ComponentOne

PayPal for ASP.NET

Corporate Headquarters
ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com
Web site: http://www.componentone.com

Sales
E-mail: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of ComponentOne LLC. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the original CD (or diskettes) are free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective CD (or disk) to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for a defective CD (or disk) by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. We are not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne PayPal eCommerce for ASP.NET Overview 1
- What's New in PayPal eCommerce for ASP.NET 1
- Installing PayPal eCommerce for ASP.NET 1
- PayPal eCommerce for ASP.NET Setup Files 1
- System Requirements 2
- Installing Demonstration Versions 2
- Uninstalling PayPal eCommerce for ASP.NET 3
- Deploying your Application in a Medium Trust Environment 3
- End-User License Agreement 6
- Licensing FAQs 6
- What is Licensing? 6
- How does Licensing Work? 6
- Common Scenarios 7
- Troubleshooting 9
- Technical Support 11
- Redistributable Files 11
- About This Documentation 12
- Namespaces 12
- Creating an ASP.NET Project 13
- Creating a Web Site Project 13
- Creating a Web Application Project 14
- Adding the PayPal eCommerce for ASP.NET Components to a Project 15
- Migrating a PayPal eCommerce for ASP.NET Project to Visual Studio 2005 16

Key Features 17

Getting Started with PayPal eCommerce for ASP.NET 20
- PayPal eCommerce for ASP.NET Controls 20
- C1PurchaseItem Control 20
- C1AddToCart Control 20
- C1CartCheckout Control 20
- How PayPal eCommerce for ASP.NET Controls Work 20
- Setting Up Your Web Form 21
- Using the Properties Window 22
- Using the C1PayPal Wizard 22
- Entering Your Business Information 22
- Customizing Button Appearance and Navigation 23
- Entering the Product Information 24
- Entering Shipping and Handling Information 25
- Customizing Navigation After the Transaction 26

PayPal eCommerce for ASP.NET Samples 27

Building a Web Store 28
- Using Unbound PayPal eCommerce for ASP.NET Controls 28
- Using Data-Bound PayPal eCommerce for ASP.NET Controls 32
- Step 1 of 4: Creating and Formatting the DataList Control 32
- Step 2 of 4: Adding a Data Source to the Form 33
- Step 3 of 4: Binding the Controls to a Data Source 34
- Step 4 of 4: Running the Application 36

ComponentOne PayPal eCommerce for ASP.NET Overview

ComponentOne PayPal eCommerce™ for ASP.NET is a suite of Web controls for Microsoft ASP.NET that allows you to sell items on-line using services provided by PayPal (http://www.paypal.com/). Using PayPal, **PayPal eCommerce for ASP.NET** provides real-time credit card processing for ASP.NET websites and applications with three eCommerce wizards that create payment buttons for single-item purchases (C1PurchaseItem) or multiple-item purchases (C1AddToCart, C1CartCheckOut). With **PayPal eCommerce for ASP.NET**, you can easily turn your Web site into an e-commerce enabled store.

Getting Started

Get started with the following topics:

- Key Features
- About the Controls
- The C1PayPal Wizard
- Building a Web Store
- Samples

What's New in PayPal eCommerce for ASP.NET

This documentation was last revised on August 12, 2009. There were no new features added to **PayPal eCommerce for ASP.NET**.

Tip: A version history containing a list of new features, improvements, fixes, and changes for each product is available on HelpCentral at http://helpcentral.componentone.com/VersionHistory.aspx.

Installing PayPal eCommerce for ASP.NET

The following sections provide helpful information on installing **ComponentOne PayPal eCommerce for ASP.NET**.

PayPal eCommerce for ASP.NET Setup Files

The **ComponentOne Studio for ASP.NET** installation program will create the following directory: **C:\Program Files\ComponentOne\Studio for ASP.NET**. This directory contains the following subdirectories:

bin	Contains copies of all binaries (DLLs, Exes) in the ComponentOne Visual Studio ASP.NET package.
H2Help	Contains documentation for **Studio for ASP.NET** components.

C1PayPal	Contains files (at least a readme.txt) related to the product.

Samples

Samples for the product are installed in the **ComponentOne Samples** folder by default. The path of the ComponentOne Samples directory is slightly different on Windows XP and Windows Vista machines:

Windows XP path: C:\Documents and Settings\<username>\My Documents\ComponentOne Samples

Windows Vista path: C:\Users\<username>\Documents\ComponentOne Samples

The **ComponentOne Samples** folder contains the following subdirectories:

Common	Contains support and data files that are used by many of the demo programs.
C1PayPal	Contains samples and tutorials for **PayPal eCommerce for ASP.NET**.

System Requirements

System requirements include the following:

Operating Systems:	Windows 2000
	Windows Server® 2003
	Windows Server® 2008
	Windows Server 2008
	Windows XP SP2
	Windows Vista™
	Windows 7
Web Server:	Microsoft Internet Information Services (IIS) 5.0 or later
Environments:	.NET Framework 2.0 or later
	Visual Studio 2005
	Visual Studio 2008
	Internet Explorer® 6.0 or later
	Firefox® 2.0 or later
	Safari® 2.0 or later
Disc Drive:	CD or DVD-ROM drive if installing from CD

Installing Demonstration Versions

If you wish to try **ComponentOne PayPal eCommerce for ASP.NET** and do not have a serial number, follow the steps through the installation wizard and use the default serial number.

The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that the registered version will stamp every application you compile so a ComponentOne banner will not appear when your users run the applications.

Uninstalling PayPal eCommerce for ASP.NET

To uninstall **ComponentOne PayPal eCommerce for ASP.NET**:

1. Open **Control Panel** and select **Add or Remove Programs** (**Programs and Features** in Vista).
2. Select **ComponentOne Studio for ASP.NET** and click the **Remove** button.
3. Click **Yes** to remove the program.

Deploying your Application in a Medium Trust Environment

Depending on your hosting choice, you may need to deploy your Web site or application in a medium trust environment. Often in a shared hosting environment, medium trust is required. In a medium trust environment several permissions are unavailable or limited, including OleDbPermission, ReflectionPermission,and FileIOPermission. You can configure your Web.config file to enable these permissions.

Note: ComponentOne controls will not work in an environment where reflection is not allowed.

ComponentOne ASP.NET controls include the AllowPartiallyTrustedCallers() assembly attribute and will work under the medium trust level with some changes to the Web.config file. Since this requires some control over the Web.config file, please check with your particular host to determine if they can provide the rights to override these security settings.

Modifying or Editing the Config File

In order to add permissions, you can edit the exiting web_mediumtrust.config file or create a custom policy file based on the medium trust policy. If you modify the existing web_mediumtrust.config file, all Web applications will have the same permissions with the permissions you have added. If you want applications to have different permissions, you can instead create a custom policy based on medium trust.

Edit the Config File

In order to add permissions, you can edit the exiting web_mediumtrust.config file. To edit the exiting web_mediumtrust.config file, complete the following steps:

1. Locate the medium trust policy file web_mediumtrust.config located by default in the %windir%\Microsoft.NET\Framework\{Version}\CONFIG directory.
2. Open the web_mediumtrust.config file.
3. Add the permissions that you want to grant. For examples, see Adding Permissions.

Create a Custom Policy Based on Medium Trust

In order to add permissions, you can create a custom policy file based on the medium trust policy. To create a custom policy file, complete the following steps:

1. Locate the medium trust policy file web_mediumtrust.config located by default in the %windir%\Microsoft.NET\Framework\{Version}\CONFIG directory.
2. Copy the web_mediumtrust.config file and create a new policy file in the same directory.

   Give the new a name that indicates that it is your variation of medium trust; for example, AllowReflection_Web_MediumTrust.config.
3. Add the permissions that you want to grant. For examples, see Adding Permissions.

4. Enable the custom policy file on your application by modifying the following lines in your web.config file under the `<system.web>` node:

```
<system.web>
<trust level="CustomMedium" originUrl=""/>

 <securityPolicy>
                <trustLevel name="CustomMedium"
policyFile="AllowReflection_Web_MediumTrust.config"/>
          </securityPolicy>
          ...
</system.web>
```

Note: Your host may not allow trust level overrides. Please check with your host to see if you have these rights.

Allowing Deserialization

To allow the deserialization of the license added to App_Licenses.dll by the Microsoft IDE, you should add the SerializationFormatter flag to security permission to the Web.config file. Complete the steps in the Modifying or Editing the Config File topic to create or modify a policy file before completing the following.

Add the `SerializationFormatter` flag to the `<IPermission class="SecurityPermission">` tag so that it appears similar to the following:

```
<NamedPermissionSets>
     <PermissionSet
     class="NamedPermissionSet"
     version="1"
     Name="ASP.Net">
        <IPermission
                class="SecurityPermission"
                version="1"
                Flags="Assertion, Execution, ControlThread,
ControlPrincipal, RemotingConfiguration, SerializationFormatter"/>
         ...
     </PermissionSet>
</NamedPermissionSets>
```

Adding Permissions

You can add permission, including ReflectionPermission, OleDbPermission, and FileIOPermission to the web.config file. Note that ComponentOne controls will not work in an environment where reflection is not allowed. Complete the steps in the Modifying or Editing the Config File topic to create or modify a policy file before completing the following.

ReflectionPermission

By default ReflectionPermission is not available in a medium trust environment. ComponentOne ASP.NET controls require reflection permission because LicenseManager.Validate() causes a link demand for full trust.

To add reflection permission, complete the following:

1. Open the web_mediumtrust.config file or a file created based on the web_mediumtrust.config file.
2. Add the following `<SecurityClass>` tag after the `<SecurityClasses>` tag so that it appears similar to the following:

```
<SecurityClasses>
    <SecurityClass Name="ReflectionPermission"
Description="System.Security.Permissions.ReflectionPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
```

```
...
</SecurityClasses>
```

3. Add the following <IPermission> tag after the <NamedPermissionSets> tag so it appears similar to the following:

```
<NamedPermissionSets>
     <PermissionSet class="NamedPermissionSet" version="1"
Name="ASP.Net">
        <IPermission
            class="ReflectionPermission"
            version="1"
            Flags="ReflectionEmit,MemberAccess" />
        ...
     </PermissionSet>
</NamedPermissionSets>
```

4. Save and close the web_mediumtrust.config file.

OleDbPermission

By default OleDbPermission is not available in a medium trust environment. This means you cannot use the ADO.NET managed OLE DB data provider to access databases. If you wish to use the ADO.NET managed OLE DB data provider to access databases, you must modify the web_mediumtrust.config file.

To add OleDbPermission, complete the following steps:

1. Open the web_mediumtrust.config file or a file created based on the web_mediumtrust.config file.
2. Add the following <SecurityClass> tag after the <SecurityClasses> tag so that it appears similar to the following:

```
<SecurityClasses>
    <SecurityClass Name="OleDbPermission"
Description="System.Data.OleDb.OleDbPermission, System.Data,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
...
</SecurityClasses>
```

3. Add the following <IPermission> tag after the <NamedPermissionSets> tag so it appears similar to the following:

```
<NamedPermissionSets>
     <PermissionSet class="NamedPermissionSet" version="1"
Name="ASP.Net">
        <IPermission class="OleDbPermission" version="1"
Unrestricted="true"/>
        ...
     </PermissionSet>
</NamedPermissionSets>
```

4. Save and close the web_mediumtrust.config file.

FileIOPermission

By default, FileIOPermission is not available in a medium trust environment. This means no file access is permitted outside of the application's virtual directory hierarchy. If you wish to allow additional file permissions, you must modify the web_mediumtrust.config file.

