studio with the PayPal Visual Studio extension.
2. **Create** the needed PayPal merchant and test accounts.
3. **Call** the interfaces from your app.
4. **Test** the payment functionality.
5. **Launch** your app on the Windows App Store.

**Note** To develop Windows 8 applications, you must develop on the Windows 8 OS and use Visual Studio 2012.

## Windows 8 Store vs. Windows 8 Phone integrations

You can integrate Windows 8 Checkout SDK into both Windows 8 Store applications and Windows 8 Phone apps. The Windows 8 Checkout SDK uses a Windows Runtime Component for the Store applications, while Windows Phone apps make use of a Portable Class library. The main differences between these two integration types are summarized in the following table:

Integration Type	Library File	Languages Supported
Windows 8 Store	PayPal.Checkout.vsix	XAML + C# XAML + C++ JavaScript
Windows 8 Phone	PayPal.Checkout.SDK-WindowsPhone8.dll	C#



# Setting up the SDK

Setting up the SDK for usage is a two-step process.

## 1. Install the SDK library files

The [PayPal Windows 8 Checkout SDK](#) page contains the latest version of the SDK library files.

Download and configure the SDK files according to the type of app you're creating:

### Windows 8 Store applications

Download and run `PayPal.Checkout.vsix` to install the library for global use.

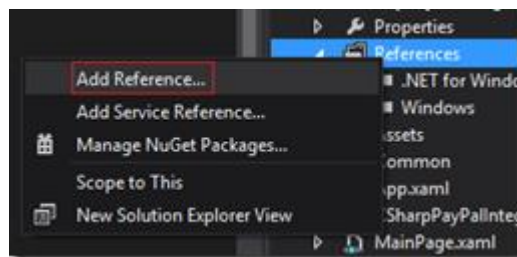
### Windows 8 Phone apps

Download `PayPal.Checkout.SDK-WindowsPhone8.dll` into your project's `lib` directory.

## 2. Reference the SDK from your project

To make use of the PayPal Windows 8 Checkout SDK in your Visual Studio projects:

1. Launch Visual Studio 2012 and open up the Windows 8 project to which you're adding the PayPal checkout functionality.
2. Right-click **References** in the Solution Explorer and select **Add Reference**.
  - For Store applications, navigate to **Windows > Extensions**, select **PayPal Checkout SDK** and click **OK**.
  - For Phone apps, browse to the **PayPal.Checkout.SDK-WindowsPhone8.dll** file and click **OK**.



*The Solution Explorer*

You're now ready to make calls to the Windows 8 Checkout interfaces. Time to code!

## PayPal accounts in SDK requests

Each PayPal transaction in a Windows 8 app involves three different PayPal accounts:

- The **receiver** account: The business account that receives (or refunds) the transaction funds.
- The **caller** account: The app account used to authenticate the API calls made from your app.
- The **sender** account: The payer account, also known as the buyer or customer account.

Your requests through the SDK reference these PayPal accounts by either their API credentials or by their associated PayPal account details (the PayPal account email or user ID value).

**Important:** As you develop your app and go live, you will make use of two different sets of user accounts. One is the set of Sandbox test accounts that you use while testing. The other set of accounts represent the real users that partake in the live transactions that your app generates as it runs in production. You configure the different live and test account values in the SDK.

## The receiver and sender accounts

In live transactions, the *receiver* account is a live PayPal Business account. This is your live merchant account. When testing, configure a Sandbox **Business** test account that you use as the receiver.

The *sender* account in live transactions is the customer who uses your Windows app. For testing purposes, you configure a Sandbox **Personal** test account to represent the buyer.

**Note** When you create a new PayPal Business account, PayPal pre-configures a Sandbox Business account that you can use in your test transactions. However, you must still create a Sandbox Personal account to act as the buyer in your mock PayPal transactions. See [\\_Creating a Personal test account in Testing Classic API Calls](#) for details on creating test accounts and using the PayPal Sandbox.

## The caller account

The *caller* account is a PayPal account that is explicitly configured to run with Windows Checkout SDK. Using a Pre-configured account protects your API credentials from being exposed and simplifies the transaction processing. You reference the caller account by their associated PayPal User ID values, as defined in the following table:

Pre-configured Windows App caller accounts	
Environment	PayPal Caller Account User ID
Sandbox	mswin_1352404208_biz_api1.paypal.com
Live	windowsstore_api1.paypal.com

## API caller permissions

It's important to understand the relationship between the caller and receiver accounts. The caller account makes API calls *on behalf* of the receiver account. Because of this, you must grant permission for the caller account to make requests on the behalf of your merchant account.

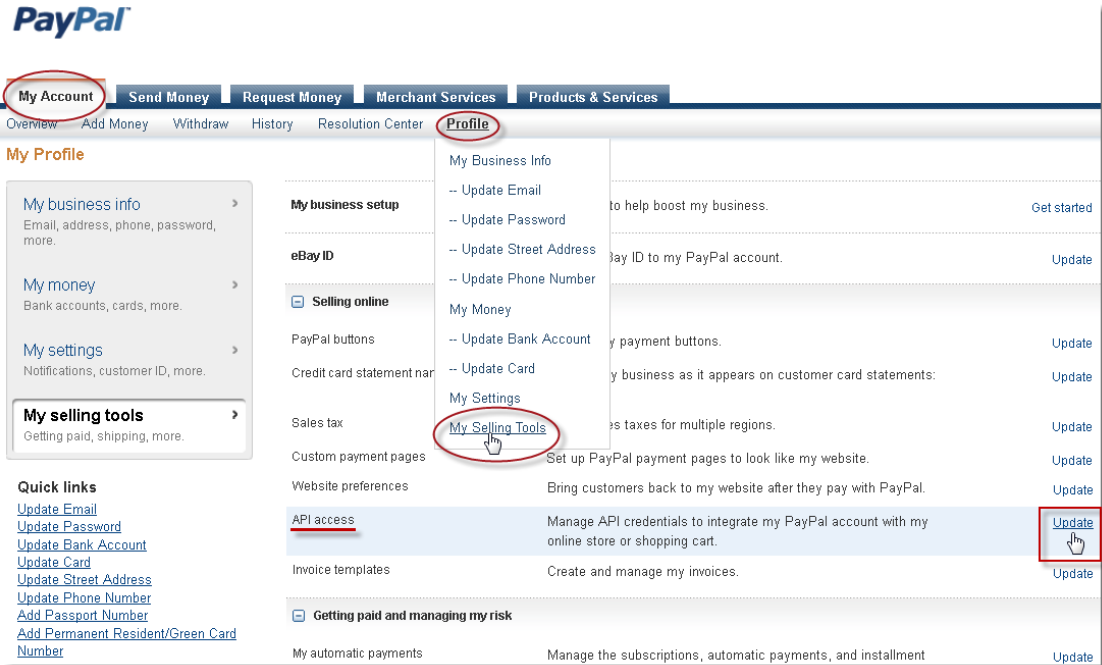
During the development process, you will grant permission twice, once for the Sandbox Business (merchant) account and once for your live merchant account. The process for granting permissions for each account type is nearly identical, the only difference is the environment you use to assign the permissions. You assign permissions for your live merchant account from the account settings on PayPal.com (<https://paypal.com>). For testing, you assign permissions from your Business test account using the *Sandbox test site* (<https://www.sandbox.paypal.com>).

To assign permissions for your Business test account:

1. Log in to the [Sandbox test site](https://www.sandbox.paypal.com) using the log-in credentials of your Business test account.

**Note:** You must first sign in to the Developer website using your merchant account before you can sign in to the Sandbox test site using the login credentials from one of your test accounts.

2. Select **My Account > Profile > My Selling Tools**, then click the **Update (API Access)** link:



### *Navigating to the API Access page*

If you're already logged in to the Sandbox site with your receiver test account, you can navigate directly to the API Access page using the following link: . [https://www.sandbox.paypal.com/us/cgi-bin/webscr?cmd=\\_profile-api-list-auths](https://www.sandbox.paypal.com/us/cgi-bin/webscr?cmd=_profile-api-list-auths)

3. On the **API Access** page, click **Grant Permissions** under **Option 1**.

The **Add New Third Party Permissions** page displays.

