ComponentOne

FormDecorator 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 FormDecorator for ASP.NET Overview

Integrate Cascading Style Sheets seamlessly into your page without adding any additional HTML using **ComponentOne FormDecorator for ASP.NET** (**C1FormDecorator**). C1FormDecorator enables styling for elements such as Textbox, Buttons, CheckBoxes, Labels, Scrollbars, RadioButtons, etc.

FormDecorator for ASP.NET is part of **ComponentOne Studio for ASP.NET**, the next breed of ASP.NET controls developed on a new client and server side framework. This new ASP.NET control suite fully exploits the AJAX framework to enable you to create highly interactive and sophisticated Web applications with Studio for ASP.NET.

What's New in ComponentOne FormDecorator for ASP.NET

This documentation was last revised on October 2, 2009. A new feature has been added to **ComponentOne FormDecorator for ASP.NET** in the 2009 v3 release:

New Features

C1FormDecorator now supports **Select** elements and **DropDownList** control. For more information on the decorated controls, see Decorated Controls.

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

Installing FormDecorator for ASP.NET

The following sections provide helpful information on installing ComponentOne Studio for ASP.NET:

FormDecorator 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.
C1.Web.UI	Contains files (at least a readme.txt) related to the C1WebUI product.
H2Help	Contains online documentation for the Studio for ASP.NET controls.
C1WebUi\VisualStyles	Contains all external file themes.

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.
C1WebUi	Contains a Samples folder for the Visual Studio sample project and a readme.txt file.

System Requirements

System requirements for **ComponentOne Studio for ASP.NET** components include the following:

Operating Systems:	Windows 7
	Windows® 2000
	Windows Server® 2003
	Windows Server 2008
	Windows XP SP2
	Windows Vista™
Web Server:	Microsoft Internet Information Services (IIS) 5.0 or later
Environments:	.NET Framework 2.0 or later
	Visual Studio 2005 or Visual Studio 2008
	Internet Explorer 6.0 or later
	Firefox® 2.0 or later
	Safari® 2.0 or later

Uninstalling Studio for ASP.NET

To uninstall **Studio for ASP.NET**:

1. Open the **Control Panel** and select the **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, the ComponentOne licensing model, and frequently asked licensing questions, 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.)

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 by Microsoft Visual Studio.

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 app 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 dialogs. 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.

Common Scenarios in Mobile Applications

The following topics describe some of the licensing scenarios you may encounter when working with Mobile components.

Updating or renewing a license

If you renew your subscription, the new license must be installed.

If the Mobile controls are licensed through a Studio subscription, then open the **About Box** of either an ASP.NET control or a .NET Windows forms control and update the license by clicking the **License** button and entering your serial number.

If the Mobile controls are licensed through a Studio for Mobile Devices subscription, then open the **About Box** of the 1.x version of a Mobile control or run the setup again to enter your serial number. Presently, the 2.x versions of the Mobile controls do not have the option to license or register from the **About Box**; therefore, it is necessary to license through another component or the setup.

Licensing 2.x Mobile Controls through the Setup

To enter the serial number (license) through the Studio for Mobile Devices 2.0 setup, follow these steps:

1. Run the ComponentOne Studio for Mobile Devices 2.0 setup.
2. Follow the instructions in the setup, and enter the serial number when prompted.

There are cases where the setup may not prompt you. If you have a valid license installed already, or if you are installing a version of the controls that has already been installed (Maintenance Mode), you will not be prompted to enter your serial number. If you need to enter your serial number for an install, and the install is running in Maintenance Mode, you will need to uninstall first. Once you uninstall, run the install again.

If you are still not prompted for a serial number by the time you get to the install screen, you must have a valid license in the registry on your machine. If you still need to install a new serial number, remove the old license and then run the setup again. There is a utility available that will remove the old license for you, which can be found below.