To modify FileIOPermission to allow read access to a specific directory outside of the application's virtual directory hierarchy, complete the following steps:

1. Open the web_mediumtrust.config file or a file created based on the web_mediumtrust.config file.
2. Add the following <SecurityClass> tag after the <SecurityClasses> tag so that it appears similar to the following:

```
<SecurityClasses>
    <SecurityClass Name="FileIOPermission"
Description="System.Security.Permissions.FileIOPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
...
</SecurityClasses>
```

3. Add the following `<IPermission>` tag after the `<NamedPermissionSets>` tag so it appears similar to the following:

```
<NamedPermissionSets>
    <PermissionSet class="NamedPermissionSet" version="1"
Name="ASP.Net">
    ...
    <IPermission class="FileIOPermission" version="1"
Read="C:\SomeDir;$AppDir$" Write="$AppDir$" Append="$AppDir$"
PathDiscovery="$AppDir$" />
  ...
</PermissionSet>
</NamedPermissionSets>
```

4. Save and close the web_mediumtrust.config file.

End-User License Agreement

All of the ComponentOne licensing information, including the ComponentOne end-user license agreements, frequently asked licensing questions, and the ComponentOne licensing model, is available online at http://www.componentone.com/SuperPages/Licensing/.

Licensing FAQs

This section describes the main technical aspects of licensing. It may help the user to understand and resolve licensing problems he may experience when using ComponentOne .NET and ASP.NET products.

What is Licensing?

Licensing is a mechanism used to protect intellectual property by ensuring that users are authorized to use software products.

Licensing is not only used to prevent illegal distribution of software products. Many software vendors, including ComponentOne, use licensing to allow potential users to test products before they decide to purchase them.

Without licensing, this type of distribution would not be practical for the vendor or convenient for the user. Vendors would either have to distribute evaluation software with limited functionality, or shift the burden of managing software licenses to customers, who could easily forget that the software being used is an evaluation version and has not been purchased.

How does Licensing Work?

ComponentOne uses a licensing model based on the standard set by Microsoft, which works with all types of components.

Note: The **Compact Framework** components use a slightly different mechanism for run-time licensing than the other ComponentOne components due to platform differences.

When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system. (Users can also

enter the serial number by clicking the **License** button on the **About Box** of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog box.)

When a licensed component is added to a form or Web page, Visual Studio obtains version and licensing information from the newly created component. When queried by Visual Studio, the component looks for licensing information stored in the system and generates a run-time license and version information, which Visual Studio saves in the following two files:

- An assembly resource file which contains the actual run-time license
- A "licenses.licx" file that contains the licensed component strong name and version information

These files are automatically added to the project.

In WinForms and ASP.NET 1.x applications, the run-time license is stored as an embedded resource in the assembly hosting the component or control by Visual Studio. In ASP.NET 2.x applications, the run-time license may also be stored as an embedded resource in the App_Licenses.dll assembly, which is used to store all run-time licenses for all components directly hosted by WebForms in the application. Thus, the App_licenses.dll must always be deployed with the application.

The licenses.licx file is a simple text file that contains strong names and version information for each of the licensed components used in the application. Whenever Visual Studio is called upon to rebuild the application resources, this file is read and used as a list of components to query for run-time licenses to be embedded in the appropriate assembly resource. Note that editing or adding an appropriate line to this file can force Visual Studio to add run-time licenses of other controls as well.

Note that the licenses.licx file is usually not shown in the Solution Explorer; it appears if you press the **Show All Files** button in the Solution Explorer's Toolbox, or from Visual Studio's main menu, select **Show All Files** on the **Project** menu.

Later, when the component is created at run time, it obtains the run-time license from the appropriate assembly resource that was created at design time and can decide whether to simply accept the run-time license, to throw an exception and fail altogether, or to display some information reminding the user that the software has not been licensed.

All ComponentOne products are designed to display licensing information if the product is not licensed. None will throw licensing exceptions and prevent applications from running.

Common Scenarios

The following topics describe some of the licensing scenarios you may encounter.

Creating components at design time

This is the most common scenario and also the simplest: the user adds one or more controls to the form, the licensing information is stored in the licenses.licx file, and the component works.

Note that the mechanism is exactly the same for Windows Forms and Web Forms (ASP.NET) projects.

Creating components at run time

This is also a fairly common scenario. You do not need an instance of the component on the form, but would like to create one or more instances at run time.

In this case, the project will not contain a licenses.licx file (or the file will not contain an appropriate run-time license for the component) and therefore licensing will fail.

To fix this problem, add an instance of the component to a form in the project. This will create the licenses.licx file and things will then work as expected. (The component can be removed from the form after the licenses.licx file has been created).

Adding an instance of the component to a form, then removing that component, is just a simple way of adding a line with the component strong name to the licenses.licx file. If desired, you can do this manually using

notepad or Visual Studio itself by opening the file and adding the text. When Visual Studio recreates the application resources, the component will be queried and its run-time license added to the appropriate assembly resource.

Inheriting from licensed components

If a component that inherits from a licensed component is created, the licensing information to be stored in the form is still needed. This can be done in two ways:

- Add a LicenseProvider attribute to the component.

  This will mark the derived component class as licensed. When the component is added to a form, Visual Studio will create and manage the licenses.licx file, and the base class will handle the licensing process as usual. No additional work is needed. For example:

```
    [LicenseProvider(typeof(LicenseProvider))]
    class MyGrid: C1.Win.C1FlexGrid.C1FlexGrid
    {
// ...
    }
```

- Add an instance of the base component to the form.

  This will embed the licensing information into the licenses.licx file as in the previous scenario, and the base component will find it and use it. As before, the extra instance can be deleted after the licenses.licx file has been created.

Please note, that C1 licensing will not accept a run-time license for a derived control if the run-time license is embedded in the same assembly as the derived class definition, and the assembly is a DLL. This restriction is necessary to prevent a derived control class assembly from being used in other applications without a design-time license. If you create such an assembly, you will need to take one of the actions previously described create a component at run time.

Using licensed components in console applications

When building console applications, there are no forms to add components to, and therefore Visual Studio won't create a licenses.licx file.

In these cases, create a temporary Windows Forms application and add all the desired licensed components to a form. Then close the Windows Forms application and copy the licenses.licx file into the console application project.

Make sure the licenses.licx file is configured as an embedded resource. To do this, right-click the licenses.licx file in the Solution Explorer window and select **Properties**. In the Properties window, set the **Build Action** property to **Embedded Resource**.

Using licensed components in Visual C++ applications

There is an issue in VC++ 2003 where the licenses.licx is ignored during the build process; therefore, the licensing information is not included in VC++ applications.

To fix this problem, extra steps must be taken to compile the licensing resources and link them to the project. Note the following:

1. Build the C++ project as usual. This should create an EXE file and also a licenses.licx file with licensing information in it.
2. Copy the licenses.licx file from the application directory to the target folder (Debug or Release).
3. Copy the C1Lc.exe utility and the licensed DLLs to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use C1Lc.exe to compile the licenses.licx file. The command line should look like this:

```
c1lc /target:MyApp.exe /complist:licenses.licx /i:C1.Win.C1FlexGrid.dll
```

5. Link the licenses into the project. To do this, go back to Visual Studio, right-click the project, select **Properties**, and go to the **Linker/Command Line** option. Enter the following:
```
/ASSEMBLYRESOURCE:Debug\MyApp.exe.licenses
```
6. Rebuild the executable to include the licensing information in the application.

Using licensed components with automated testing products

Automated testing products that load assemblies dynamically may cause them to display license dialog boxes. This is the expected behavior since the test application typically does not contain the necessary licensing information, and there is no easy way to add it.

This can be avoided by adding the string "C1CheckForDesignLicenseAtRuntime" to the AssemblyConfiguration attribute of the assembly that contains or derives from ComponentOne controls. This attribute value directs the ComponentOne controls to use design-time licenses at run time.

For example:

```
#if AUTOMATED_TESTING
   [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")]
#endif
   public class MyDerivedControl : C1LicensedControl
   {
         // ...
   }
```

Note that the AssemblyConfiguration string may contain additional text before or after the given string, so the AssemblyConfiguration attribute can be used for other purposes as well. For example:

```
   [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime,BetaVersion
")]
```

THIS METHOD SHOULD ONLY BE USED UNDER THE SCENARIO DESCRIBED. It requires a design-time license to be installed on the testing machine. Distributing or installing the license on other computers is a violation of the EULA.

Troubleshooting

We try very hard to make the licensing mechanism as unobtrusive as possible, but problems may occur for a number of reasons.

Below is a description of the most common problems and their solutions.

I have a licensed version of a ComponentOne product but I still get the splash screen when I run my project.

If this happens, there may be a problem with the licenses.licx file in the project. It either doesn't exist, contains wrong information, or is not configured correctly.

First, try a full rebuild (**Rebuild All** from the Visual Studio **Build** menu). This will usually rebuild the correct licensing resources.

If that fails follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the licenses.licx file and open it. If prompted, continue to open the file.
4. Change the version number of each component to the appropriate value. If the component does not appear in the file, obtain the appropriate data from another licenses.licx file or follow the alternate procedure following.

5. Save the file, then close the licenses.licx tab.
6. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

Alternatively, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the licenses.licx file and delete it.
4. Close the project and reopen it.
5. Open the main form and add an instance of each licensed control.
6. Check the Solution Explorer window, there should be a licenses.licx file there.
7. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

For ASP.NET 2.x applications, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Find the licenses.licx file and right-click it.
3. Select the Rebuild Licenses option (this will rebuild the App_Licenses.licx file).
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

I have a licensed version of a ComponentOne product on my Web server but the components still behave as unlicensed.

There is no need to install any licenses on machines used as servers and not used for development.

The components must be licensed on the development machine, therefore the licensing information will be saved into the executable (EXE or DLL) when the project is built. After that, the application can be deployed on any machine, including Web servers.

For ASP.NET 2.x applications, be sure that the App_Licenses.dll assembly created during development of the application is deployed to the bin application bin directory on the Web server.

If your ASP.NET application uses WinForms user controls with constituent licensed controls, the run-time license is embedded in the WinForms user control assembly. In this case, you must be sure to rebuild and update the user control whenever the licensed embedded controls are updated.

I downloaded a new build of a component that I have purchased, and now I'm getting the splash screen when I build my projects.

Make sure that the serial number is still valid. If you licensed the component over a year ago, your subscription may have expired. In this case, you have two options:

Option 1 – Renew your subscription to get a new serial number.

If you choose this option, you will receive a new serial number that you can use to license the new components (from the installation utility or directly from the **About Box**).

The new subscription will entitle you to a full year of upgrades and to download the latest maintenance builds directly from http://prerelease.componentone.com/.

Option 2 – Continue to use the components you have.

Subscriptions expire, products do not. You can continue to use the components you received or downloaded while your subscription was valid.

Technical Support

ComponentOne offers various support options. For a complete list and a description of each, visit the ComponentOne Web site at http://www.componentone.com/Support.

Some methods for obtaining technical support include:

- **Online Support via HelpCentral**
  ComponentOne HelpCentral provides customers with a comprehensive set of technical resources in the form of FAQs, samples, Version Release History, Articles, searchable Knowledge Base, searchable Online Help, and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**
  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This e-mail will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.
- **Peer-to-Peer Product Forums and Newsgroups**
  ComponentOne peer-to-peer product forums and newsgroups are available to exchange information, tips, and techniques regarding ComponentOne products. ComponentOne sponsors these areas as a forum for users to share information. While ComponentOne does not provide direct support in the forums and newsgroups, we periodically monitor them to ensure accuracy of information and provide comments when appropriate. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.
- **Installation Issues**
  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.
- **Documentation**
  ComponentOne documentation is installed with each of our products and is also available online at HelpCentral. If you have suggestions on how we can improve our documentation, please e-mail the Documentation team. Please note that e-mail sent to the Documentation team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

Note: You must create a ComponentOne Account and register your product with a valid serial number to obtain support using some of the above methods.