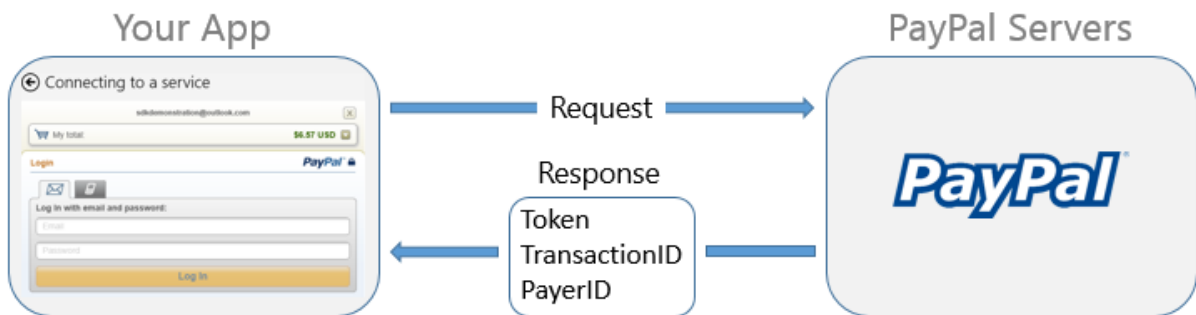
4. In the **Third Party Permission Username** input box, enter the User ID of the caller account to which you are granting permissions, and click the **Lookup** button.
  - Use `mswin_1352404208_biz_api1.paypal.com` for the Sandbox environment.
  - Use `windowsstore_api1.paypal.com` for the live environment.
5. From the list of Available Permissions, check the following boxes, then click **add** to establish the permissions:
  - **Use Express Checkout to process payments**
  - **Use Express Checkout to process mobile payments**
6. (Optional) If your situation requires, configure your account to accept payments for Digital Goods. If you do not, users will not be allowed to purchase digital goods from your app using the SDK. See [Accepting a Basic Digital Goods Payment Using Express Checkout](#) for more information.

That's all you need to do to configure your accounts to use SDK! Notice that for testing, you only need to add SDK permissions to your Sandbox Business account, you need not update any configuration settings for your Sandbox Personal test accounts.

For more information on granting permissions and making calls on behalf of a third-party, see the [Permissions Service Getting Started Guide](#).

## Processing PayPal transactions

The following diagram shows the process flow of a Windows 8 Checkout SDK transaction:



*The Windows 8 Checkout SDK app flow*

Here's the flow:

1. Your app sends a purchase request to PayPal, including information describing the item (or cart of items) to be purchased, and the authentication information for the PayPal buyer account.
2. Upon approval of the payment, PayPal responds to your app with the following:
  - **Token** - Authentication token unique to the transaction.
  - **Transaction ID** - Used in subsequent requests associated with the transaction.
  - **Payer ID** - Uniquely identifies the payer and needed to verify the payment.

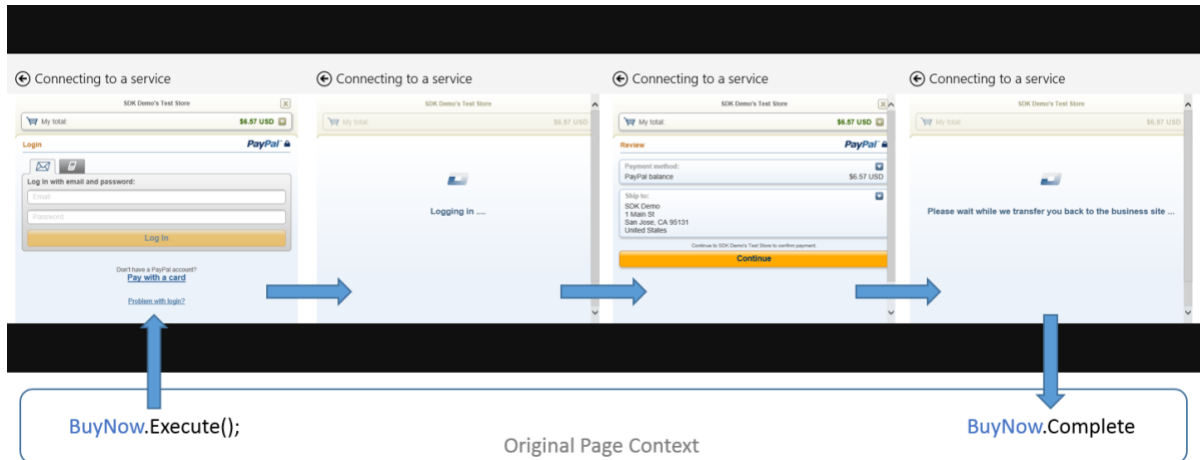
## Running your app in test mode

Make sure your app runs as expected by testing its functionality using the Sandbox test accounts you've configured. While testing, use a *Personal* test account to represent the buyer in your test transactions.

To test your app, run your app and click the **Purchase Item** button to have your app launch the PayPal login page. Use the login credentials from your buyer test account to sign into the PayPal Sandbox.

From here you can complete or cancel the purchase. When you complete a purchase, transaction records are sent to both your Sandbox Personal and Business (merchant) accounts.

The following diagram shows how the transaction process looks to your app user:



### The user flow

When testing, complete or cancel the transaction as you expect your users will; the Sandbox is a self-contained environment where you can review your transaction results from both the payer and payee's standpoints. Create a robust application by testing all process flows to ensure your app gracefully handles all user actions. Streamline your PayPal transaction processing, then push your app to production.

## Launching your app

The next step is to launch your app in the Windows App Store. Configure your app to go live as follows:

1. In the SDK, set `BuyNow.UseSandbox = false`.

This configures the SDK to use the live PayPal account values you entered, instead of using the configured Sandbox values.

2. Follow the directions in the *API Caller Permissions* section above to configure the permissions for your live merchant account.

Your app is now ready to be submitted for certification as a Windows App and once certified, you can launch the app in the Windows App Store and start collecting profits! For details, see [Using the Windows Store dashboard](#).

# How to make Windows 8 Checkout SDK calls

This page contains an example Windows 8 Store application that shows how to initiate the purchase of an item using the *PayPal Windows 8 Checkout SDK*.

The example app is presented in both C# and JavaScript and it uses PayPal Sandbox accounts to complete mock transactions. The application is minimal, it presents a single button that, when clicked, creates a new instance of the `BuyNow` object that represents a new purchase transaction. While you can use both examples to create Windows 8 Store applications, the C# code can also be used to create Windows 8 Phone apps.

When the user clicks the **Purchase Item** button, the app creates the item to be purchased using the `ItemBuilder` class. The item object contains information such as name, description, and price. The app then adds the item to the `BuyNow` object and executes the transaction flow.

While there's a lot of functionality consolidated into one button, the code that accomplishes the tasks is fairly streamlined.

## Create and purchase an item using C#/XAML

Launch Visual Studio and use a **Blank App Template** to create a new C#/XAML project with the name of your choice. After creating your project, add a reference from the project to the PayPal SDK library.

In the Solution Explorer, open `MainPage.xaml` and insert the following XAML button into the blank Grid control:

```
<Button Content="Purchase Item" HorizontalAlignment="Left" Height="100"
    VerticalAlignment="Top" Width="200" Foreground="White"
    x:Name="PurchaseButton" Click="PurchaseButtonClick" />

<TextBlock x:Name="PurchaseStatus" />
```

After creating the button, open `MainPage.xaml.cs` and insert the following handler for that button's click event:

```
using PayPal.Checkout;

private async void PurchaseButtonClick(object sender, RoutedEventArgs e)
{
    // Create a new instance of the BuyNow object to represent a purchase
    // The BuyNow constructor takes your PayPal merchant account email address as argument.
    PayPal.Checkout.BuyNow purchase =
        new PayPal.Checkout.BuyNow("insert merchant account address here");

    // Configure it to use the SandBox environment.
    purchase.UseSandbox = true;

    // Set the currency to use US Dollars
    purchase.Currency = "USD";
}
```

```

// Use the ItemBuilder to create a new example item
PayPal.Checkout.ItemBuilder itemBuilder = new PayPal.Checkout.ItemBuilder("Example Item")
    .ID("Item12345")
    .Price("10.00")
    .Description("An Example Item")
    .Quantity(1);

// Add the item to the purchase,
purchase.AddItem(itemBuilder.Build());

// Attach event handlers so you will be notified of important events
// The BuyNow interface provides 5 events - Start, Auth, Cancel, Complete and Error
// See http://paypal.github.io/Windows8SDK/csharp.html#Events for more
purchase.Error += new EventHandler<PayPal.Checkout.Event.ErrorEventArgs>((source,
eventArg) =>
{
    this.PurchaseStatus.Text = "There was an error processing your payment: " +
eventArg.Message;
});
purchase.Complete += new EventHandler<PayPal.Checkout.Event.CompleteEventArgs>((source,
eventArg) =>
{
    this.PurchaseStatus.Text = "Payment is complete. Transaction id: " +
eventArg.TransactionID;
});
purchase.Cancel += new EventHandler<PayPal.Checkout.Event.CancelEventArgs>((source,
eventArg) =>
{
    this.PurchaseStatus.Text = "Payment was canceled by the user.";
});

// Launch the secure PayPal interface. This is an asynchronous method
await purchase.Execute();
}

```

Lastly, make sure to replace the placeholder text in the initialization code for the BuyNow object with the email address of your Sandbox merchant account. And that's it, you're done!

Run the app and click the button to see what the user will see when purchasing items from your store.