License Remover

The license remover will search for the following licenses on your system: Studio Enterprise, Studio for Mobile Devices, and individual Studio Mobile product licenses. Any licenses that are found will be populated into a list so that you can select the ones that you would like to remove. To remove a Studio for Mobile Devices license with the license remover, follow these steps:

1. Unzip and run the C1LicBomb.exe.
2. You will see an app with a list of the installed licenses on your machine. If you see both Studio Enterprise and Studio for Mobile devices listed, select both for removal, otherwise select the one that you see. If the new serial number that you wish to enter for Studio for Mobile Devices is not a Studio Enterprise serial

number, you will need to enter your Studio Enterprise serial number again, but this can be done through any of the About Boxes for C1 controls with the exception of the Mobile Studio About Boxes.

3. Select any other licenses you wish to remove, then click the **Remove Selected Licenses** button.
4. You will be asked if you are sure that you want to delete each key; click **OK** when you get the dialog box.
5. There will also be a dialog box to let you know that a particular key was removed successfully; click **OK** for that one also.
6. Now you can close the program and run your Studio for Mobile Devices setup again to enter your new serial number.

Follow through with the setup, enter your serial number when prompted, and ComponentOne Studio for Mobile Devices will be licensed on your machine.

Updating a project after renewing a license

Once you have installed a new license, each project must be updated in order to use the new control; rebuilding the control is not sufficient. It is necessary for the control to regenerate the license embedded in its **SupportInfo** property. To do this, it is necessary to force Visual Studio to update the control properties stored in the form. The easiest way to do this is to simply modify a property. The simplest choice is to toggle a Boolean property such as the **Visible** property, and then toggle it back to its original value. This results in no changes or side effects in the control configuration, but it forces the IDE to update **SupportInfo** and embed the new run-time license.

Instantiating a Mobile control at run time

The Mobile controls behave the same way as other ComponentOne controls when they are created at run time and not placed on the form at design time. Because the IDE does not have an opportunity to obtain a run-time license on its own in this case, it is necessary to force the IDE to include a run-time license. To accomplish this, include at least one instance of the control on a form in the assembly that is instantiated BEFORE the dynamically created instance is created. Once a control of the same type is licensed, all others on the form are accepted as licensed.

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 runtime 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 email 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 email 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 Studio 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.UI.2.dll
- C1.Web.UI.Controls.2.dll
- C1.Web.UI.3.dll
- C1.Web.UI.Controls.3.dll

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

About This Documentation

Acknowledgements

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.UI.**Controls. The following code fragment shows how to declare a **C1Menu** (which is one of the core **Studio for ASP.NET** classes) using the fully qualified name for this class:

- Visual Basic
```
Dim menu As C1.Web.UI.Controls.C1Menu
```
- C#
```
C1.Web.UI.Controls.C1Menu menu;
```

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*.

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 C1Menu = C1.Web.UI.Controls.C1Menu
Imports MyMenu = MyProject.Objects.C1Menu

Dim wm1 As C1Menu
Dim wm2 As MyMenu
```

- C#

```
using C1Menu = C1.Web.UI.Controls.C1Menu;
using MyMenu= MyProject.Objects.C1Menu;