Redistributable Files

ComponentOne PayPal eCommerce for ASP.NET is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.Web.C1PayPal.2.dll

Note: A distribution license must be obtained for each server CPU on which your Developed Web Server Application is installed. If your Developed Web Server Application is installed on a server for which a license has already been purchased, and only one CPU is operating the application, it is not necessary to purchase an additional license. However, additional licenses must be purchased for each additional CPU on the network that is operating the Developed Web Server Application.

Site licenses are available for groups of multiple developers. Please contact Sales@ComponentOne.com for details.

About This Documentation

Acknowledgements

Microsoft, Visual Studio, Visual Basic, Internet Explorer, Windows, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation; PayPal is a registered trademark of PayPal Inc. in the United States and/or other countries; Firefox is a registered trademark of the Mozilla Foundation. Safari is a trademark of Apple Inc., registered in the U.S. and other countries.

ComponentOne

If you have any suggestions or ideas for new features or controls, please call us or write:

Corporate Headquarters

ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA
412.681.4343
412.681.4384 (Fax)

http://www.componentone.com/

ComponentOne Doc-To-Help

This documentation was produced using ComponentOne Doc-To-Help® Enterprise.

Namespaces

Namespaces organize the objects defined in an assembly. Assemblies can contain multiple namespaces, which can in turn contain other namespaces. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries.

The general namespace for ComponentOne Web products is **C1.Web**. The namespace for the **ComponentOne PayPal eCommerce for ASP.NET** components is **C1.Web.C1PayPal**. The following code fragment shows how to declare a **C1PurchaseItem** component using the fully qualified name for this class:

- Visual Basic
```
Dim C1PurchaseItem1 As C1.Web.C1PayPal.C1PurchaseItem
```
- C#
```
C1.Web.C1PayPal.C1PurchaseItem C1PurchaseItem1;
```

Namespaces address a problem sometimes known as *namespace pollution*, in which the developer of a class library is hampered by the use of similar names in another library. These conflicts with existing components are sometimes called name *collisions*.

For example, if you create a new class named **C1PurchaseItem**, you can use it inside your project without qualification. However, the **C1.Web.C1PayPal.dll** assembly also implements a class called **C1PurchaseItem**. So, if you want to use the **C1PurchaseItem** class in the same project, you must use a fully qualified reference to make the reference unique. If the reference is not unique, Visual Studio .NET produces an error stating that the name is ambiguous. The following code snippet demonstrates how to declare these objects:

- Visual Basic
```
'  Define a new C1PurchaseItem object.
Dim MyItem as C1PurchaseItem'  Define a new C1PurchaseItem.
Dim NewItem as C1.Web.C1PayPal.C1PurchaseItem
```

- C#

```
//  Define a new C1PurchaseItem object.
MyItem as C1PurchaseItem;
//  Define a new C1PayPal.C1PurchaseItem object.
NewItem as C1.Web.C1PayPal.C1PurchaseItem;
```

Fully qualified names are object references that are prefixed with the name of the namespace where the object is defined. You can use objects defined in other projects if you create a reference to the class (by choosing **Add Reference** from the **Project** menu) and then use the fully qualified name for the object in your code.

Fully qualified names prevent naming conflicts because the compiler can always determine which object is being used. However, the names themselves can get long and cumbersome. To get around this, you can use the **Imports** statement (**using** in C#) to define an alias – an abbreviated name you can use in place of a fully qualified name. For example, the following code snippet creates aliases for two fully qualified names, and uses these aliases to define two objects:

- Visual Basic

```
Imports C1PurchaseItem = C1.Web.C1PayPal.C1PurchaseItem
Imports MyItem = MyProject.Objects.C1PurchaseItem
Dim c1 As C1PurchaseItem
Dim c2 As MyItem
```

- C#

```
using C1PurchaseItem = C1.Web.C1PayPal.C1PurchaseItem;
using MyItem = MyProject.Objects.C1PurchaseItem;
 c1 C1PurchaseItem;
 c2 MyItem;
```

If you use the **Imports** statement without an alias, you can use all the names in that namespace without qualification provided they are unique to the project

Creating an ASP.NET Project

When creating ASP.NET projects, Visual Studio gives you the option of creating a Web site project or a Web application project; the latter is similar to creating a Web project in Visual Studio 2003. The Web application project option was provided to help developers converting Web projects from Visual Studio 2003 to Visual Studio 2005.

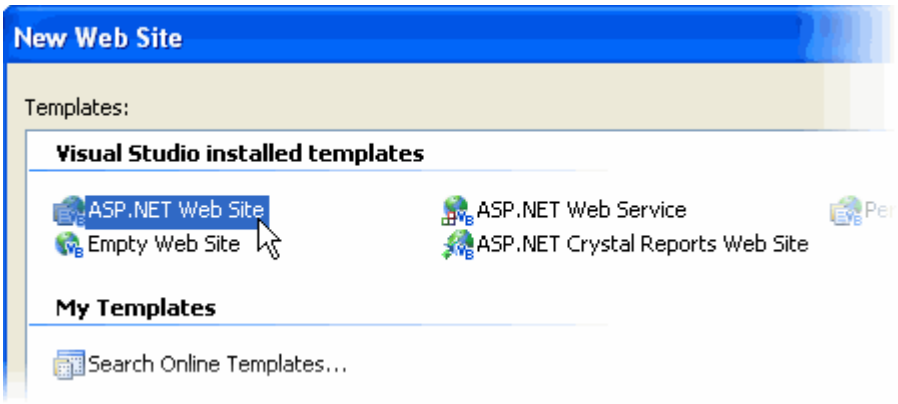
Creating a Web application project requires installation of a Visual Studio 2005 update and add-in, which can be found at http://msdn.microsoft.com/. See Microsoft's Web site for more detailed information and comparisons on Web site and Web application projects.

The steps for creating both types of projects have been provided for your convenience in the Creating a Web Site Project and Creating a Web Application Project topics.

Creating a Web Site Project

To create a Web site project, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio, select **New Web Site**. The **New Web Site** dialog box opens.
2. Select **ASP.NET Web Site** from the list of Templates.



3. Enter a URL for your application in the **Location** field and click **OK**.

> **Note:** The Web server must have IIS version 5 or later and the .NET Framework installed on it. If you have IIS on your computer, you can specify http://localhost for the server.

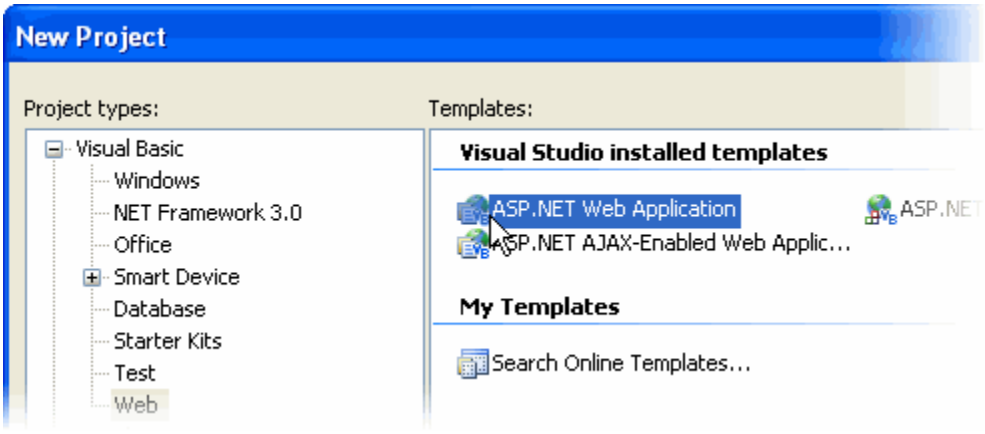
A new Web Forms project is created at the root of the Web server you specified. In addition, a new Web Forms page called WebForm1.aspx is displayed in the Web Forms Designer in Design view.

4. Double-click the **PayPal eCommerce for ASP.NET** component in the Toolbox to add it to WebForm1.aspx. For information on adding a component to the Toolbox, see Adding the PayPal eCommerce for ASP.NET Components to a Project.

Creating a Web Application Project

To create a new ASP.NET Web application project, complete the following steps.

1. From the **File** menu in Microsoft Visual Studio, select **New Project**. The **New Project** dialog box opens.
2. Under **Project Types**, choose either **Visual Basic Projects** or **Visual C# Projects**. Note that one of these options may be located under **Other Languages**.
3. Select **ASP.NET Web Application** from the list of **Templates** in the right pane.



4. Enter a URL for your application in the **Location** field and click **OK**.

Note: The Web server must have IIS version 5 or later and the .NET Framework installed on it. If you have IIS on your computer, you can specify http://localhost for the server.

5. A new Web Forms project is created at the root of the Web server you specified. In addition, a new Web Forms page called Default.aspx is displayed in the Web Forms Designer in **Design** view.
6. Double-click the **PayPal eCommerce for ASP.NET** components in the Toolbox to add them to WebForm1.aspx. For information on adding a component to the Toolbox, see Adding the PayPal eCommerce for ASP.NET Components to a Project.

Adding the PayPal eCommerce for ASP.NET Components to a Project

When you install ComponentOne Studio for ASP.NET 2.0, the **Create a ComponentOne Visual Studio 2008\2005 Toolbox Tab** checkbox is checked, by default, in the installation wizard. When you open Visual Studio, you will notice a **ComponentOne Studio for ASP.NET 2.0** tab containing the ComponentOne controls has automatically been added to the Toolbox.

If you decide to uncheck the **Create a ComponentOne Visual Studio 2008\2005 Toolbox Tab** checkbox during installation, you can manually add ComponentOne controls to the Toolbox at a later time.

ComponentOne PayPal eCommerce for ASP.NET provides the following controls:

- C1AddToCart
- C1CartCheckout
- C1PurchaseItem

To use **PayPal eCommerce for ASP.NET**, add these controls to the form or add a reference to the C1.Web.C1PayPal.2 assembly in your project.

Manually Adding PayPal eCommerce for ASP.NET to the Toolbox

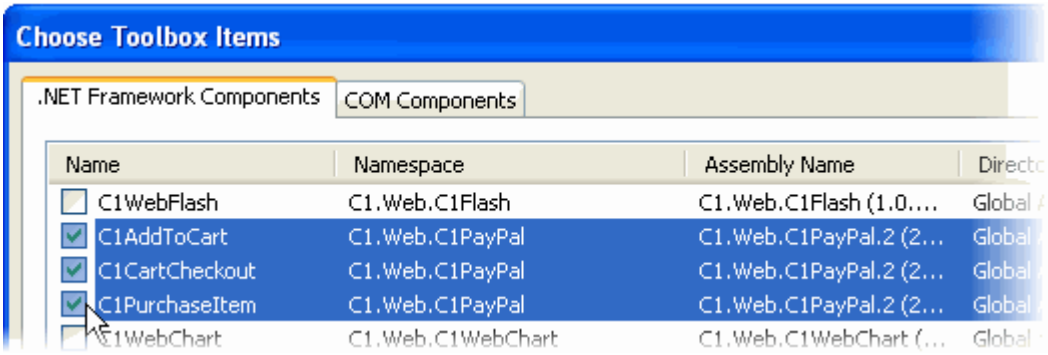
When you install **ComponentOne PayPal eCommerce for ASP.NET**, the following **PayPal eCommerce for ASP.NET** components will appear in the Visual Studio Toolbox customization dialog box:

- C1AddToCart
- C1CartCheckout
- C1PurchaseItem

To manually add the **PayPal eCommerce for ASP.NET** controls to the Visual Studio Toolbox:

1. Open the Visual Studio IDE (Microsoft Development Environment). Make sure the Toolbox is visible (select **Toolbox** in the **View** menu, if necessary) and right-click the Toolbox to open the context menu.
2. To make the **PayPal eCommerce for ASP.NET** components appear on their own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1PayPal**, for example.
3. Right-click the tab where the components are to appear, and select **Choose Items** from the context menu.