For more information, see the Windows 8 Checkout SDK Reference:

- [C# API Reference](#)



## Create and purchase an item using JavaScript/HTML5

Launch Visual Studio and use a **Blank App Template** to create a new JavaScript project with the name of your choice. After creating your project, add a reference from the project to the PayPal SDK.

In the Solution Explorer, open the file `default.html`. Find the `<body>` tag and replace its contents with the following HTML button:

```
<p>
  <button id="PurchaseButton">Purchase Item</button>
  <textarea id="PurchaseStatus"></textarea>
</p>
```

In the Solution Explorer, open the `js` folder, then open the file `default.js`. Remove the `app.onactivated` event handler and replace it with the following JavaScript code that registers and defines a handler the button's click event:

```
app.onactivated = function (args) {
  if (args.detail.kind === activation.ActivationKind.launch) {
    if (args.detail.previousExecutionState !==
activation.ApplicationExecutionState.terminated) {
      // TODO: This application has been newly launched. Initialize
      // your application here.
    } else {
      // TODO: This application has been reactivated from suspension.
      // Restore application state here.
    }
    WinJS.UI.processAll().done(function () {
      var purchaseButton = document.getElementById("PurchaseButton");
      purchaseButton.addEventListener("click", purchaseButtonClick, false);
    });
  }
};

// handler for the purchase button click
function purchaseButtonClick() {

  // Create a new instance of the BuyNow object to represent the purchase.
  // The BuyNow constructor takes your PayPal merchant account email address as argument.
  var purchase = new PayPal.Checkout.BuyNow("insert merchant account address here");

  // Configure it to use the Sandbox environment
  purchase.useSandbox = true;

  // Set the currency to US Dollars
  purchase.currency = "USD";

  // Use the ItemBuilder to create a new example item
  var item = PayPal.Checkout.ItemBuilder("Example Item")
    .id("Item12345")
    .price("10.00")
    .description("An Example Item")
    .quantity(1)
```

```

        .build();

// Add the item to the purchase.
purchase.addItem(item);

// Attach event handlers so we are notified of important events
purchase.addEventListener('cancel', function (e) {
    document.getElementById("PurchaseStatus").textContent =
        "User canceled payment: " + e.token;
});
purchase.addEventListener('complete', function (e) {
    document.getElementById("PurchaseStatus").textContent =
        "Payment complete: " + e.token + ' transaction id: ' + e.transactionID;
});
purchase.addEventListener('error', function (e) {
    document.getElementById("PurchaseStatus").textContent = "Error: " + e.message;
});

// Launch the secure PayPal interface. This is an asynchronous method
purchase.execute();
}

```

Lastly, make sure to replace the placeholder text in the initialization code for the BuyNow object with the email address of your Sandbox merchant account. And that's it, you're done!

Run the app and click the button to see what the user will see when purchasing items from your store.

For more information, see the Windows 8 Checkout SDK Reference:

- [JavaScript API Reference](#)

# Mobile Express Checkout Getting Started Guide

## Contents

- [Key MEC Concepts](#)
- [Making Your First MEC Call](#)
- [Next Steps](#)

## Mobile Express Checkout Overview

PayPal's *Mobile Express Checkout* (MEC) gives you the ability to place a **Check Out with PayPal** button on your mobile website, which enables customers to check out via their PayPal accounts. You can also use MEC to support users without PayPal accounts by creating a secondary checkout button that lets users log in to PayPal as a guest and pay with a credit or debit card. You can download the PayPal [Mobile Checkout buttons here](#).

MEC is based on Express Checkout. It gives mobile users a streamlined checkout process—they don't have to enter their banking or shipping information into their mobile devices because this information is contained, and shielded, in their PayPal accounts. With MEC, customers follow the same checkout flow as they do with Express Checkout.

## Key MEC Concepts

MEC works on a wide range of mobile devices (such as iPhone, iPad, Android, and Blackberry mobile devices) and has the advantage over Express Checkout in that it optimizes the pages for smaller mobile screens and mobile keyboards. In addition, MEC is flexible in that you can give customers without PayPal accounts the option to check out using their credit or debit cards.

Based on PayPal's Express Checkout, MEC uses the same APIs and transaction flow as does Express Checkout. With MEC, transaction processing is redirected from your mobile website to PayPal, where the customer reviews and completes the payment transaction on PayPal servers. PayPal returns the customer to your mobile site once they complete the payment flow.

Because of its close link to Express Checkout, it is best to become familiar with the workings of Express Checkout before using MEC. The following resources provide a good foundation for how to use MEC:

- [Mobile Express Checkout — Best Practices](#)
- [How to Use Express Checkout for One-Time Payments](#)
- [Express Checkout Advanced Features Guide](#)

**NOTE:** For instructions on getting started with the PayPal APIs, how to use the Sandbox for testing, and how to move your app into production, see [Apps 101](#).

## Making Your First MEC Call

This steps in this section describe how to modify existing Express Checkout code so that you can use it on a mobile website. See the [Express Checkout API Getting Started Guide](#) if you do not have existing Express Checkout code.

To modify existing Express Checkout code to support mobile devices:

1. Modify the `returnUrl` and `cancelUrl` values in your `SetExpressCheckout` call so that they point to the appropriate pages on your mobile site.
2. Modify the `cmd` argument in the Express Checkout redirect URL to `_express-checkout-mobile`.

The redirect URL should resemble the following (wrapped for readability):

```
https://www.paypal.com/cgi-bin/webscr?  
cmd=_express-checkout-mobile  
&token=tokenValueReturnedFromSetExpressCheckoutCall
```

This ensures that PayPal optimizes the browser pages for use with mobile devices.

3. Ensure that you do not use any Express Checkout functionality that is not supported Mobile Express Checkout. For a full list of the functionality supported by MEC, see [Express Checkout on Mobile Devices](#).

To see this code in action, log into the [Sandbox](#) and set up Sandbox test account that you can then use in your calls. You will need both Sender and Receiver test accounts.

**Voila!** You have now updated an Express Checkout payment flow so that it can be used on a mobile site.

# Creating Sandbox Test Accounts

To create a test account in the Sandbox:

1. Log in to the Developer site at <https://developer.paypal.com> and navigate to **Applications > Sandbox Accounts**.
2. Click the **Create Account** button and populate the following fields:

<b>Account type</b>	Tick either the <b>Personal</b> or <b>Business</b> radio button.
<b>Email address</b>	<p>The email address doesn't need to be a real email address; the Sandbox doesn't send any email outside of the Sandbox environment. Email to Sandbox accounts, generated as a result of your test API requests, are listed on both the <a href="#">Notifications</a> tab on the Developer site, as well as on the Sandbox test site.</p> <p>Use the assigned email value to reference this test account in your API calls, and to log in to the Sandbox site when you want to review the details associated with the account.</p>
<b>Password</b>	The password must be 8-20 alpha-numeric characters in length. Use the password to log in to the Sandbox site as the test account.
<b>First and Last names</b>	The optional name fields accept alpha-numeric characters.
<b>PayPal balance</b>	While this field is optional, it's a good idea to create test accounts with positive bank balances. Enter an integer value between 1 and 5000.
<b>Bank Verified Account</b>	You should create both <i>Verified</i> and <i>Unverified</i> test accounts so you can fully test your application.
<b>Select Payment Card</b>	Test payments made with different payment cards by selecting either <b>Discover</b> or <b>PayPal</b> . ( <i>Personal accounts only.</i> )
<b>Credit card type</b>	You must select a single credit card type for each test account. The Sandbox associates a mock credit card number with the account.

<b>Log in with PayPal</b>	<p>Tick the <b>I want to add Log in with PayPal to my site</b> box and complete the following fields to enable Log In with PayPal testing:</p> <ul style="list-style-type: none"> <li>○ <b>Display name</b></li> <li>○ <b>Return URL</b></li> <li>○ <b>Privacy policy URL</b></li> <li>○ <b>User agreement URL</b></li> </ul> <p><i>(Business accounts only.)</i></p>
<b>Notes</b>	Add details specific to this test account.

3. Click **Create Account** after completing the account details.

The new test account is listed with your other test accounts on the Accounts page.

## Accepting credit cards in test transactions

To use a credit card as a payment method in your test transactions, you must configure a test Business account as a PayPal Payments Pro account:

1. Create a test Business account.
2. Navigate to the **Profile** page of the Business account and click the **Upgrade to Pro** link.
3. Click **Enable** on the resulting screen.

**Tip:** Once you enable PayPal Payments Pro for a test Business account, the setting is permanent and you cannot undo the configuration for that account. We recommend you create multiple Business test accounts with various settings in order to test all the variations you might need to handle with your application.

# PayPal Checkout SDK for Windows 8

## PayPal Checkout C# API

The PayPal namespace currently contains the following objects:

- [PayPal.Checkout.BuyNow](#) - The PayPal Checkout API
- [PayPal.Checkout.Item](#) - A model object used to describe an item being purchased
- [PayPal.Checkout.ItemBuilder](#) - The builder used to create an Item

## PayPal.Checkout.BuyNow object

This is the main PayPal checkout API. You can code against this if you have your own UI control or other mechanism to trigger purchases.