C1Menu wm1;
MyMenu wm2;
```

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 AJAX-Enabled ASP.NET Project

ComponentOne FormDecorator for ASP.NET requires you to create an ASP.NET AJAX-Enabled project so that Microsoft ASP.NET AJAX Extensions and a **ScriptManager** control are included in your project before the C1FormDecorator control is placed on the page. This allows you to take advantage of ASP.NET AJAX and certain features such as partial-page rendering and client-script functionality of the Microsoft AJAX Library.

When creating AJAX-Enabled ASP.NET projects, Visual Studios 2008 and 2005 both give you the option of creating a Web site project or a Web application project. MSDN provides detailed information on why you would choose one option over the other.

If you are using Visual Studio 2008 with .NET Framework 2.0 or .NET Framework 3.0 or if you are using Visual Studio 2005, you must install the ASP.NET AJAX Extensions 1.0, which can be found at http://ajax.asp.net/. Additionally for Visual Studio 2005 users, 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/; however, if you have Visual Studio 2005 SP1, Web application project support is included and a separate download is not required.

If you are using Visual Studio 2008 and .NET Framework 3.5, you can easily create an AJAX-enabled ASP.NET project without installing separate add-ins because the framework has a built-in AJAX library and controls.

The following table summarizes the installations needed:

Visual Studio Version	Additional Installation Requirements
Visual Studio 2008, .NET Framework 3.5	None
Visual Studio 2008 and .NET Framework 2.0 or 3.0 Visual Studio 2005 Service Pack 1	ASP.NET AJAX Extensions 1.0 http://www.asp.net/ajax/downloads/archive/
Visual Studio 2005	ASP.NET AJAX Extensions 1.0 Visual Studio update and add-in (2 installs for Web application project support)

The following topics explain how to create both types of projects in Visual Studio 2008 and 2005.

- Creating an AJAX-Enabled Web Site Project in Visual Studio 2008

To create a Web site project in Visual Studio 2008, complete the following steps:

1. From the File menu, select **New** | Web Site. The New Web Site dialog box opens.
2. Select .NET Framework 3.5 or the desired framework in the upper right corner. Note that if you choose .NET Framework 2.0 or 3.0, you must install the extensions first.
3. In the list of templates, select **AJAX 1.0-Enabled ASP.NET 2.0 Web Site**.
4. Click **Browse** to specify a location and then click **OK**.

> **Note:** The Web server must have IIS version 6 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 AJAX-Enabled Web Site is created at the root of the Web server you specified. In addition, a new Web Forms page called Default.aspx is displayed and a **ScriptManager** control is placed on the form. The **ScriptManger** is needed to enable certain features of ASP.NET AJAX such as partial-page rendering, client-script functionality of the Microsoft AJAX Library, and Web-service calls.

- Creating an AJAX-Enabled Web Application Project in Visual Studio 2008

To create a new Web application project in Visual Studio 2008, complete the following steps.

1. From the **File** menu, select **New** | Project. The New Project dialog box opens.
2. Select .NET Framework 3.5 or the desired framework in the upper right corner. Note that if you choose .NET Framework 2.0 or 3.0, you must install the extensions first.
3. Under **Project Types**, choose either **Visual Basic** or **Visual C#** and then select **Web**. Note that one of these options may be located under **Other Languages**.
4. Select **AJAX 1.0-Enabled ASP.NET 2.0 Web Application** from the list of **Templates** in the right pane.
5. Enter a URL for your application in the **Location** field and click **OK**.