   The **Choose Toolbox Items** dialog box opens.
4. In the **Choose Toolbox Items** dialog box, go to the **.NET Framework Components** tab. Sort the list by Namespace (click the Namespace column header) and check the check box for the component belonging to namespace **C1.Web.C1PayPal**. Note that there may be more than one component for each namespace.



Adding PayPal eCommerce for ASP.NET to the Form

To add **PayPal eCommerce for ASP.NET** to a form:

1. Add the **PayPal eCommerce for ASP.NET** controls to the Visual Studio Toolbox.
2. Double-click the control or drag it onto your form.

Adding a Reference to the Assembly

To add a reference to the **PayPal eCommerce for ASP.NET** assembly:

1. Select the **Add Reference** option from the **Website** menu of your Web Site project or from the **Project** menu of your Web Application project.
2. Select the **ComponentOne PayPal eCommerce for ASP.NET** assembly from the list on the **.NET** tab or browse to find the C1.Web.C1PayPal.2.dll file and click **OK**.
3. Double-click the form caption area to open the code window. At the top of the file, add the following **Imports** statements (**using** in C#):

```
Imports C1.Web.C1PayPal
```

> **Note:** This makes the objects defined in the **PayPal eCommerce for ASP.NET** assembly visible to the project. See Namespaces for more information.

Migrating a PayPal eCommerce for ASP.NET Project to Visual Studio 2005

To migrate a project using ComponentOne components to Visual Studio 2005, there are two main steps that must be performed. First, you must convert your project to Visual Studio 2005, which includes removing any references to a previous assembly and adding a reference to the new assembly. Second, you must update the project References to use the new **PayPal eCommerce for ASP.NET** .dll.

To convert the project:

1. Open Visual Studio 2005 and select **File | Open | Project/Solution**.
2. Locate the **.sln** file for the project that you wish to convert to Visual Studio 2005. Select it and click **Open**. The **Visual Studio Conversion Wizard** appears.
3. Click **Next**.
4. Select **Yes, create a backup before converting** to create a backup of your current project and click **Next**.
5. Click **Finish** to convert your project to Visual Studio 2005. The **Conversion Complete** window appears.
6. Click **Show the conversion log** when the wizard is closed if you want to view the conversion log.

7. Click **Close**. The project opens. Now you must remove references to any of the previous ComponentOne .dlls and add references to the new ones.
8. Go to the Solution Explorer (**View | Solution Explorer**) and select the project node. Click the **Properties** button.
9. The **Properties Pages** dialog box appears. Select C1.Web.C1PayPal reference and click **Remove**.
10. Click **Add Reference**. Locate and select **C1.Web.C1PayPal.2.dll**. Click **OK** to add it to the project.
11. Click **OK** to close the **Property Pages** dialog box.

To update each Web form:

1. Open an .aspx file. Switch to Source view and locate the line at the top of the document that reads:

```
<%@ Register TagPrefix="cc1" Namespace="C1.Web.C1PayPal"
Assembly="C1.Web.C1PayPal" %>
```

2. Change it to read:

```
<%@ Register TagPrefix="cc1" Namespace="C1.Web.C1PayPal"
Assembly="C1.Web.C1PayPal.2" %>
```

3. Repeat for each .aspx file.
4. Right-click the main .aspx page and select **Set As Start Page**.
5. Click the **Start Debugging** button to compile and run the project.

The migration process is complete.

Key Features

By leveraging the power of ASP.NET, the **PayPal eCommerce for ASP.NET** controls make PayPal services (http://www.paypal.com/) easier to use and even more powerful. Setting up your Web store with **PayPal** has many advantages:

- Using PayPal is free. There are no fees to set up or use the service (PayPal charges per transaction, like credit card companies).
- Making a purchase is an easy, fast experience for your users. They can enter their information each time they make a purchase or they can use a current PayPal account or set up a new account to use.
- You can add your company logo to the PayPal payment pages, making your brand visible to your customers at all times.
- There's no need to learn CGI, and no need to worry about security; PayPal handles all the details for you. You can even use a free Web hosting company.
- Using the **PayPal eCommerce for ASP.NET** controls makes PayPal even easier and more powerful, because it leverages the power of ASP.NET. With **PayPal eCommerce for ASP.NET**, you can:
  - Set up real-time credit card processing with PayPal
  - Create payment and shopping cart buttons on your pages simply by dragging and dropping the **PayPal eCommerce for ASP.NET** controls on the page (you don't need to know HTML or CGI).
  - Use the **PayPal eCommerce for ASP.NET** wizards to set up each control quickly and easily – no code needed.
  - Use ADO.NET data binding to create your store based on information stored in a database, so the store is updated automatically for you.

- Controls create payment buttons for both single-item purchases (C1PurchaseItem) or multiple-item purchases (C1AddToCart, C1CartCheckOut).

- The **PayPal eCommerce for ASP.NET** controls are also free and do not require a license. ComponentOne only requires that you register at www.componentone.com. After registering, you can install and use the controls on as many machines as you want.

Getting Started with PayPal eCommerce for ASP.NET

The following topics describe the controls included in **ComponentOne PayPal eCommerce for ASP.NET** and how they work, and explain how to set up an account, add the **PayPal eCommerce for ASP.NET** controls to your form, change property settings, and use the **C1PayPal Wizard** to set up the controls.

PayPal eCommerce for ASP.NET Controls

The **ComponentOne PayPal eCommerce for ASP.NET** package contains three Web controls in a single DLL (C1.Web.C1PayPal.2.dll). The controls can be added to the Visual Studio Toolbox and dropped onto any ASP.NET Web Form, just like the standard HTML and Web controls that ship with Visual Studio. See Adding the PayPal eCommerce for ASP.NET Components to a Project for additional information. The controls are described in the following sections.

C1PurchaseItem Control 

The C1PurchaseItem control is an image button that contains information about your business and a specific item that you want to sell. When the button is clicked, the user is taken to a secure page where he can enter his information (name, address, credit card number, or simply his PayPal account) and confirm the purchase. After the purchase is confirmed, you receive a notification via e-mail and your PayPal account receives the credit for the purchase.

C1AddToCart Control 

The C1AddToCart control is also an image button that contains information about your business and a specific item that you want to sell. When the button is clicked, it shows a shopping cart table in a new window. The user can click other buttons to add items to the shopping cart, and he can use the shopping cart window to remove items from the cart, change quantities, or check out to finish the transaction. After the purchase is confirmed, you receive a notification via e-mail and your PayPal account receives the credit for the purchases.

C1CartCheckout Control 

The C1CartCheckout control is used with the C1AddToCart control. It also displays the shopping cart table in a new window, but does not add any items to the cart. The user can review the items in the cart, edit the table, and check out.

How PayPal eCommerce for ASP.NET Controls Work

Processing payments over the Web is not a trivial task. There are many issues to consider, including security, reliability, and availability. PayPal is one of the leading providers of e-commerce services. It is a public company, the services are available in many countries, and it offers great security and reliability. Plus, the service is free (PayPal charges a fee per transaction, like credit card companies).

ComponentOne PayPal eCommerce for ASP.NET controls work by trapping click events, building a custom URL string with information about your business and the item being purchased, and redirecting the browser to the PayPal site, where the transaction is processed.

This design shields you and your site from the complexities of designing and maintaining the e-commerce infrastructure, while at the same time giving you total control over the Web store itself.

The PayPal pages where your customers make the payments are not on your site, but they can be customized with your logo, so your brand is always visible and the user experience is smooth and comfortable.

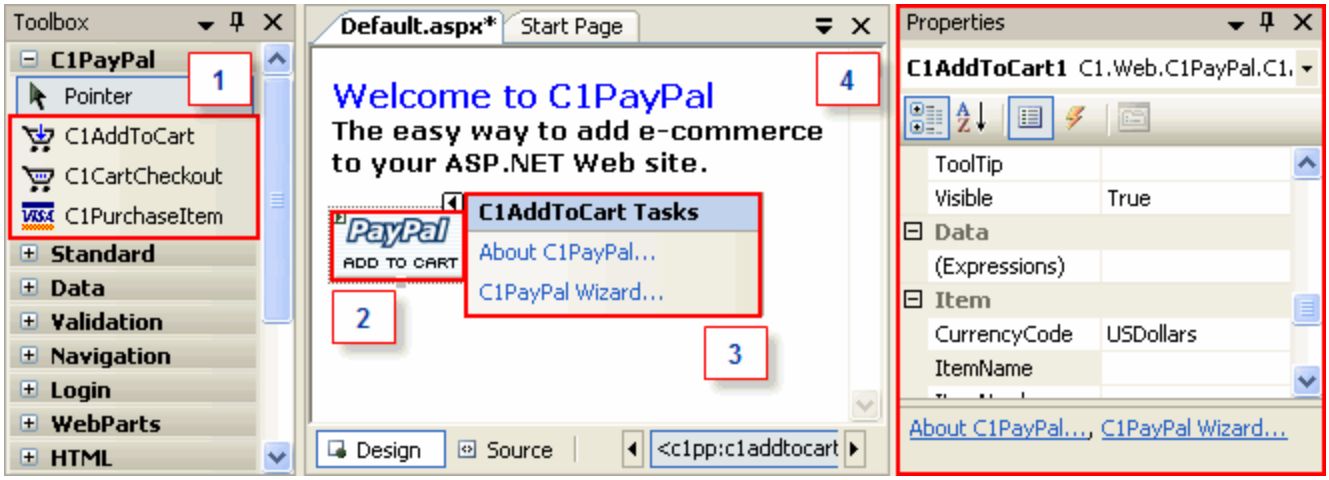
The PayPal Web site (www.paypal.com) provides a wealth of information on the services provided by PayPal. There is also a section devoted to developers, **PayPal Developer Central** (http://www.paypal.com/developer) that you should definitely visit. Because the **PayPal eCommerce for ASP.NET** controls use the standard PayPal protocols, all the information on the PayPal site is relevant to **PayPal eCommerce for ASP.NET** users.

Setting Up Your Web Form

To use the **ComponentOne PayPal eCommerce for ASP.NET** controls, you should have a PayPal business account. If you don't have one yet, you can set one up at the PayPal Web site (go to www.paypal.com and click the "Sign Up" link, it's free).

Next, you should add the **PayPal eCommerce for ASP.NET** controls to the Visual Studio Toolbox. For instructions, see Adding the PayPal eCommerce for ASP.NET Components to a Project.

This is what the **PayPal eCommerce for ASP.NET** controls look like in Visual Studio:



The following table describes what each numbered box represents in Visual Studio:

Box Number	Description
1	These icons are the PayPal eCommerce for ASP.NET controls in the Toolbox. You can drag them from here to the .aspx page on the right.
2	This is a C1AddToCart control on the page. This is the default image, obtained from the PayPal Web site. You can select other images available at the PayPal Web site or use your own custom images.
3	This is the C1AddToCart Tasks menu, which is available by clicking the smart tag (▶) above the control. The C1PayPal Wizard link invokes the C1PayPal Wizard. The wizard allows you to setup the PayPal eCommerce for ASP.NET controls in a step by step fashion, providing information about each property as you go. You can choose to use the Wizard or the property page. Both allow you to set the control properties.
4	The Properties window allows you to enter information about your business and the item being purchased. This information is submitted to PayPal so they can handle the transaction, credit your account, and send you notifications.

Note: To use the **PayPal eCommerce for ASP.NET** controls effectively, you should have an active Internet connection. The controls interact with PayPal even at design time to show predefined button images and help

content directly from the PayPal site (as shown on the above picture). If you don't have an active Internet connection, you can still use the controls and create projects and Web pages, but the controls will not display or work correctly until a connection becomes available.