### Constructor

<b>BuyNow(String merchantId)</b>	Creates an instance of the BuyNow object for the provided merchantId (the emailid of your PayPal Merchant account).
----------------------------------	---

### Events

Attach event handlers on the BuyNow object to be notified of important events. For example,

```
PayPal.Checkout.BuyNow bn = new PayPal.Checkout.BuyNow();  
bn.Start += new EventHandler<PayPal.Checkout.Event.StartEventArgs>((source, args) => {  
    // Do something  
});
```

<i><b>Event</b></i>	<i><b>Description</b></i>	<i><b>event</b></i>
<b>onstart</b>	Occurs when a transaction is initiated.	Start
<b>onauth</b>	Occurs when buyer authorization is triggered.	Auth: token
<b>oncancel</b>	Occurs when a user cancels the transaction. <b>Note:</b> <i>complete</i> and <i>error</i> will not fire.	Cancel: token
<b>oncomplete</b>	Occurs when a user successfully completes a transaction. <b>Note:</b> <i>cancel</i> and <i>error</i> will not fire.	Complete: token, transactionId, payerId
<b>onerror</b>	Occurs when a transaction is unable to be completed due to an error. <b>Note:</b> <i>cancel</i> and <i>complete</i> will not fire.	Error: message, [status], [code]

## Methods

<b>Execute()</b> : boolean	Initiates a purchase transaction based on the information set on the BuyNow object. Note that is an asynchronous method.
<b>AddItem(PayPal.Checkout.Item item)</b> : void	Adds a PayPal.Checkout.Item to the current purchase.
<b>clearItems()</b>	Clears all items from the current purchase.

## Properties

<i><b>Name (Type)</b></i>	<i><b>Description</b></i>
<b>merchantId</b> (String)	The merchant ID for the current transaction. This is the emailid of your PayPal Merchant account.
<b>UseSandbox</b> (bool)	Set to true to use the PayPal <a href="#">Sandbox</a> environment, false to use the live PayPal environment. Defaults to false.
<b>displayShipping</b> (Number)	(optional, defaults to 0) Indicates how to display shipping options: 0 = Show the shipping address. 1 = Hide the shipping address. 2 = Use the address from the buyers profile if one isn't provided.  <b>Note:</b> This property is automatically set to 1 for Digital Goods transactions.
<b>confirmedShippingOnly</b> (Boolean)	(optional, defaults to false) Set to true if you require the buyer to have a confirmed shipping address.  <b>Note:</b> This property is automatically set to false for Digital Goods transactions.
<b>currency</b> (String)	(optional, defaults to 'USD') Set to the three-character ISO-4217 currency code for this purchase. See table 2 in <a href="https://developer.paypal.com/webapps/developer/docs/classic/api/currency_codes/">https://developer.paypal.com/webapps/developer/docs/classic/api/currency_codes/</a> for full list of supported currencies.
<b>IPN</b> (String)	(optional) An <a href="#">Instant Payment Notification</a> URL to which PayPal should POST purchase information.



<b>custom</b> (String)	(optional) An arbitrary custom string value to associate with the current transaction. (NOTE: This field is limited to 256 single-byte alphanumeric characters.)
<b>locale</b> (String)	(optional, defaults to 'en_US') Set to the five-character locale code to control the locale of PayPal pages.
<b>paymentMethod</b> (Paypal.Checkout.PaymentMethod)	(optional, defaults to 'PayPal') Selects which payment flow is rendered by PayPal - the standard PayPal login page or the guest-checkout page for paying using a credit card.

## PayPal.Checkout.Item object

This is an immutable object that describes the item being purchased. It can be created using [PayPal.Checkout.ItemBuilder](#).

### Properties

<i>Name (Type)</i>	<i>Description</i>
<b>name</b> (String)	The name of the item to be purchased. Should be limited to 127 single-byte characters.
<b>description</b> (String)	(optional) The description of the item to be purchased. Should be limited to 127 single-byte characters.
<b>id</b> (String)	(optional) The ID of the item to be purchased. Should be limited to 127 single-byte characters.
<b>price</b> (String)	The price of the item to be purchased. Must be a positive number with 2 decimal places. The decimal separator must be a period (.), and the optional thousands separator must be a comma (,).
<b>tax</b> (String)	(optional) The tax on the item to be purchased. . Must be a positive number with 2 decimal places. The decimal separator must be a period (.), and the optional thousands separator must be a comma (,).
<b>quantity</b> (Number)	(optional, defaults to 1) Quantity of this item to be purchased. Must be a positive integer value.
<b>metadata</b> (Object)	(optional) Arbitrary data used to describe the item. (e.g. setting "type" to "Digital" will mark this item as a Digital goods purchase.)

## PayPal.Checkout.ItemBuilder object

This object is a builder used to create item objects.

### Methods

<b>Name(String name):</b> ItemBuilder	Sets the name of the item to be purchased.
<b>Description(String description):</b> ItemBuilder	Sets the description of the item to be purchased.
<b>Id(String id):</b> ItemBuilder	Sets the ID of the item to be purchased.
<b>Price(String price):</b> ItemBuilder	Sets the price of the item to be purchased.
<b>Tax(String tax):</b> ItemBuilder	Sets the tax on the item to be purchased.
<b>Quantity(Number quantity):</b> ItemBuilder	Sets the quantity of this item to be purchased. Defaults to 1.
<b>AddMetadata(String key, String value):</b> ItemBuilder	Adds arbitrary data relating to the item to be purchased. <b>Note:</b> This mainly aids in normalizing APIs and only specific keys are used. For example, setting "type" to "Digital" marks the item as a Digital Goods item.
<b>Build():</b> Item	Builds a PayPal.Checkout.Item object from the provided configuration options.

# PayPal Checkout SDK for Windows 8

## PayPal Checkout JavaScript API

JavaScript/HTML applications can decorate their markup to easily integrate the UI Control, or programmatically instrument their application using the API below.

The PayPal namespace currently contains the following objects:

- [PayPalJS.UI.BuyNow](#) - The WinJS UI Control
- [PayPal.Checkout.BuyNow](#) - The PayPal Checkout API
- [PayPal.Checkout.Item](#) - A model object used to describe an item being purchased
- [PayPal.Checkout.ItemBuilder](#) - The builder used to create an Item
- [PayPalJS.config](#) - A utility for managing PayPal configuration

## PayPalJS.UI.BuyNow object

This object is a custom WinJS control that enables developers to perform PayPal checkout transactions. To begin, include the paypal.js file in your application. This makes the PayPalJS namespace available to your application.

```
<script src="ms-appx://PayPal.Checkout/js/paypal.js"></script>
```

To use the control declaratively, just decorate your element as you would any other:

```
<button data-win-control="PayPalJS.UI.BuyNow"
        data-win-options="{ item: { name: 'My Product', id: 'a00001', price: '0.99' } }">
    Buy Now
</button>
```

## Syntax

### HTML

```
<button data-win-control="PayPalJS.UI.BuyNow"
        data-win-options="{ item: {name: 'My Product', id: 'a00001', description: 'My
awesome product', price: '0.99', 'type': 'Digital'} }">
    Buy Now
</button>
```

### JavaScript

```
var buyNow = new PayPalJS.UI.BuyNow(element);
buyNow.item = item;
```

## Events

Events dispatched by [PayPalJS.UI.BuyNow](#) can provide additional data via the `event.detail` property.

<i>Event</i>	<i>Description</i>	<i>event.detail</i>
<b>onstart</b>	Occurs when a transaction has been initiated.	Start
<b>onauth</b>	Occurs when authorization is triggered (e.g. WAB is displayed).	Auth: token
<b>oncancel</b>	Occurs when a transaction has been canceled by the user. <b>Note:</b> <i>complete</i> and <i>error</i> will not fire.	Cancel: token
<b>oncomplete</b>	Occurs when a transaction has successfully been completed by the user. <b>Note:</b> <i>cancel</i> and <i>error</i> will not fire.	Complete: token, transactionId, payerId
<b>onerror</b>	Occurs when a transaction is unable to be completed due to an error. <b>Note:</b> <i>cancel</i> and <i>complete</i> will not fire.	Error: message, [status], [code]

## Methods

<b>BuyNow(HTMLElement element, [Object options]): BuyNow</b>	Initializes a new instance of the BuyNow control.
<b>execute(): boolean</b>	Initiates a purchase transaction using the configuration provided via the arguments of the options constructor. Note that is an asynchronous method.
<b>addEventListener(String type, Function handler): undefined</b>	Adds an event listener. <a href="http://msdn.microsoft.com/en-us/library/windows/apps/br211690.aspx">http://msdn.microsoft.com/en-us/library/windows/apps/br211690.aspx</a>
<b>dispatchEvent(String type, [Object args]): boolean</b>	Raises an event of the specified type and with additional properties. Returns true if preventDefault was called on the event, otherwise returns false. <a href="http://msdn.microsoft.com/en-us/library/windows/apps/br211692.aspx">http://msdn.microsoft.com/en-us/library/windows/apps/br211692.aspx</a>

<b>removeEventListener(String type, Function handler):</b> undefined	Removes a listener for the specified event. <a href="http://msdn.microsoft.com/en-us/library/windows/apps/br211695.aspx">http://msdn.microsoft.com/en-us/library/windows/apps/br211695.aspx</a>
--	--

## Properties

<i>Name (Type)</i>	<i>Description</i>
<b>disabled</b> (bool)	Gets or set the disabled state of the control. Disabled if true, enabled if false.
<b>useSandbox</b> (bool)	Set to true to use the PayPal <a href="#">Sandbox</a> environment, false to use the live PayPal environment. Defaults to false.
<b>item</b> (PayPal.Checkout.Item)	Sets the PayPal.Checkout.Item for the current purchase.