> **Note:** The Web server must have IIS version 6 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 Default.aspx is displayed and a **ScriptManager** control is placed on the form. The **ScriptManger** is needed to enable certain features of ASP.NET AJAX such as partial-page rendering, client-script functionality of the Microsoft AJAX Library, and Web-service calls.

- Creating an AJAX-Enabled Web Site Project in Visual Studio 2005

To create a Web site project in Visual Studio 2005, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio .NET, select **New Web Site**. The **New Web Site** dialog box opens.
2. Select **ASP.NET AJAX-Enabled Web Site** from the list of Templates.
3. Enter a URL for your site in the **Location** field and click **OK**.

> **Note:** The Web server must have IIS version 6 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 Default.aspx is displayed and a **ScriptManager** control is placed on the form.

The **ScriptManger** is needed to enable certain features of ASP.NET AJAX such as partial-page rendering, client-script functionality of the Microsoft AJAX Library, and Web-service calls.

- Creating an AJAX-Enabled Web Application Project in Visual Studio 2005

  To create a new Web application project in Visual Studio 2005, complete the following steps.

  1. From the **File** menu in Microsoft Visual Studio 2005, 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 AJAX-Enabled 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 6 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 Default.aspx is displayed and a **ScriptManager** control is placed on the form. The **ScriptManger** is needed to enable certain features of ASP.NET AJAX such as partial-page rendering, client-script functionality of the Microsoft AJAX Library, and Web-service calls.

Adding the C1FormDecorator Component to a Project

When you install ComponentOne Studio for ASP.NET, the **Create a ComponentOne Visual Studio Toolbox Tab** check box is checked, by default, in the installation wizard. When you open Visual Studio 2005, you will notice a ComponentOne Studio for ASP.NET Projects tab containing the ComponentOne controls that have automatically been added to the Toolbox.

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

To use **C1FormDecorator**, add the C1FormDecorator control to the form or add a reference to the C1.Web.UI.Controls.C1FormDecorator assembly in your project.

Manually Adding the Studio for ASP.NET controls to the Toolbox

When you install **ComponentOne Studio for ASP.NET**, the following C1FormDecorator component will appear in the Visual Studio Toolbox customization dialog box:

- C1FormDecorator

To manually add the FormDecorator for ASP.NET control 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 it to open the context menu.
2. To make the **C1FormDecorator** component appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1FormDecorator**, for example.
3. Right-click the tab where the component is to appear and select **Choose Items** from the context menu. The **Choose Toolbox Items** dialog box opens.
4. In the dialog box, select the **.NET Framework Components** tab. Sort the list by Namespace (click the **Namespace** column header) and check the check boxes for the C1Menu component belonging to namespace C1.Web.UI.Controls.C1FormDecorator.
5. Click **OK** to close the dialog box.

   The C1FormDecorator control is added Visual Studio Toolbox.

Adding C1FormDecorator to the Form

To add C1FormDecorator to a form:

1. Add it to the Visual Studio toolbox.
2. Double-click each control or drag it onto your form.

Adding a Reference to the Assembly

To add a reference to the C1.Web.UI.Controls.C1FormDecorator 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 most recent version of the **ComponentOne Studio for ASP.NET** assembly from the list on the **NET** tab or browse to find the C1.Web.UI.Controls.2.dll file and click **OK**.
3. Select the **Form1.vb** tab or go to **View|Code** to open the Code Editor. At the top of the file, add the following **Imports** directive (**using** in C#):