Using the Properties Window

Once you have an account and the controls are installed in the toolbox, you can use them by dragging them onto your .aspx Web pages and filling out the following properties:

- **Business:** This is the e-mail address you used to register your PayPal account. The PayPal site will look up your information based on this address.
- **BusinessLogo:** This item is optional. If provided, it should be the URL of an image, 150 by 50 pixels, containing your business logo. The image is used to customize the PayPal payment and shopping cart pages. Ideally, the image should reside on a secure site (https://…), because the PayPal site is secure and the browser will display a warning if non-secure items are included.
- **ItemName, ItemNumber, ItemPrice:** These properties describe the item being purchased or added to the shopping cart. (These properties do not apply to the C1CartCheckout control, which shows the shopping cart but does not add any items to it).

These are the main properties that you need to set. There are many other optional properties that allow you to customize the transaction, specifying what information you want from customers, shipping and handling charges, and so on. These properties are described later, in the **Control Reference** section.

As with any control, you can set properties by entering their values in the Properties window or typing them programmatically. For example, you could store the information about the items in a database and build the controls dynamically. This approach is usually the best way to build real-world Web, because the information is stored in a single place; therefore, your Web store will always be up to date. See the Using Data-Bound PayPal eCommerce for ASP.NET sample that explains how to do this.

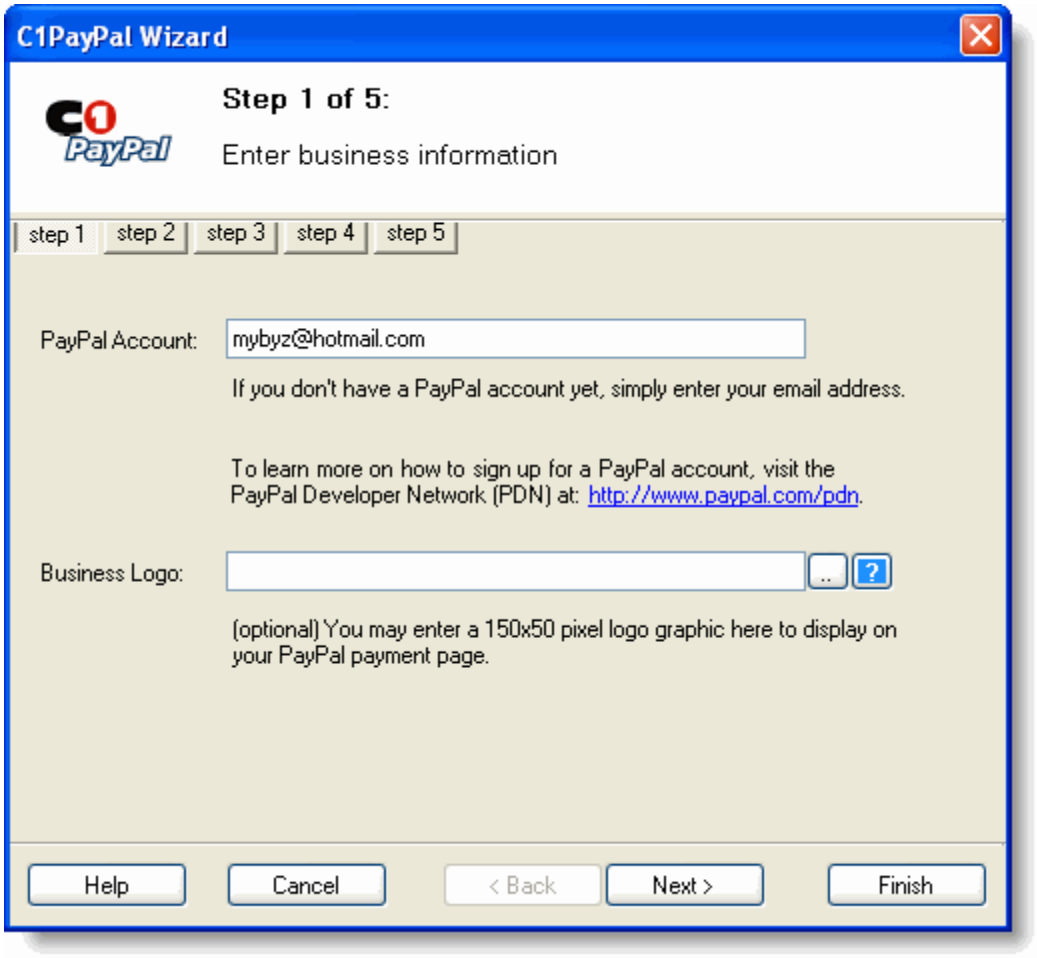
Using the C1PayPal Wizard

If you are going to set up the **ComponentOne PayPal eCommerce for ASP.NET** controls manually, you can use the built-in wizard instead of the Property pages. The wizard provides additional help on each property and groups properties based on their function, making the whole process much easier than using the Properties window directly. To invoke the wizard, click the **C1PayPal Wizard** link that appears at the bottom of the Properties window when a **PayPal eCommerce for ASP.NET** control is selected.

The wizard has up to five steps (for the C1PurchaseItem control, fewer steps for the other controls).

Entering Your Business Information

Use this page to enter your business information (Business and BusinessLogo properties):



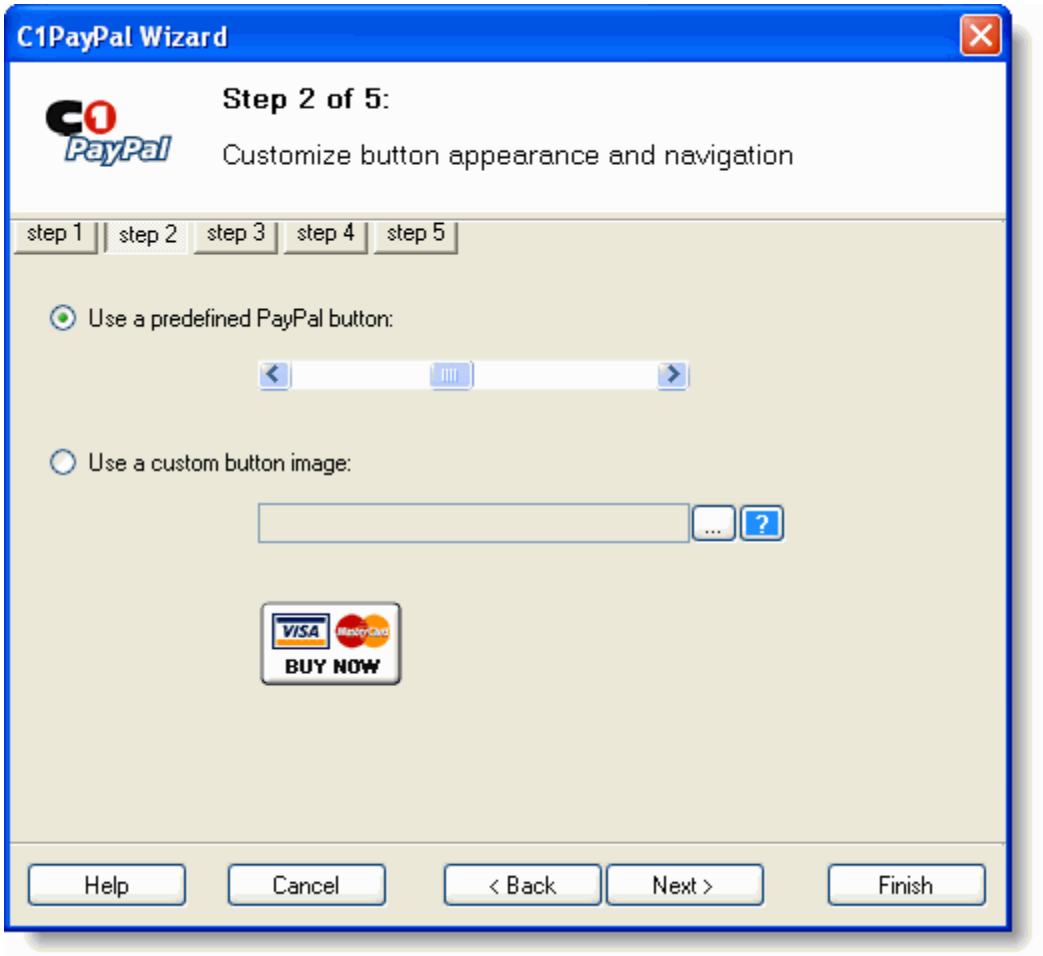
The **PayPal Account** entry corresponds to the Business property. This is the e-mail address you provided for notifications when you set up your PayPal account. This is how the PayPal site will identify your business when it sets up a transaction. It will allow PayPal to:

- Credit your account when someone purchases an item from you.
- Notify you of the transaction so you can send the items to the customer.

The **Business Logo** entry corresponds to the BusinessLogo property and is optional. It allows you to specify an image that will be displayed in the PayPal payment and shopping cart pages, branding your Web store. Ideally, the image should be 150 by 50 pixels and it should reside on a secure site (https://…), because the PayPal site is secure and the browser will display a warning if non-secure items are included.

Customizing Button Appearance and Navigation

Use this page to configure the appearance of the button on the page (ButtonType and ImageUrl properties):

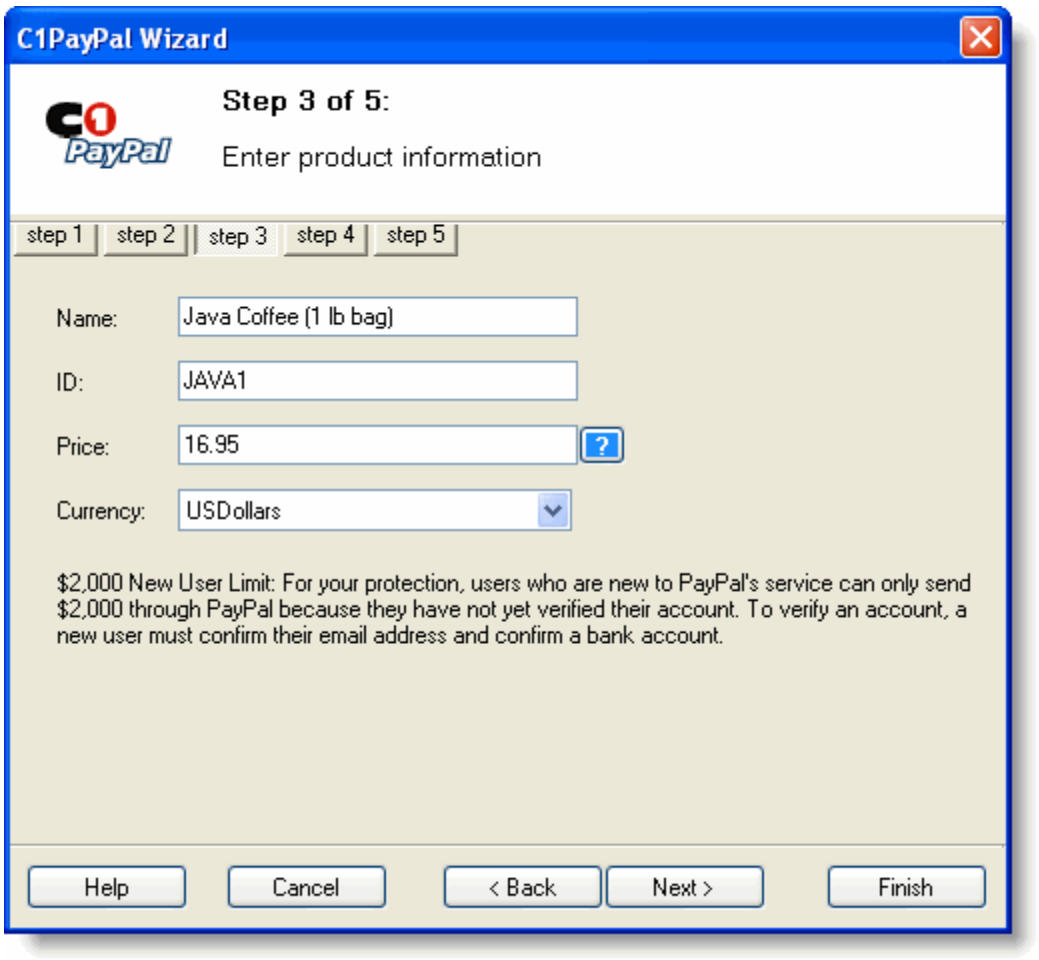


You can select one of the predefined images available at the PayPal site, or provide a custom image for the button.