## Styling

<i>CSS Class</i>	<i>Description</i>
paypal-ui-buynow	Styles the entire BuyNow control.

## Code Samples

The following code shows how to declaratively instantiate the [BuyNow](#) control.

```
<button id="buyNow1"
    data-win-control="PayPalJS.UI.BuyNow"
    data-win-options="{ item: { name: 'My Product', id: 'a00001', price: '0.99', 'type':
'Digital' } }">
    Buy Now
</button>
<script type="text/javascript">
    WinJS.UI.processAll();
</script>
```

The following code shows how to programmatically instantiate a [BuyNow](#) control.

```
<button name="buyNow2">Buy Now</button>
var buyNowElement = document.getElementById('buyNow2');

// var buyNow = new PayPalJS.UI.BuyNow(buyNowElement,
//   { item: { name: 'My Product', id: 'a00001', price: '0.99', 'type': 'Digital' }
// });
// - or -

var buyNow = new PayPalJS.UI.BuyNow(buyNowElement);
```

```
buyNow.addItem(item);
```

## PayPal.Checkout.BuyNow object

This is the main PayPal checkout API. You can code against this if you have your own UI control or other mechanism to trigger purchases.

### Syntax

```
var buyNow = new PayPal.Checkout.BuyNow("{myMerchantId}");
buyNow.addItem(item);

buyNow.addEventListener('start',    function () {});
buyNow.addEventListener('auth',    function () {});
buyNow.addEventListener('error',    function () {});
buyNow.addEventListener('cancel',   function () {});
buyNow.addEventListener('complete', function () {});

buyNow.execute();
```

### Constructor

**BuyNow(String merchantId)** Creates an instance of the BuyNow object for the provided merchantId.

### Events

<i>Event</i>	<i>Description</i>	<i>event</i>
<b>onstart</b>	Occurs when a transaction is initiated.	Start
<b>onauth</b>	Occurs when an authorization is triggered (e.g. WAB is displayed).	Auth: token
<b>oncancel</b>	Occurs when a user cancels the transaction. <b>Note:</b> <i>complete</i> and <i>error</i> will not fire.	Cancel: token
<b>oncomplete</b>	Occurs when a user successfully completes a transaction. <b>Note:</b> <i>cancel</i> and <i>error</i> will not fire.	Complete: token, transactionId, payerId
<b>onerror</b>	Occurs when a transaction is unable to be completed due to an error. <b>Note:</b> <i>cancel</i> and <i>complete</i> will not fire.	Error: message, [status], [code]

## Methods

<b>execute()</b>	Initiates a purchase transaction based on the information set on the BuyNow object. Note that is an asynchronous method.
<b>addItem(PayPal.Checkout.Item item)</b>	Adds a PayPal.Checkout.Item to the current purchase.
<b>clearItems()</b>	Clears all items from the current purchase.

## Properties

<b>Name (Type)</b>	<b>Description</b>
<b>useSandbox</b> (Boolean)	Set to true to use the PayPal <a href="#">Sandbox</a> environment, false to use the live PayPal environment. Defaults to false.
<b>merchantId</b> (String)	The merchant ID for the current transaction.
<b>displayShipping</b> (Number)	(optional, defaults to 0) Indicates how to display shipping options: 0 = Show the shipping address. 1 = Hide the shipping address. 2 = Use the address from the buyers profile if one isn't provided.  <b>Note:</b> This property is automatically set to 1 for Digital Goods transactions.
<b>currency</b> (String)	(optional, defaults to 'USD') Set to the three-character ISO-4217 currency code for this purchase. See table 2 in <a href="https://developer.paypal.com/webapps/developer/docs/classic/api/currency_codes/">https://developer.paypal.com/webapps/developer/docs/classic/api/currency_codes/</a> for full list of supported currencies.
<b>IPN</b> (String)	(optional) An <a href="#">Instant Payment Notification</a> URL to which PayPal should POST purchase information.
<b>custom</b> (String)	(optional) An arbitrary custom string value to associate with the current transaction. (NOTE: This field is limited to 256 single-byte alphanumeric characters.)
<b>locale</b> (String)	(optional, defaults to 'en_US') Set to the five-character locale code to control the locale of PayPal pages.

<b>paymentMethod</b> (Paypal.Checkout.PaymentMethod)	(optional, defaults to 'PayPal') Selects which payment flow is rendered by PayPal - the standard PayPal login page or the guest-checkout page for paying using a credit card.
---	---

## PayPal.Checkout.Item object

This is an immutable object that describes the item being purchased. It can be created using [PayPal.Checkout.ItemBuilder](#).

### Properties

<i>Name (Type)</i>	<i>Description</i>
<b>name</b> (String)	The name of the item to be purchased. Should be limited to 127 single-byte characters.
<b>description</b> (String)	(optional) The description of the item to be purchased. Should be limited to 127 single-byte characters.
<b>id</b> (String)	(optional) The ID of the item to be purchased. Should be limited to 127 single-byte characters.
<b>price</b> (String)	The price of the item to be purchased. Must be a positive number with 2 decimal places. The decimal separator must be a period (.), and the optional thousands separator must be a comma (,).
<b>tax</b> (String)	(optional) The tax on the item to be purchased. . Must be a positive number with 2 decimal places. The decimal separator must be a period (.), and the optional thousands separator must be a comma (,).
<b>quantity</b> (Number)	(optional, defaults to 1) Quantity of this item to be purchased. Must be a positive integer value.
<b>metadata</b> (Object)	(optional) Arbitrary data used to describe the item. (e.g. setting "type" to "Digital" will mark this item as a Digital goods purchase.)

## PayPal.Checkout.ItemBuilder object

This object is a builder used to create item objects.

### Syntax

```
var item = new PayPal.Checkout.ItemBuilder('My product')
    .id('a0001')
```



```

    .price('0.99')
    .description('An awesome product')
    .addMetadata('type', 'Digital');

```

## Methods

Signature	Returns	Description
<b>name(String name)</b>	ItemBuilder	Sets the name of the item to be purchased.
<b>description(String desc)</b>	ItemBuilder	Sets the description of the item to be purchased.
<b>id(String id)</b>	ItemBuilder	Sets the ID of the item to be purchased.
<b>price(String price)</b>	ItemBuilder	Sets the price of the item to be purchased.
<b>tax(String tax)</b>	ItemBuilder	Sets the tax on the item to be purchased.
<b>quantity(Number quantity)</b>	ItemBuilder	Sets the quantity of this item to be purchased. Defaults to 1.
<b>addMetadata(String key, String value)</b>	ItemBuilder	Adds arbitrary data relating to the item to be purchased. <b>Note:</b> This mainly aids in normalizing APIs and only specific keys are used. For example, setting "type" to "Digital" marks the item as a Digital Goods item.
<b>build()</b>	Item	Builds a PayPal.Checkout.Item object from the provided configuration options.

## PayPalJS.config object

This utility object provides the PayPal namespace with the associated configuration settings.

### Syntax

```

// Set values via 'put'. 'put' returns a Promise for async config situations.
PayPalJS.config.put('mykey', 'myvalue'); // Primitives
PayPalJS.config.put('myobject', {mykey2, 'myvalue2'}); // complex objects
PayPalJS.config.put({settingEnabled, true}); // initialization objects
PayPalJS.config.put(new Uri('ms-appx:///mysettings.json')); // json configuration files

```

```

// Reading values
PayPalJS.config.read('mykey'); // 'myvalue'
PayPalJS.config.read('myobject').mykey2; // 'myvalue2'
PayPalJS.config.read('settingEnabled'); // true

```

```
// Removing values
PayPalJS.config.del('mykey'); // true
PayPalJS.config.put('myobject', undefined); // true
PayPalJS.config.del('settingEnabled'); // true
```

## Methods

Signature	Returns	Description
<b>put(String key, * value)</b> <b>put(URI uri)</b> <b>put(Object data)</b>	WinJS.Promise	Adds arbitrary values to the PayPal configuration object and returns a Promise.  Once the value is added, the complete callback is called with the single argument being the added value.
<b>read(String key)</b>	*	Reads the value out of the configuration object for a given key.
<b>del(String key)</b>	Boolean	Deletes the value out of the configuration object for a given key.