```
Imports C1.Web.UI.Controls.C1FormDecorator
```

> **Note:** This makes the objects defined in the **C1.Web.UI.Controls.2** assembly visible to the project. See Namespaces for more information.

Key Features

The **C1FormDecorator** control includes several unique features, including the following:

- **5 Visual Styles**

  Supports 5 kinds of embedded visual styles which can be used to decorate standard controls such as CheckBox, RadioButton, Button, TextBox, TextArea, FieldSet, etc.

- **Designate an Area to Decorate**

  Enables users to specify which part of the form should be decorated by setting the DecorationZoneID property. See the Decorated Zones topic for more information.

- **Designate Controls to Decorate**

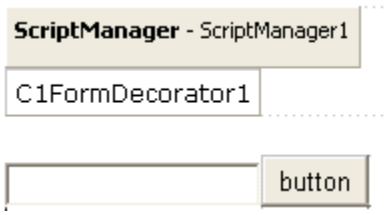
  Allows users to specify which controls should be decorated by setting the DecoratedControls property.

FormDecorator for ASP.NET Quick Start

The **C1FormDecorator** Quick Start describes how to get started with the ASP.NET control, **C1FormDecorator**. In the quick start you'll create an AJAX Enabled ASP.NET Web Site, add a **C1FormDecorator** control to the page, add some HTML controls, and set the C1FormDecorator's VisualStyle to change the style for all of the HMTL controls placed on the page.

1. In Visual Studio 2008, begin by creating a new AJAX Enabled ASP.NET Web Site.
2. In Design view, navigate to the Visual Studio Toolbox and double-click the **C1FormDecorator** icon to add the **C1FormDecorator** control to your page.
3. Add a HTML textbox control to the page.
4. Add a HTML button control to the page.

   Your page should appear like the following:



5. Click on the C1FormDecorator's smart tag to open its Tasks menu and select Vista from its VisualStyle dropdown box.
6. Run the program and observe the following:

   The Vista style is applied to both the HTML TextBox and Button controls:

Design-Time Support

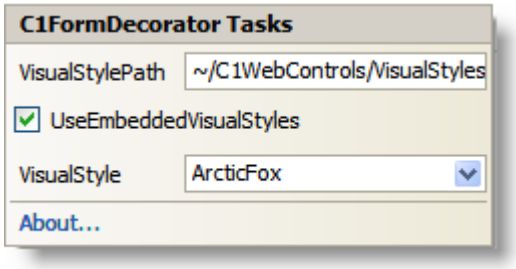
C1FormDecorator provides smart tags, designer, and a bindings collection editor that offers rich design-time support and simplifies working with the object model.

The following topics describe how to use C1Menu's design-time environment to configure the **C1FormDecorator** control.

C1FormDecorator Smart Tag

The **C1FormDecorator** control includes a smart tag in Visual Studio. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in **C1FormDecorator**.

To access the **C1FormDecorator Tasks** menu, click on the smart tag in the upper-right corner of the **C1FormDecorator** control. This will open the **C1FormDecorator Tasks** menu.



The **C1FormDecorator Tasks** menu operates as follows:

- **VisualStylePath**

  Displays the default path for the visual style.

- **UseEmbeddedVisualStyles**

  When the UseEmbeddedVisualStyles is selected it enables you to use the embedded visual styles.

- **VisualStyle**

  Clicking the **VisualStyle** drop-down arrow enables you to select from different built-in visual styles. See Visual Styles for more information.

- **About**

  Clicking on the **About** item displays the **About** dialog box, which is helpful in finding the version number of **FormDecorator for ASP.NET** and online resources.

Decorated Controls

C1FormDecorater can style most ASP.NET Web Server controls, ASP.NET HTML Server controls, and HTML tags such as FieldSet and <h3> through <h5> heading tags. The controls and elements that C1FormDecorator can style are known as decorated controls. By using C1FormDecorator you can apply its built-in visual styles such as the "Arctic Fox" visual style to the decorated controls and/or elements to create a more unified look.

C1FormDecorator has a property called DecoratedControls that allows you to select which controls you want C1FormDecorator to style. This property takes an enumeration of type FormDecoratorDecoratedControls.

To specify the decorated controls select one of the values in the FormDecoratorDecoratedControls Enum. The following table lists and describes each of the values included in the FormDecoratorDecoratedControls Enum:

Value	Description
All	Decorates all of the supported decorated controls. This is the default value.
Button	Decorates the Web Server and the HTML Server Button controls such as the Button, Submit, and Reset buttons.
CheckBox	Decorates the Web Server CheckBox and CheckBoxList controls and/or the HTML Server Input (Checkbox) control.
FieldSet	Decorates the FieldSet tag to that is used to logically group together elements in a form.

H3H4H5	Decorates the heading 3, heading 4, and heading 5 styles assigned in the HTML tags.
Label	Decorates the Web Server and/or the HTML Server Label controls.
None	Decorates none of the supported decorated controls.
RadioButton	Decorates the Web Server RadioButton and RadioButtonList controls.
ScrollBar	Decorates any scrollbars included in the supported decorated controls as well as the scrollbars inside a Panel control.
TextArea	Decorates the HTML Server Textarea control.
TextBox	Decorates the Web Server TextBox and the HTML Server Input (Text) controls.
DropDownList	Decorates the HTML Server Select and the Web Server DropDownList controls.

Decorated Zones

You can control which form elements get decorated by using the **DecorationZoneID** property. To designate a decorated area simply set the HTML element's ID to a value such as ZoneID and then assign the ZoneID to the **DecorationZoneID** property.

Note that you can define one zone. If you would like to use different visual styles for the decorated zones you must place a **C1FormDecorator** control for every zone you wish to decorate. For example if you wanted to have three decorated zones with three different visual styles then you would have a C1FormDecorator control for each zone.

The following sample code uses one **C1FormDecorator** control to define one decorated zone on the page. The left fieldset element's ID is set to ZoneID so the **DecorationZoneID** property can apply the visual style, Office2007Blue, to the controls inside the fieldset element.

Sample Code