Note that the button only displays an image. Usually, you will place other controls next to the button to show the item name, price, description, image, and so on. This can be done easily with other controls such as **Label**, **LinkButton**, and **Image** controls. If you have many items on your store, you will probably prefer to do this using code.

Entering the Product Information

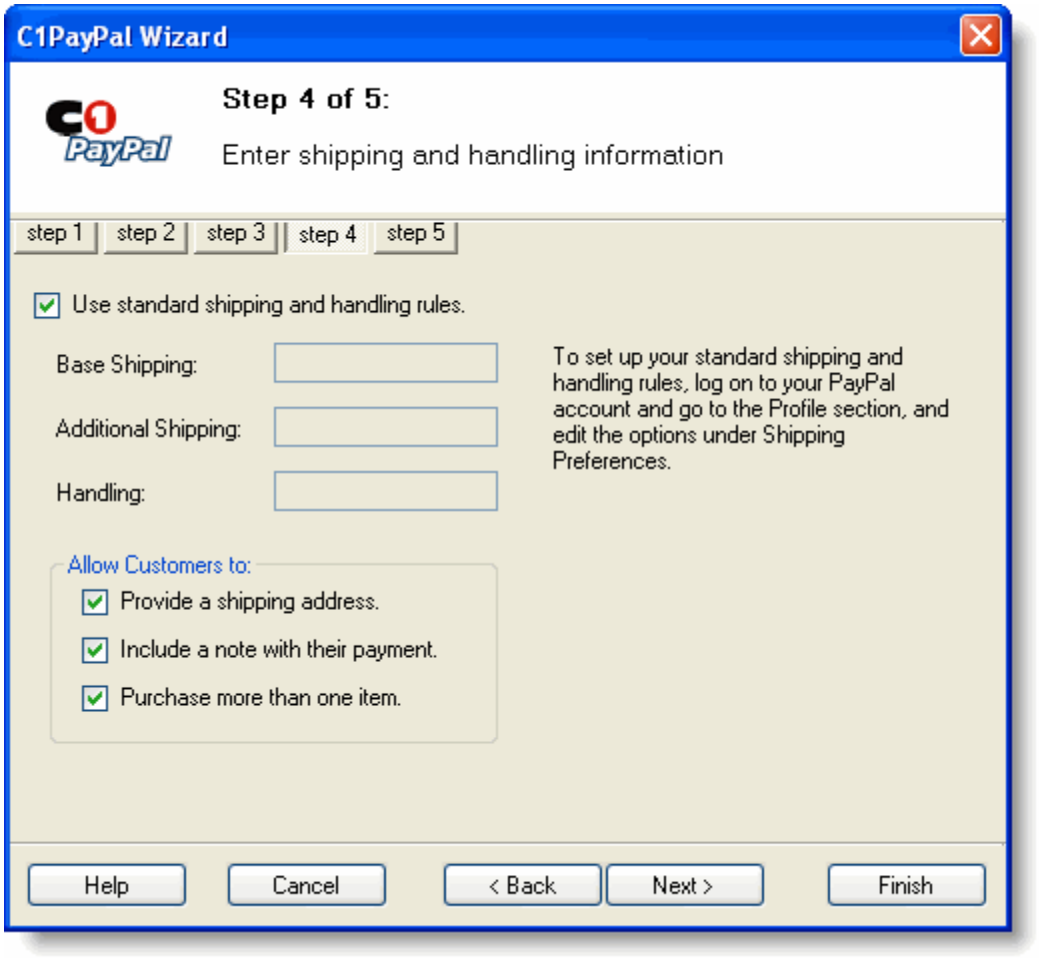
Use this page to enter information about the item you are selling (ItemName, ItemNumber, ItemPrice and CurrencyCode properties):



Note that PayPal has a $2,000 limit on transactions for new users. Once a customer has verified their account by confirming their e-mail address and bank account information, the limit is removed.

Entering Shipping and Handling Information

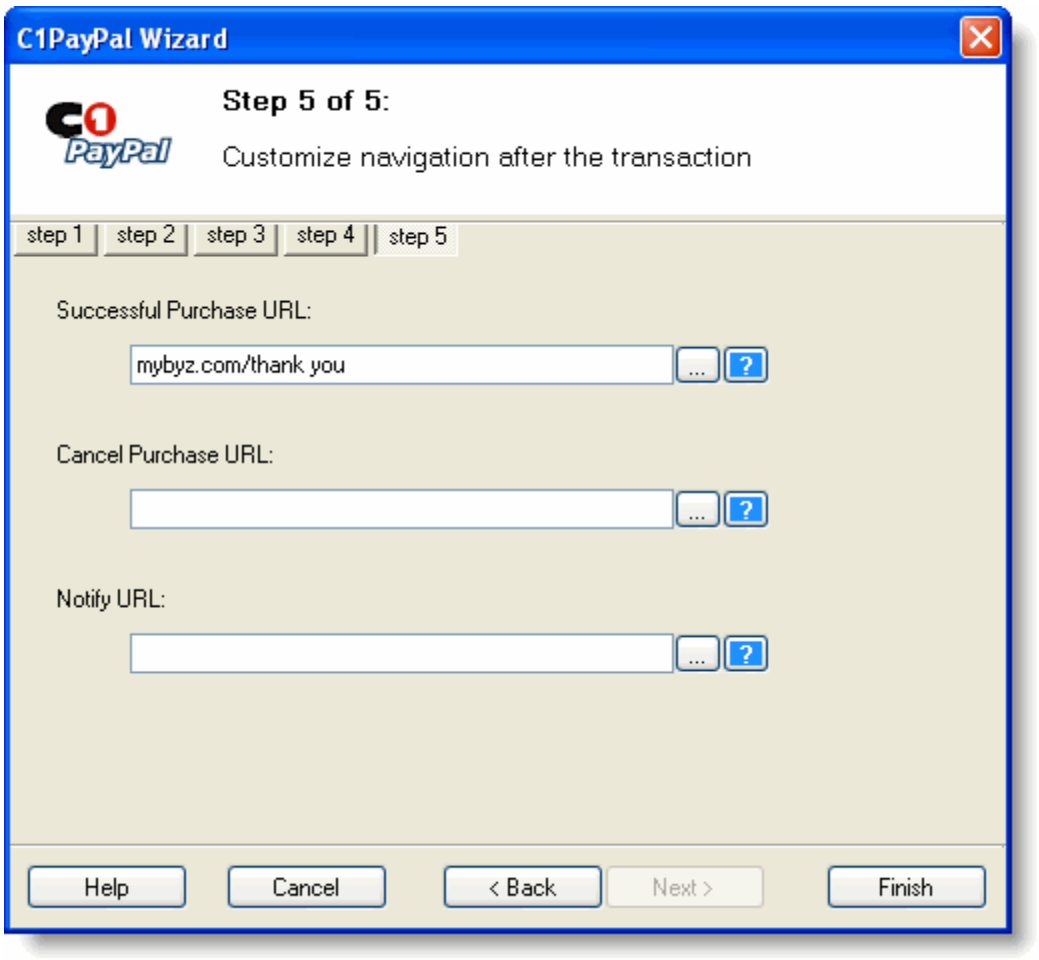
Use this page to specify shipping and handling information (Shipping, ShippingAdditional, Handling, PromptAddress, PromptNote, PromptQuantity properties):



By default, **ComponentOne PayPal eCommerce for ASP.NET** uses the shipping and handling rules you specified when you set up your PayPal account, and you should rarely have to override them. To set up or change the shipping and handling rules, go to your PayPal account, Profile section, and edit the Shipping Preferences.

Customizing Navigation After the Transaction

Use this last page to customize the navigation after each transaction (PageOK, PageCancel and NotifyURL properties):



By default, the user will return to your page after each transaction. You can change this behavior by specifying other pages that should be used after the user concludes a successful transaction or after a transaction is canceled.

For example, you could use this feature to direct users to a "thank you for your purchase" page that tells them when they can expect the product to arrive by mail. You can also append parameters to the url to help expedite or log the transaction.

PayPal eCommerce for ASP.NET Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other development tools included with the ComponentOne Studios.

Note: The ComponentOne Samples are also available at http://helpcentral.componentone.com/Samples.aspx.

C# Samples

Sample	Description
SouthWind	Use **ComponentOne PayPal eCommerce for ASP.NET** controls

	to build a Web store. This sample uses the C1AddToCart, C1PurchaseItem, and C1CartCheckout controls.

Building a Web Store

This section describes the following two applications that are based on the **ComponentOne PayPal eCommerce for ASP.NET** controls:

- The Unbound PayPal eCommerce for ASP.NET Controls sample uses the controls in unbound mode. All the information in each control is entered manually, using the wizard or the property pages. This eliminates any coding. Maintaining the pages will require you to go back to Visual Studio and edit the content of the controls.
- The Data-Bound PayPal eCommerce for ASP.NET Controls sample uses the controls in data-bound mode. All the information about the items is stored in an ADO.NET database, and the pages are built on-the-fly. This takes a little bit more planning. However, updating the store does not require heavy maintenance. Just change the database and the store will reflect the changes automatically. You can add, remove, or change the information for any items, and the next time someone visits your store they will see the new information.

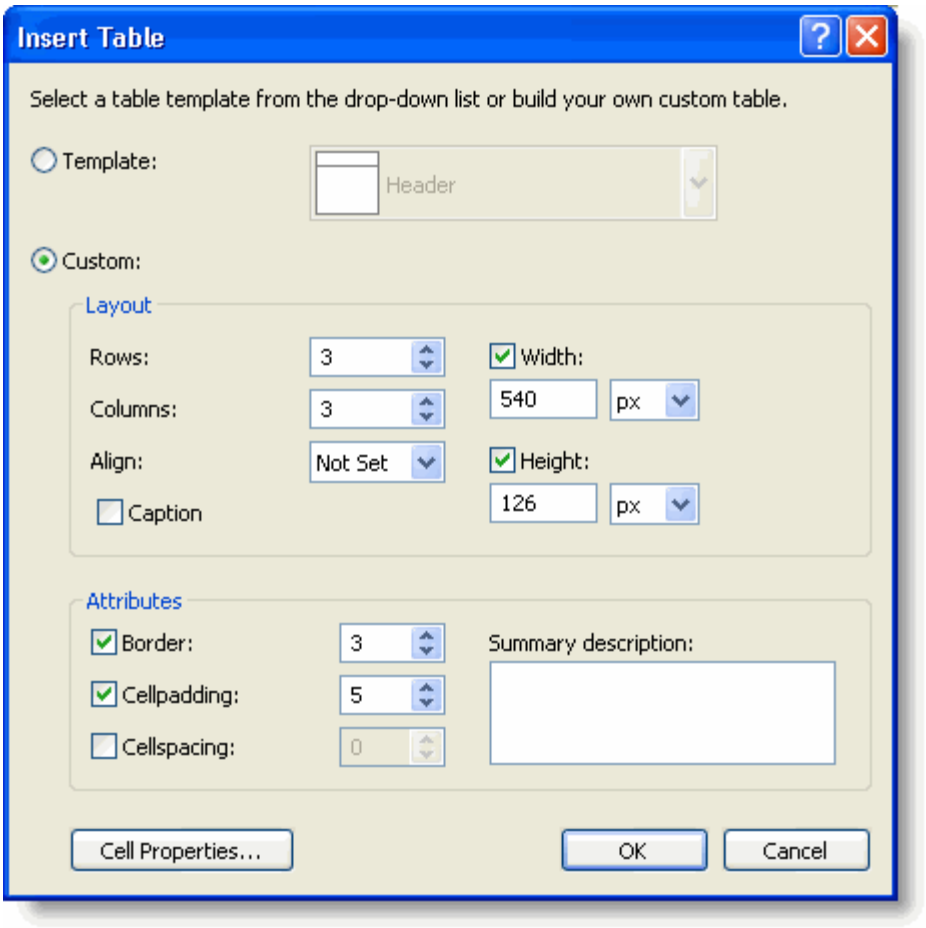
Using Unbound PayPal eCommerce for ASP.NET Controls

This sample consists of a simple Web store built to sell items from a fictitious company called SouthWind. The store consists of a table with three columns:

- The first column shows the item name and description.
- The second column shows the item price.
- The last column contains a **C1PurchaseItem** button that the user can click in order to purchase the item.