## Code Samples

The following code shows different ways to add values to PayPalJS.config. Keep in mind that put is asynchronous; the example below executes the command serially for demonstration purposes only.

```
// Basic key/value
PayPalJS.config.put('showDescriptions', true)
    .then(function complete(showDescriptions) {
        console.log(showDescriptions); //true
        // config now contains: showDescriptions
    });

// Initialization object
PayPalJS.config.put({showDetails: true, title: 'My App'})
    .then(function complete(data) {
        console.dir(data); // { "showDetails": true, "title": "My App"}
        // config now contains: showDescriptions, showDetails, title
    });

// Adding objects
PayPalJS.config.put('key', { value: 'value1' })
    .then(function complete(key) {
        console.dir(key); // { "value" : "value1" }
        // config now contains: showDescriptions, showDetails, title, key
    });

// Removing key/values
PayPalJS.config.put('showDescriptions', undefined)
    .then(function complete(value) {
        console.log(value); // undefined (key/value was removed if it existed)
```

```

        // config now contains: showDetails, title, key
    });

    // Initialization via JSON
    PayPalJS.config.put(new Windows.Foundation.Uri('ms-appx:///paypal-settings.json'))
        .then(function complete(json) {
            console.dir(json); // the JSON data read from the provided file
            // config now contains: showDetails, title, key, and all key/values specified in
JSON
        });

    // Initializing keys as JSON. This adds the contents of the JSON file at the key 'props'
    PayPalJS.config.put('props', new Windows.Foundation.Uri('ms-appx:///paypal-settings.json'))
        .then(function complete(props) {
            console.dir(props); // the JSON data read from the provided file
            // config now contains: showDetails, title, key, props, and all key/values specified
in JSON
        });

```

Reading values out of config is straightforward via the read method:

```

var showDescriptions = PayPalJS.config.read('showDescriptions');
console.log(showDescriptions); // undefined (remember, it was deleted in the last example)

var showDetails = PayPalJS.config.read('showDetails'),
    title = PayPalJS.config.read('title');

console.log(showDetails); // true
console.log(title);       // "My App"

var key = PayPalJS.config.read('key');
console.dir(key); // { "value" : "value1" }

```

Deleting values is also easy via del:

```

PayPalJS.config.del('showDescriptions'); // false (already deleted)
PayPalJS.config.del('showDetails');     // true
PayPalJS.config.del('title');           // true

// PayPalJS.config currently supports only top-level keys.
// To delete a nested property, you need to get the object and delete it manually.
var key = PayPalJS.config.read('key');
key.value = undefined;
PayPalJS.config.read('key'); // {}

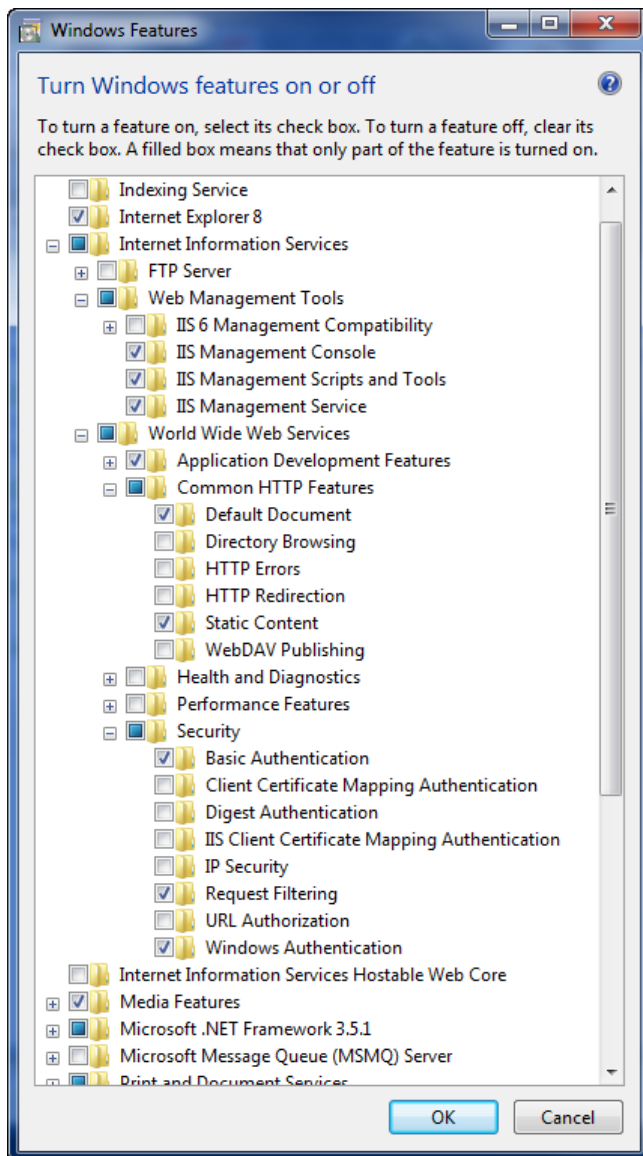
```

## **Classic ASP and C# COM**

### **Debugging Classic ASP in Visual Studio 2005/2008/2010**

Windows 7 - Control Panel >> Programs >> Turn Windows Features on or off

- Under Internet Information Services, enable IIS Management Console, IIS Management Scripts & Tools, and IIS Management Service.
- Under World Wide Web Services:
  - Application Development Features - enable everything
  - Common HTTP Features - enable Default Document and Static Content
  - Security - enable Basic Authentication, Request Filtering and Windows Authentication
- Click OK



In IIS, add a website that points to the physical location of your files. I'd recommend using a specific port for running the site - see the example below for creating a site named TestApp running on port 1422 - you'll use `http://localhost:1422` to browse to the site. Note that an Application Pool is created with the same name - TestApp.

**Add Web Site**

Site name:  Application pool:

**Content Directory**

Physical path:

Pass-through authentication

**Binding**

Type:  IP address:  Port:


Host name:

Example: www.contoso.com or marketing.contoso.com

☒ Start Web site immediately

Next, in IIS Manager Features View, double-click Default Document and ensure the Default.asp is in the list. Backup to the Features View and double-click - ASP icon.

- Enable Parent Paths - set to True
- Expand Debugging Properties
  - Enable Client-side Debugging - set to True
  - Enable Server-side Debugging - set to True
  - Send Errors to Browser - set to True