```
<cc1:C1FormDecorator ID="C1FormDecorator1" runat="server"
        UseEmbeddedVisualStyles="True" VisualStyle="Office2007Blue"
DecorationZoneID="ZoneID" />

      <div >
                <div id="ZoneID" style="float: left; width: 289px; height:
360px;">
                    <fieldset>
                        <legend>Decorated</legend>
                        <div style="float: left; width: 250px; height:
79px;">
                            <asp:CheckBoxList ID="CheckBoxList1"
runat="server">
                                <asp:ListItem>CheckBox1</asp:ListItem>
                                <asp:ListItem>CheckBox2</asp:ListItem>

                            </asp:CheckBoxList>
                        </div>
                        <div style="float: left; width: 250px; height:
65px;">
                            <asp:RadioButtonList ID="RadioButtonList1"
runat="server">
```

```
                                <asp:ListItem>RadioButton1</asp:ListItem>
                                <asp:ListItem>RadioButton2</asp:ListItem>

                            </asp:RadioButtonList>
                        </div>
            <div style="float: left; width: 250px; height: 40px;">
<asp:DropDownList ID="DropDownList1" runat="server">
            <asp:ListItem>Item1</asp:ListItem>
            <asp:ListItem>Item2</asp:ListItem>
</asp:DropDownList>
              </div>
                         <div style="float: left; width: 250px; height:
120px; color: #000; font-size: 1.3em;">
                            <h3 style="color: #000; padding: 0; margin: 0;">
                                H3</h3>
                            <br />
                            <h4 style="color: #000; padding: 0; margin: 0;">
                                H4</h4>
                            <br />
                            <h5 style="color: #000; padding: 0; margin: 0;">
                                H5</h5>
                        </div>
                    </fieldset>
                </div>
                <div style="float: left; width: 264px; height: 360px;">
                    <fieldset >
                        <legend>Undecorated</legend>
                        <div style="float: left; width: 250px; height:
91px;">
                            <textarea cols="1" rows="3" style="width:
250px">TextArea</textarea>
                        </div>
                        <div style="float: left; width: 250px; height:
55px;">
                            <div style="width: 80px; float: left;">
                                <input type="button" value="button"
style="width: 70px;" />
                            </div>
                            <div style="width: 80px; float: left;">
                                <input type="reset" value="reset"
style="width: 70px;" />
                            </div>
                            <div style="width: 80px; float: left;">
                                <input type="submit" value="submit"
style="width: 70px;" />
                            </div>
                        </div>
                        <div style="float: left; width: 250px; height: 120px;
color: #000; font-size: 1.3em;">
                            <h3 style="color: #000; padding: 0; margin: 0;">
                                H3</h3>
                            <br />
                            <h4 style="color: #000; padding: 0; margin: 0;">
                                H4</h4>
                            <br />
                            <h5 style="color: #000; padding: 0; margin: 0;">
                                H5</h5>
```