To build a Web store using unbound **PayPal eCommerce for ASP.NET** controls, complete the following steps:

1. To set up the table, select **Layout | Insert Table** from the Visual Studio menu. Here's the information the sample uses to create the table:



- To edit the color, select the **Cell Properties** button. The **Cell Properties** dialog box appears. Set the background color to **Lemon Chiffon** and select **OK**.
- From the table's Properties window, set the **BorderColor** property to **Gold**.

2. Now that the table is on the page, you need to type the names of the products into the first column and their prices into the second. You can format the strings and the table using the Visual Studio menus and toolbars. You can also edit the text, by clicking the **ellipsis** button located next to the **Style** property in the Properties pane. The **StyleBuilder** dialog box appears.
3. On the document, just above the table, type "C1PurchaseItem" as Header 2 style and "Click the buttons below to buy **Southwind** products:" as Normal style.
4. Next, drag a C1PurchaseItem control into the last cell of each row of the table. By default, the control will display a bitmap with credit card images. This is one of the predefined images available at the PayPal site. You can change the image by changing the ButtonType property to use one of the other predefined images. For this sample, set the ButtonType property to the **WinXP** button type. Do this for each control button.
5. Set each C1PurchaseItem.**BusinessLogo** property to **master.gif**, which is located in the C:\Program Files\ComponentOne Studio.NET 2.0\C1PayPal\Samples\CS\SouthWind directory.

> **Note:** The image must be added to your project folder in order for it to appear in the **SelectImage** dialog box.

Click on the **ellipsis** button to open the **Select Image** dialog box and select the **master.gif** image from the contents list.

6. Then, set each Business property to the e-mail address you used when you set up your PayPal account. For this sample we set the **Business** property to paypaltransactions@SouthWind.com.
7. Next, set the properties for each **C1PurchaseItem** button to the following:

C1PurchaseItem1

Property	Setting
ItemName	Open Water Scuba Diving Course
ItemNumber	C-BASIC
ItemPrice	123

C1PurchaseItem2

Property	Setting
ItemName	Advanced Scuba Diving Course
ItemNumber	C-ADVANCED
ItemPrice	213

C1PurchaseItem3

Property	Setting
ItemName	Dive Master Course
ItemNumber	C-MASTER
ItemPrice	632

Note: Make sure the ItemPrice property matches the value you entered in the second column of the table, or your customers will be surprised (they may be very happy or very mad at you).

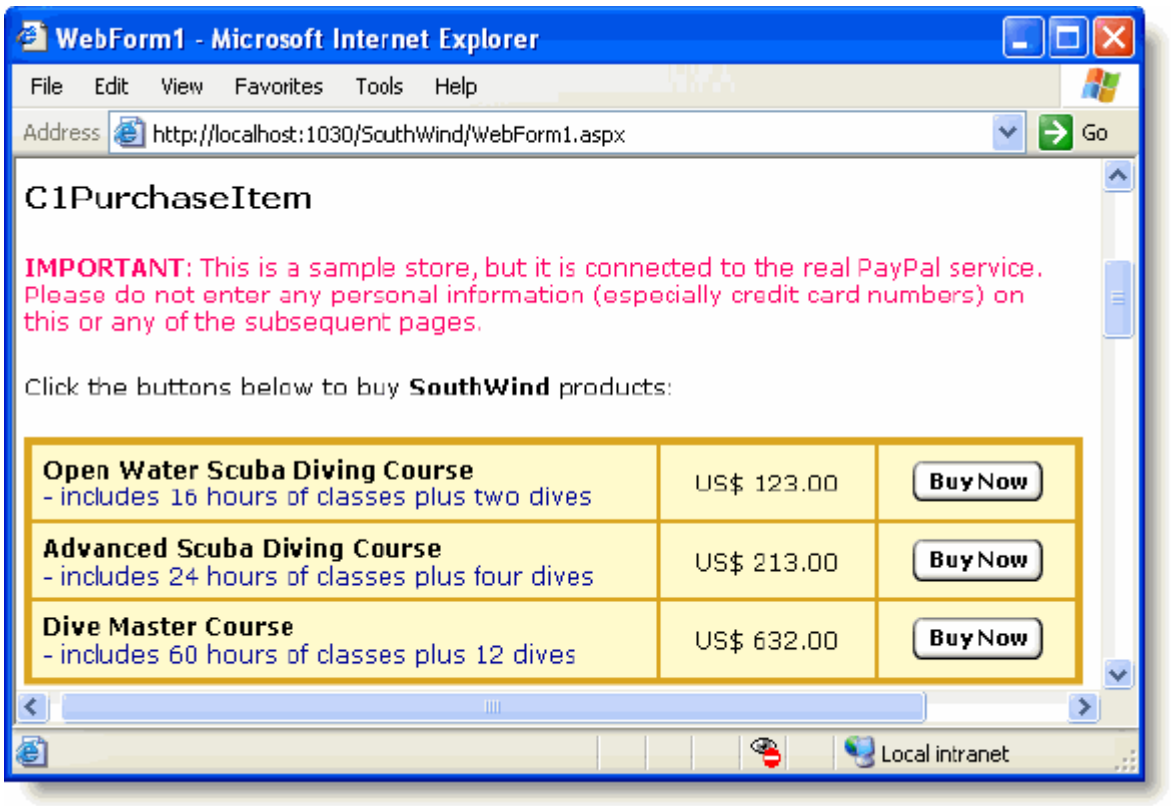
At this point, your page should look more or less like this:

If you switch to the HTML view, you'll see the code behind each C1PurchaseItem control. Here is the code behind the **C1PurchaseItem1** control:

```
<cc1:C1PurchaseItem id="C1PurchaseItem1" runat="server"
DESIGNTIMEDRAGDROP="67" ImageUrl="https://www.paypal.com/images/x-click-
but23.gif" Handling="-1" ShippingAdditional="-1" Shipping="-1"
BusinessLogo="http://localhost/PayPalDemos/SouthWind/master.gif"
ItemPrice="123" ItemNumber="C-BASIC" ItemName="Open Water Scuba Diving
Course" Business="paypaltransactions@SouthWind.com"
ButtonType="WinXP"></cc1:C1PurchaseItem>
```

This is really all there is to it. You can run the project now and you will see a fully functional Web store. If the Business property on your buttons is set to a valid PayPal account, you can actually click the buttons and buy the items.

This is what the application looks like when viewed in the Internet Explorer:



This approach is simple and works well. The problem is that some of the information appears in multiple places (control properties and text), and it might be difficult to maintain the page. For example, if the price of a product changes you need to update the table text and the ItemPrice property on the C1PurchaseItem control.

A better approach in most practical applications would be to get all the item information from a database, and build the page that way. This is described in the next sample.

Using Data-Bound PayPal eCommerce for ASP.NET Controls

This sample shows how you might build a more realistic Web store using data stored in a database to automatically set up the table and all the controls in it.

To the end user (customer), the store will look similar to the one created in the previous example. However, this time we will create the store using a **DataList** control. The **DataList** control is one of the server controls that ship with the .NET framework. It provides a flexible mechanism to display items stored in a database. In this case, each item is a product that you want to sell, and each will be represented by a table row with three columns as before.

Step 1 of 4: Creating and Formatting the DataList Control

In this step you'll create and then format the **DataList** control.

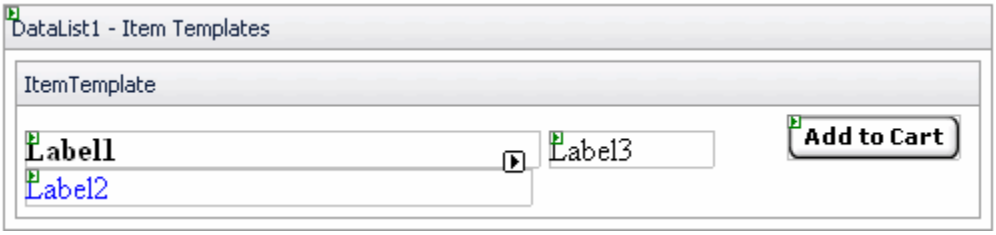
To create the DataList control, complete the following steps:

1. From the Toolbox, drag a **DataList** control to the page. The control displays a message saying that an **ItemTemplate** is required. The **DataList** control uses templates to define the appearance of the list header, footer, items, and item separators.
2. To create the **ItemTemplate,** click the smart tag (▶) above the **DataList** control and select **Edit Templates** from the **Data List Tasks** menu.

   Visual Studio will show a placeholder where you can drag controls that make up the template.

3. Click the **ItemTemplate** division and add three **Label** controls and one C1AddToCart control to the template. The labels will display the item name, description, and price.
4. Set the following properties to format the controls:
   - Set **Label1**'s **Font.Bold** property to **True**.
   - Set **Label2**'s **ForeColor** property to **Blue**.
   - Set **C1AddToCart1**'s ButtonType property to **WinXP**.
5. Resize and type spaces between the controls.

   The **ItemTemplate** should look similar to the following:



To format the DataList's appearance, complete the following steps:

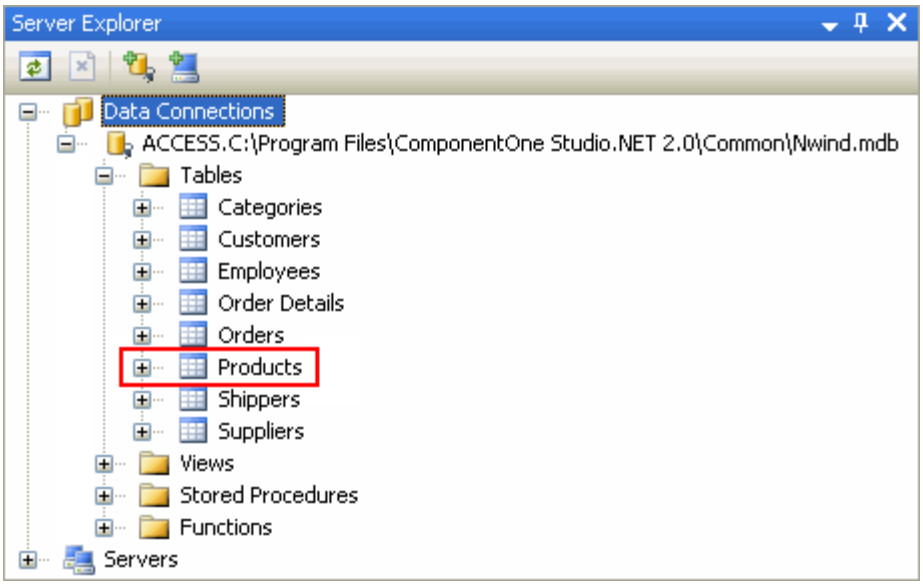
1. Right-click the **DataList** control and select the **End Template Editing** menu item.
2. In the **DataList Tasks** menu, select **Auto Format**. The **Auto Format** dialog box appears.
3. Choose the **Sand & Sky** scheme and click **OK**.
4. To add a header to the **DataList** control:
   - Click the smart tag (▶) above the **DataList** control and select **Edit Templates** from the **Data List Tasks** menu. Then select **Header Template** from the **Display** drop-down box.
   - Type "Welcome to the SouthWind Web Store" into the template.