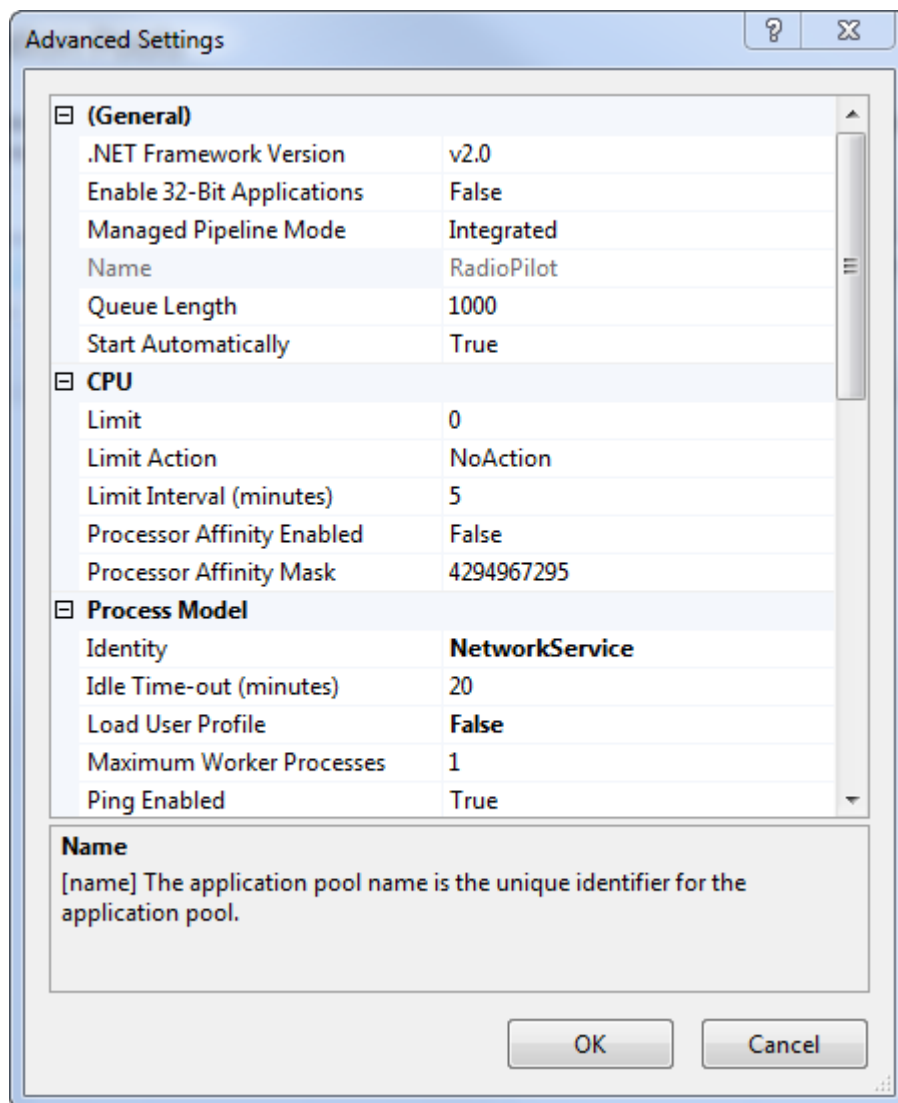

**ASP**

Display: **Friendly Names**

<b>Behavior</b>	
Code Page	0
Enable Buffering	True
Enable Chunked Encoding	True
Enable HTML Fallback	True
Enable Parent Paths	<b>True</b>
<b>Limits Properties</b>	
Locale ID	0
Restart On Config Change	True
<b>Compilation</b>	
<b>Debugging Properties</b>	
Calculate Line Numbers	True
Catch COM Component Exceptions	True
Enable Client-side Debugging	<b>True</b>
Enable Log Error Requests	True
Enable Server-side Debugging	<b>True</b>
Log Errors to NT Log	False
Run On End Functions Anonymously	True
Script Error Message	<b>An error occurred on the server when processing the script.</b>
Send Errors To Browser	<b>True</b>
Script Language	VBScript
<b>Services</b>	
<b>Caching Properties</b>	
<b>Global Assemblies</b>	

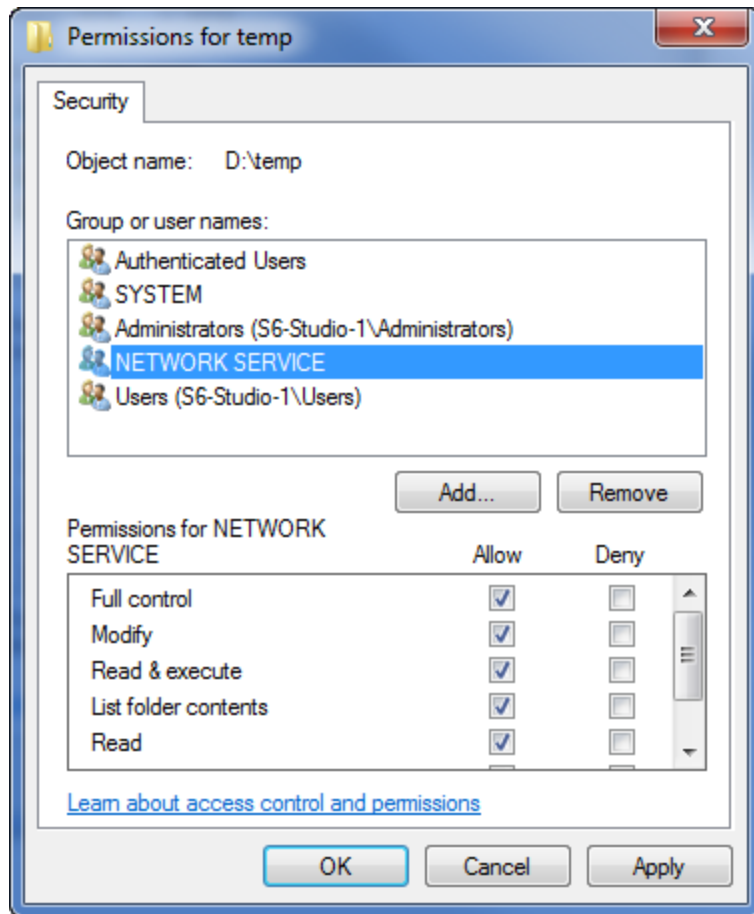
**Debugging Properties**  
 Properties related to debug settings.

Now go back to IIS Connections and select the Application Pools node, select the Application Pool you created for your site and then select Advance Settings. In Advanced Settings, change the Identity to NetworkService (this makes it easy to set you security next).

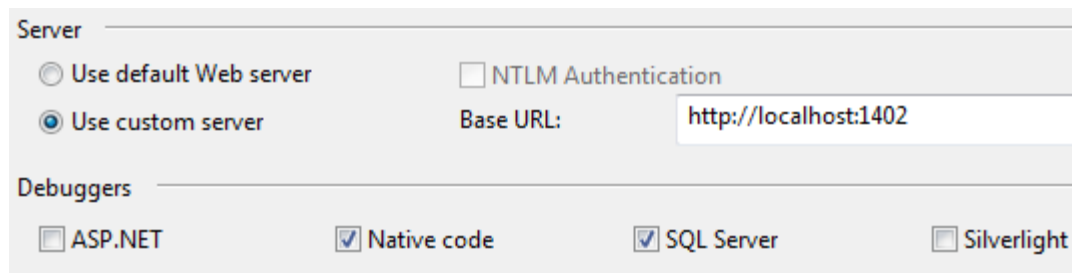


Now we need to give the User "Network Service" full permissions on the physical site. Go to Explorer, right-click on the top folder, select Properties >> Security. Add the user NETWORK SERVICE and assign full control.

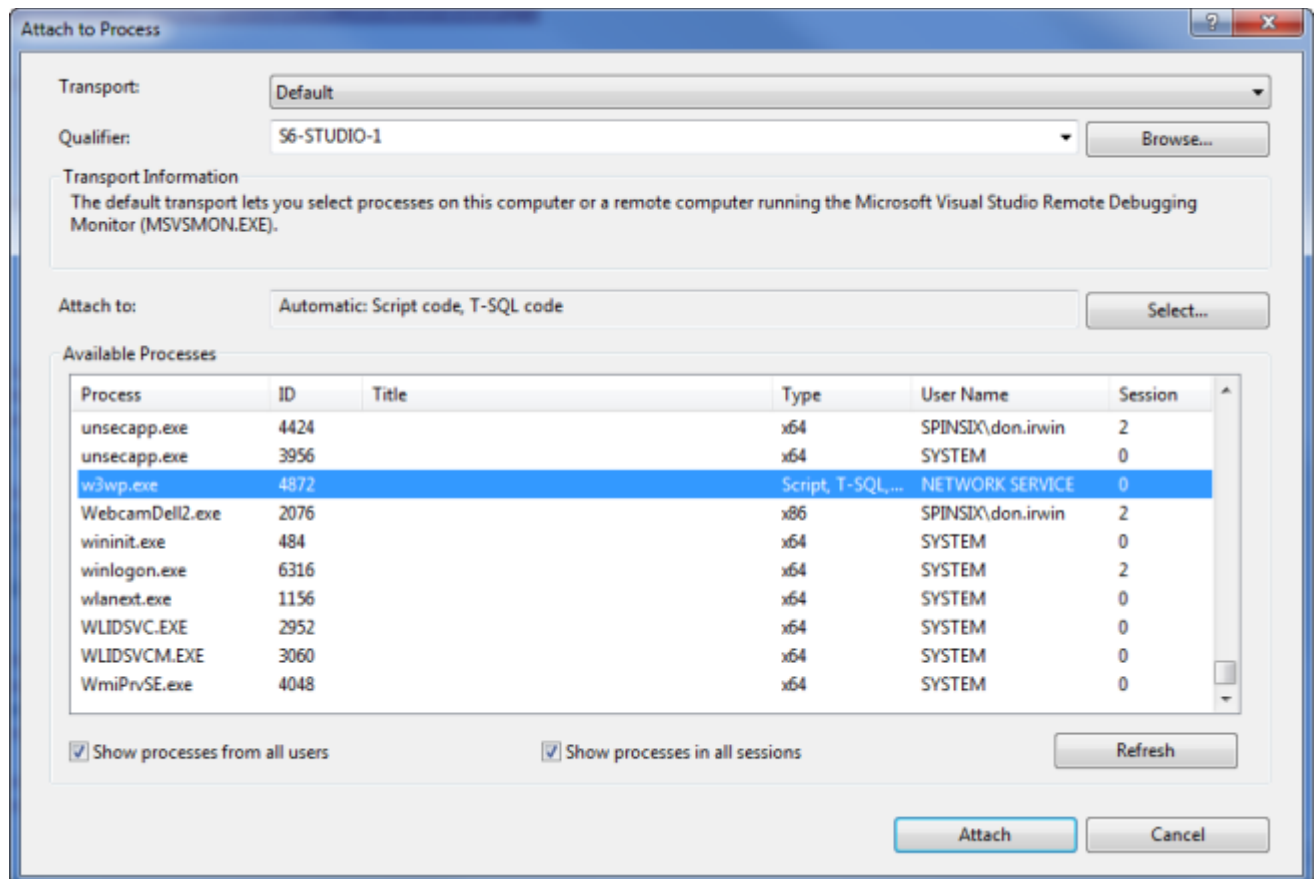




Finally to Visual Studio 2010 - run Visual Studio as an administrator! Then, open your website - File >> Open Web Site >> File System and select the root folder then click open. Now in Solution Explorer, right-click on the site root and select Property Pages. Under Build, select "No Build", then under Start Options >> Server, select "Use custom server" and enter the base URL with the port number for the site you created earlier: `http://localhost:1422` - click OK.