```
                    </div>
                </fieldset>
            </div>
              </div>
            <div style="width: 660px; height: 60px; float: left;">
            </div>
```

C1FormDecorator Appearance

C1FormDecorator is designed to make customization easy for you. Without writing any code, you can apply any of the visual styles to the supported HTML controls. The following topics provide information on C1FormDecorator's appearance.

Visual Styles

C1FormDecorator is installed with five built-in visual styles for the control – **ArcticFox**, **Office2007Black**, **Office2007Blue, Office2007Silver**, and **Vista**. Each style can be easily applied to the control by setting the VisualStyle property. To see how to apply a style to any of C1FormDecorator's decorated controls see, FormDecorator for ASP.NET Quick Start.

Visual Style Settings

The following table illustrates each of the five built-in visual styles when applied to a Textbox and a Button control.

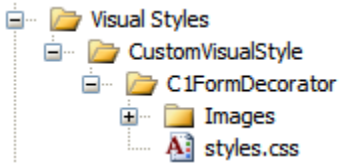
Visual Style	Appearance
ArcticFox	
Office2007Black	
Office2007Blue	
Office2007Silver	
Vista	

Custom Visual Styles

While **FormDecorator for ASP.NET** comes with five built-in styles, we recognize that there are instances where you will want to customize your C1FormDecorator control. To customize the C1FormDecorator control, you will create a custom CSS style sheet and add it to your project as a visual style. The custom CSS stylesheet must always be named "styles.css".

Tip: The easiest way to create a custom visual style is by modifying one of the control's pre-existing visual styles. You can find the .css sheets and images for **C1FormDecorator**'s visual styles within the installation directory at **C:\Program Files\ComponentOne\Studio for ASP.NET\C1WebUI\VisualStyles**.

Before you add your .css file and images, you will have to create a folder to contain them. The folder that will contain them will be buried within a hierarchy of folders. On the top-level of your project, create a folder named "VisualStyles". Underneath the VisualStyles folder, create a sub-folder bearing the theme name (such as "CustomVisualStyle"), and then, beneath that, create a sub-folder named "C1FormDecorator". The image folder and .css file should be placed underneath the **C1FormDecorator** folder. The folder hierarchy will resemble the following:



This structure of these folders is very important; **C1FormDecorator** will always look for the **~/VisualStyles/StyleName/C1FormDecorator/styles.css** path, as it is the default visual style path for the control.

Once the .css file and images are in place, set the **VisualStylePath** property to the path of the first folder (~/VisualStyles), set the **UseEmbeddedVisualStyles** property to **False,** and then set the VisualStyle property to the custom theme name.

FormDecorator 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 ComponentOne development tools included with ComponentOne Studio Enterprise.

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

C# Samples

The following pages within the **ControlExplorer** sample installed with **ComponentOne Studio for ASP.NET** detail the C1FormDecorator control's functionality:

Sample	Description
DecoratedControls	Shows the decorated controls that C1FormDecorator supports.
DecorationZone	Shows how to designate a specific area on your page that styles the decorated controls.
VisualStyles	Displays 6 FieldSet elements with decorated children controls. It Uses a dropdown listbox to show each of the five built-in visual styles for the C1FormDecorator control.

FormDecorator for ASP.NET Task-Based Help

The task-based help assumes that you are familiar with programming in ASP.NET and know how to use controls in general. Each topic provides a solution for specific tasks using the **C1FormDecorator** control. Each task-based help topic also assumes that you have created a new AJAX Enabled ASP.NET project.

Changing C1FormDecorator's Built-in Visual Style

This task illustrates how to change your visual style in Source view, Design view, and in code.

Changing the visual style using the designer:

To change the visual style for a Button control for example using the C1FormDecorator, follow these steps:

1. Add a Button control to your page.
2. Click C1FormDecorator's smart tag to open the **C1FormDecorator Tasks**.
3. Click the **VisualStyles** drop-down arrow and select a visual style from the list. For this example, choose **Office2007Blue**.

   The **Office2007Blue** visual style is applied to the C1FormDecorator.

Changing the visual style in source view:

To change the visual style of your HTML control using the C1FormDecorator in Source view, add `VisualStyle="Office2007Blue"` to the `<cc1:C1FormDecorator>` tag so that it resembles the following:

```
<cc1:C1FormDecorator ID="C1FormDecorator1" runat="server"
VisualStyle="Office2007Blue"
VisualStylePath="~/C1WebControls/VisualStyles">
```

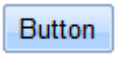
Changing the visual style programmatically:

To change the visual style, follow these steps:

1. Import the following namespace into your project:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1FormDecorator
     ```
   - C#
     ```
     using C1.Web.UI.Controls.C1FormDecorator;
     ```
2. Add the following code to the Page_Load event:
   - Visual Basic
     ```
     Me.C1FormDecorator1.VisualStyle = "Office2007Blue"
     ```
   - C#
     ```
     this.C1FormDecorator1.VisualStyle = "Office2007Blue";
     ```
3. Run the program.

This task illustrates the following:

The following image shows a decorated control, **Button**, with the **Office2007Blue** visual style:


Button