Step 2 of 4: Adding a Data Source to the Form

If you already have the item information stored in a database, you can now bind it to the controls on the form. You can use the NorthWind database, located by default in the **C:\Program Files\ComponentOne Studio.NET 2.0\Common** directory, for the sample.

To add a data source to the page, complete the following steps:

1. Select **View | Server Explorer** from the Visual Studio menu.
2. In the Server Explorer, expand the tree to locate the data table you want to use, and drag the table to the form. (If you are using the NorthWind database, Nwind.mdb, drag the **Products** table to the form).



This will add **AccessDataSource1** to the form.

3. So that the **Products** table does not appear on the form, select **GridView1** and set its **Visible** property to **False** in the Properties window.
4. Now that the data source object is available, right-click the data list and select the **End Template Editing** menu item.

Step 3 of 4: Binding the Controls to a Data Source

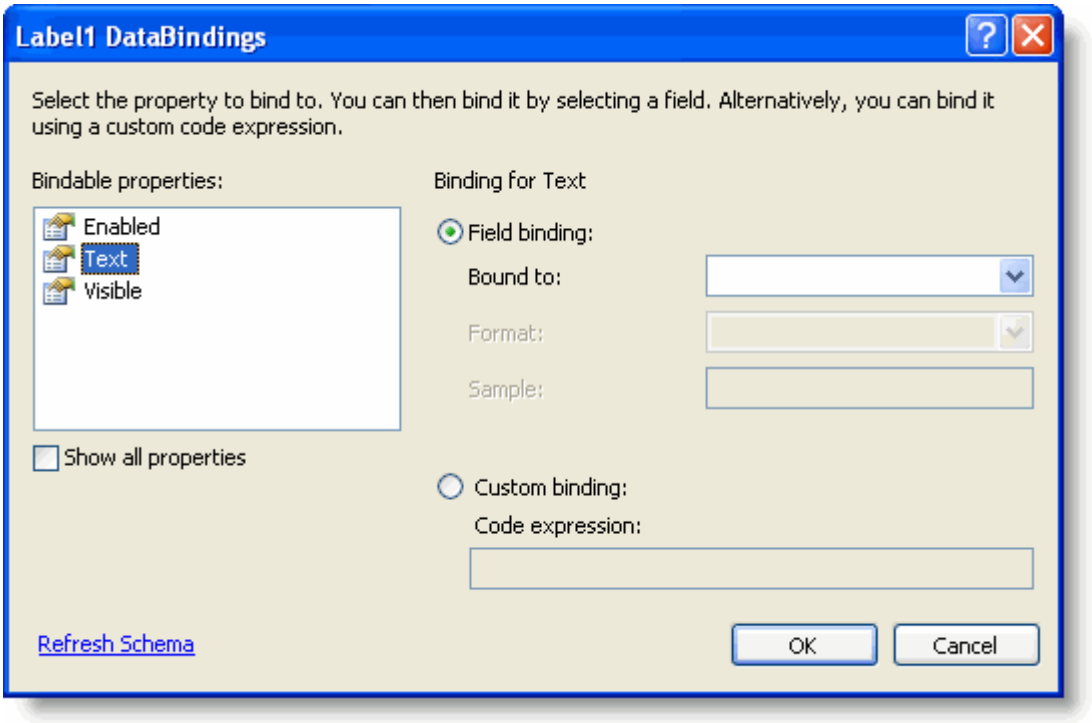
To bind the controls to a data source, complete the following steps:

1. Select the **DataList** control and use the Properties window to set the **DataSourceID** property to point to the **AccessDataSource1**. Click **No** to the dialog box that appears for resetting the template.

   This will cause the **DataList** control to bind to the table that contains the product information and to create one item on the page for each item in the database.

   Switch back to template editing mode. To bind the controls on the template to specific fields in the database click the smart tag (▶) above the **DataList** control and select **Edit Templates** from the **Data List Tasks** menu.
2. Then select the first **Label** control and click the smart tag (▶) above the Label control and select **Edit DataBindings**.

   The **DataBindings** dialog box appears, which allows you to bind one or more control properties to data fields.



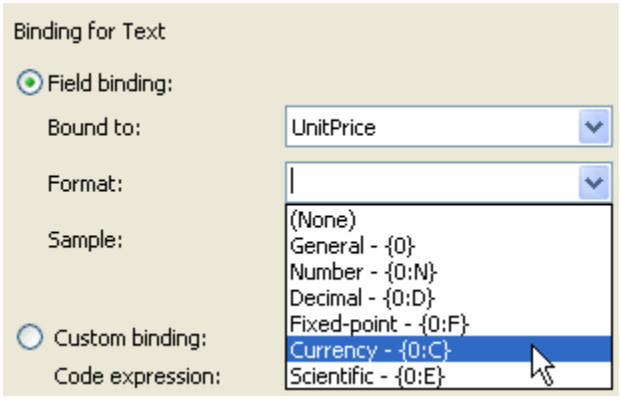
3. Bind the **Text** property on the **Label1** control to the **ProductName** field in the **Container** object, which in this case is the **DataList** control.

   Be careful not to bind the item directly to the database, or the list will always display the same item. The **Container** object takes care of enumerating all the records in the data source and making the current item available to the controls in the template.

4. Click **OK** to bind the label and close the **Label1 DataBindings** dialog box

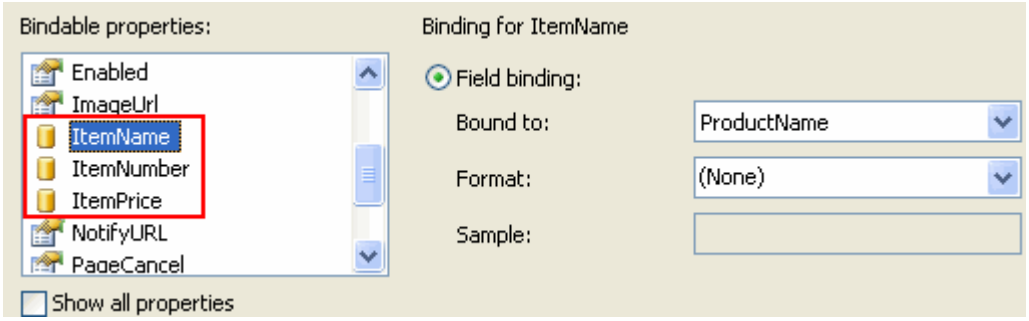
5. Repeat this procedure to bind the **Label2** control to the **QuantityPerUnit** field and the **Label3** control to the **UnitPrice** field (since there isn't a product description field handy in the NorthWind **Products** table).

   Note when binding the **Label3** control, you can set the format to **Currency** in the **DataBindings** dialog box:



Note: To add a custom business name, business logo or button type, select the C1AddToCart control and set the Business, BusinessLogo, and ButtonType properties in the Properties window. These will be the same for every item on the table.

6. Then click the smart tag (▶) above the **C1AddToCart** control, and select **Edit DataBindings** from the **C1AddToCart Tasks** menu. The **C1AddToCart1 DataBindings** dialog box appears.
7. This time, bind the **ItemName**, **ItemNumber,**and **ItemPrice** properties to the **ProductName**, **ProductID**, and **UnitPrice** database fields, respectively. This will set up the button correctly for each item in the data table.



Notice the icon that appears next to the property name for the data-bound properties. After you close the dialog box, the icon also appears next to the property name in the Properties window. This is a helpful feature while you are setting up the form.

8. Click **OK** to save your bindings and close the **C1AddToCart1 DataBindings** dialog box.

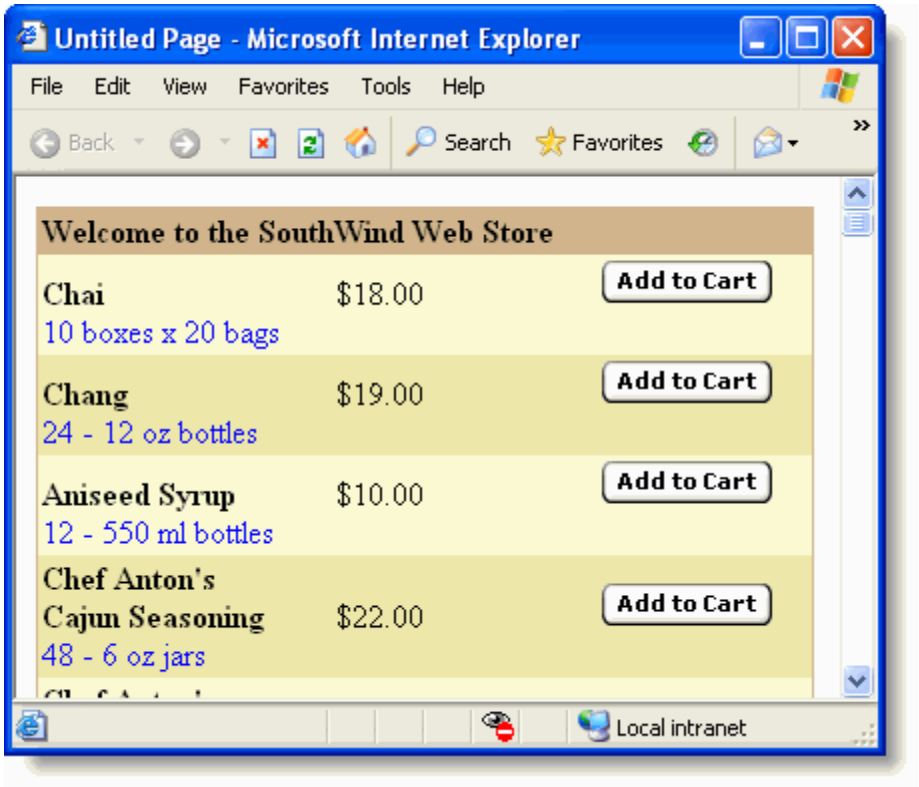
Step 4 of 4: Running the Application

The application is now ready. It took only a little bit of work to set up the data binding, and the new application has three important advantages:

- The ASP.NET code does not have to change if the product list changes. New items, prices, codes, and any other changes made to the database will always be reflected on this ASP.NET page.
- Information that appears in multiple places (the item price, for example) will always be in synch, and you don't run the risk of updating the item price displayed on the page and forgetting to update the **ItemPrice** property on the purchase button.
- If you want to change the appearance of the form, there is only one set of controls to edit (the ones in the template). For example, if you want to update the **BusinessLogo** property, there is only one **C1AddToCart** button to edit.

Run the application and observe:

When viewed in Internet Explorer the new application will look similar to the following:


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Welcome to the SouthWind Web Store
Chai
10 boxes x 20 bags
$18.00
Add to Cart
Chang
24 - 12 oz bottles
$19.00
Add to Cart
Aniseed Syrup
12 - 550 ml bottles
$10.00
Add to Cart
Chef Anton's
Cajun Seasoning
48 - 6 oz jars
$22.00
Add to Cart
Local intranet