Now run your site - Debug (F5) - either the default page or the page you specified in Properties >> Start Options should launch in a browser now. Go back to Visual Studio 2010 and select Debug >> Attach to Process. At the bottom of the dialog, select bot "Show Processes..." check boxes - then in the "Available Processes" list, scroll to the bottom and select w3wp.exe and then click "Attach". If you get a warning select YES/Continue.



You should now be able to set breakpoints in your code and step through the site.

**C# Example Code Snippet (ClassicASPCOM.dll) built with Strong Name and Registered for COM Interop in Visual Studio 2005 (.NET Framework 2.0) and Windows 7**

```
using System.Runtime.InteropServices;

namespace ComNamespace1
{
    [Guid("EAA4976A-45C3-4BC5-BC0B-E474F4C3C83F")]
    public interface ComClass1Interface
    {
        [DispId(1)]
        string GetMessage(string Message);
    }

    [Guid("7BD20046-DF8C-44A6-8F6B-687FAA26FA71"),
        InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]
    public interface ComClass1Events
    {
    }

    [Guid("0D53A3E8-E51A-49C7-944E-E72A2064F938"),
        ClassInterface(ClassInterfaceType.None),
        ComSourceInterfaces(typeof(ComClass1Events))]
    public class ComClass1 : ComClass1Interface
    {
        public string GetMessage(string Message)
        {
            return "Your Message: " + Message;
        }
    }
}
```

### **Exporting to COM to be called from Classic ASP**

- Exposing Visual C# objects to COM requires declaring a class interface, an events interface if it is required, and the class itself. Class members must follow these rules to be visible to COM
- The class must be public
- Properties, methods, and events must be public
- Properties and methods must be declared on the class interface
- Events must be declared in the event interface
- COM does not support static methods  
This rule ignores property and event accessors, operator overloading methods, or methods that are marked by using either the `System.Runtime.InteropServices.ComRegisterFunctionAttribute` attribute or the `System.Runtime.InteropServices.ComUnregisterFunctionAttribute` attribute
- By default, the following are visible to COM: assemblies, public types, public instance members in public types, and all members of public value types
- Other public members in the class that are not declared in these interfaces will not be visible to COM, but they will be visible to other .NET Framework objects
- To expose properties and methods to COM, you must declare them on the class interface and mark them with a `DispId` attribute, and implement them in the class. The order in which the members are declared in the interface is the order used for the COM vtable
- To expose events from your class, you must declare them on the events interface and mark them with a `DispId` attribute. The class should not implement this interface
- The class implements the class interface; it can implement more than one interface, but the first implementation will be the default class interface. Implement the methods and properties exposed to COM here. They must be marked public and must match the declarations in the class interface. Also, declare the events raised by the class here. They must be marked public and must match the declarations in the events interface

### **Register Component for COM Interop**

- With the project node selected in Solution Explorer, from the Project menu, click Properties (or right-click the project node in Solution Explorer, and click Properties)
- In the Project Designer, click the Build tab
- Select the Register for COM interop check box

### **Sign assembly**

- With the project node selected in Solution Explorer, from the Project menu, click Properties (or right-click the project node in Solution Explorer, and click Properties)
- In the Project Designer, click the Signing tab
- Select the Sign the assembly check box
- Specify a new key file. In the Choose a strong name key file drop-down list, select <New...>. Note that new key files are always created in the .pfx format
- The Create Strong Name Key Dialog Box appears
- In the Create Strong Name Key dialog box, enter a name and password for the new key file, and then click OK

### **Create a Public/Private Key Pair (Strong Name)**

- Create a strong-name key (cryptographic key pair) and store it in the file PayPalAPCOMKeyPair.snk  
sn -k PayPalAPCOMKeyPair.snk
- Extract the public key from PayPalAPCOMKeyPair.snk and put it into PayPalAPCOMPublicKey.snk  
sn -p PayPalAPCOMKeyPair.snk PayPalAPCOMPublicKey.snk  
sn -tp PayPalAPCOMPublicKey.snk
- Get the public key stored in the file PayPalAPCOMPublicKey.snk  
sn -tp PayPalAPCOMPublicKey.snk

Public key is

```
00240000048000000940000000602000000240000525341310004000001000100852fb0f89f9ad1
8db88b3f6ec7c2c475bd079c9e6d16293257986b7aed572b08cfd994b92a594445285d384b187f
8f426d102a03dce3699b022b2c6ed17a90c9f2db2306179b1c09038e91a9249a1c7ea7732f854e
25a919da7a924733bd90a9be39ad0c6c211470fa4f5813bcb11c50b0eca90198aed042e79569a7
d67ebfc0
```

Public key token is 9cdbbf45e4ede757

#### **COM Installation Batch Script**

```
Regasm ClassicASPCOM.dll
Regasm ClassicASPCOM.dll /codebase
Regasm ClassicASPCOM.dll /tlb
gacutil/i ClassicASPCOM.dll
```

#### **COM Uninstallation Batch Script**

```
iisreset
Regasm/u ClassicASPCOM.dll
Regasm/u ClassicASPCOM.dll /tlb
gacutil/u ClassicASPCOM.dll
```

### 'VBScript (ClassicASPHome.asp)

```
<%  
Response.Write("Classic ASP Page:")  
%>  
<br />  
<br />  
<%  
Dim com  
Set com = Server.CreateObject("ComNamespace1.ComClass1")  
If TypeName(com.error) <> "String" Then  
    Response.Write(com.GetMessage("Hello World!"))  
Else  
%>  
    Exception message:  
<%  
    Response.Write com.error  
End if  
%>
```

## Certificate

Log In to Developer website <https://developer.paypal.com/>

PayPal account:

#####@email.com

PayPal account password:

\*\*\*\*\*

API username:

####\*@email.com

API password:

\*\*\*\*\*

<https://www.sandbox.paypal.com/>

\*####\*@email.com

\*\*\*\*\*

```
Dim CertificateSubjectDistinguishedName
CertificateSubjectDistinguishedName = "C=US, S=TX, L=Austin, O=SDK
Seller, CN= ####*@email.com"
```

```
Dim configHashtable
Set configHashtable = Server.CreateObject("System.Collections.Hashtable")
configHashtable.Add "mode", "sandbox"
configHashtable.Add "account1.apiUsername", "####*@email.com"
configHashtable.Add "account1.apiPassword", "*****"
configHashtable.Add "account1.applicationId", "##-*****"
configHashtable.Add "account1.apiCertificate",
CertificateSubjectDistinguishedName
configHashtable.Add "account1.privateKeyPassword", "*****"
```

```
>openssl pkcs12 -export -in cert_key_pem.txt -inkey cert_key_pem.txt -out cert_key.p12
```

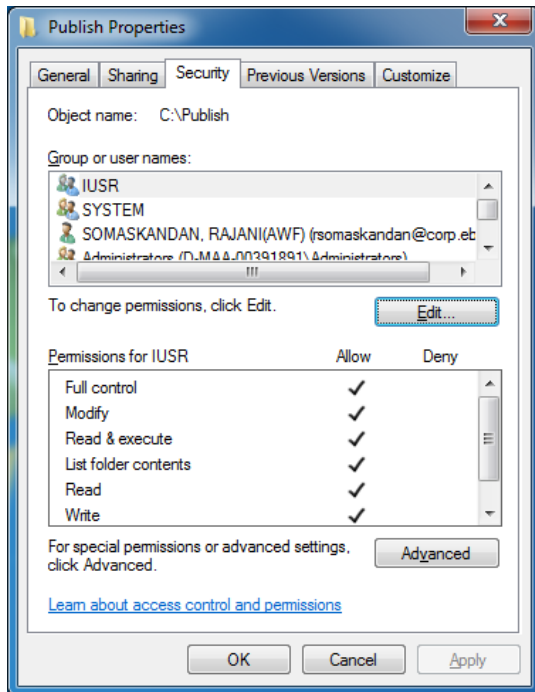
Loading 'screen' into random state - done

Enter Export Password:

Verifying - Enter Export Password:

## Publish Classic ASP

- Ensure IUSR is added to the publish folder



- Ensure IIS\_IUSRS is added to the publish folder



