ComponentOne

Expander 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 Expander for ASP.NET Overview

ComponentOne Expander for ASP.NET lets you easily create expandable and collapsible information blocks that can include text, images, controls, and external Web sites. **ComponentOne Expander for ASP.NET** includes one control, **C1Expander**, which includes several built-in animation and styling effects to easily customize you application. With CSS styling and built-in Office 2007 and Vista themes, you can easily create sophisticated Web applications.

Getting Started

Get started with the following topics:

Key Features

Quick Start

Samples

What's New in Expander for ASP.NET

This documentation was last revised for 2009 v3 October 2, 2009. There were no new features added to **ComponentOne Expander 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 Expander for ASP.NET

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

Expander 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.
C1WebUi	Contains files (at least a readme.txt) related to the product.
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 samples and tutorials for **Expander for ASP.NET**.

System Requirements

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

Operating Systems:	Windows 2000
	Windows Server® 2003
	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

Note: Expander for ASP.NET requires Microsoft ASP.NET AJAX Extensions installed and a **ScriptManager** on the page before the C1Expander control is placed on the page. You must create an ASP.NET AJAX-Enabled Project so that the **ScriptManager** and Microsoft AJAX Extensions are included on the page. For more information, see Creating an AJAX-Enabled ASP.NET Project. For more information about Microsoft ASP.NET AJAX Extensions, see http://ajax.asp.net/. For information about the **ScriptManager**, see MSDN.

Installing Demonstration Versions

If you wish to try **Studio 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 Expander for ASP.NET

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

1. Open the **Control Panel** and select **Add or Remove Programs** or **Programs and Features** (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 ComponentOne 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 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

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 **C1TreeView** (which is one of the core **Studio for ASP.NET** classes) using the fully qualified name for this class:

- Visual Basic
```
Dim TreeView As C1.Web.UI.Controls.C1TreeView
```
- C#
```
C1.Web.UI.Controls.C1TreeView TreeView;
```

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 C1TreeView = C1.Web.UI.Controls.C1TreeView
```

```
Imports MyTreeView = MyProject.Objects.C1TreeView

Dim wm1 As C1TreeView
Dim wm2 As MyTreeViewMenu
```

- C#

```
using C1TreeView = C1.Web.UI.Controls.C1TreeView;
using MyTreeView = MyProject.Objects.C1TreeView;

 C1TreeView wm1;
 MyTreeView 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 Expander 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 C1Expander 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 Expander for ASP.NET Component to a Project

When you install **ComponentOne Studio for ASP.NET**, the **Create a ComponentOne Visual Studio 2005\2008 Toolbox Tab** check box is checked, by default, in the installation wizard. When you open Visual Studio, 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\2008 Toolbox Tab** check box during installation, you can manually add ComponentOne controls to the Toolbox at a later time.

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

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

- C1Expander

To manually add the Studio 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 it to open the context menu.
2. To make the Studio 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, Studio for ASP.NET, 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 all components belonging to namespace C1.Web.UI.Controls.2.dll. Note that there may be more than one component for each namespace.
5. Click **OK** to close the dialog box.

The controls are added to the Visual Studio Toolbox.

Adding Studio for ASP.NET Controls to the Form

To add Studio for ASP.NET controls to a form:

1. Add C1Expander to the Visual Studio toolbox.
2. Double-click the C1Expander control or drag it onto your form.

Note: Expander for ASP.NET requires Microsoft ASP.NET AJAX Extensions installed and a **ScriptManager** on the page before the C1Expander control is placed on the page. You must create an ASP.NET AJAX-Enabled Project so that the **ScriptManager** and Microsoft AJAX Extensions are included on the page. For more information, see Creating an AJAX-Enabled ASP.NET Project. For more information about Microsoft ASP.NET AJAX Extensions, see http://ajax.asp.net/. For information about the **ScriptManager**, see MSDN.

Adding a Reference to the Assembly

To add a reference to the C1.Web. UI.Conttrols.2 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
```

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

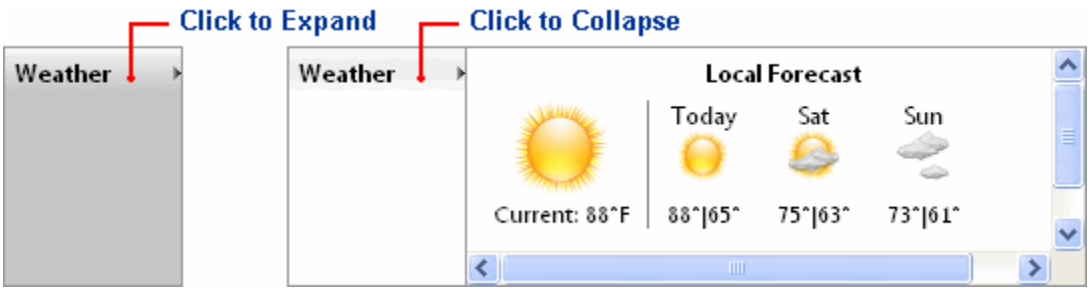
C1Expander includes several unique features, including the following:

- **Expanded on Page Load**

  You can choose whether or not the **C1Expander** control is initially expanded on page load by using the Expanded property. By default the Expanded property is set to **True** and the control is initially appears expanded.

- **Expand Direction**

  C1Expander has the ability to expand in several different directions. The ExpandDirection property indicated in which direction the control expands and can be set to **Top**, **Right**, **Bottom**, and **Left**.

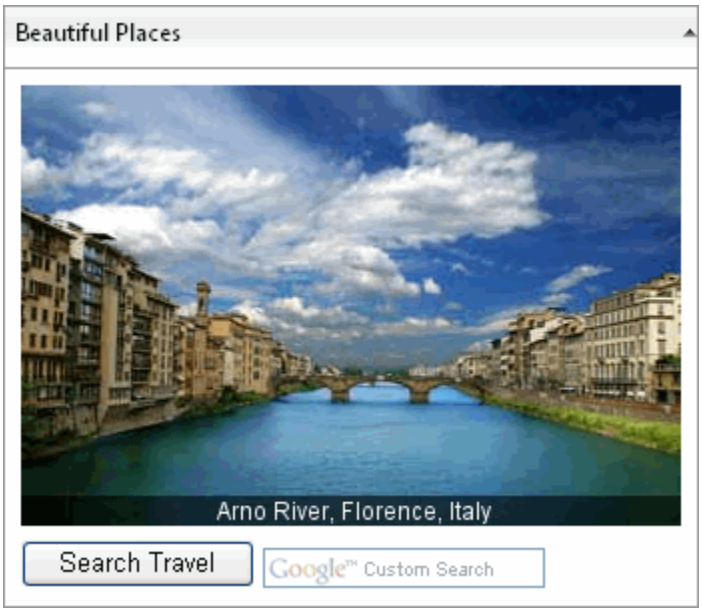


- **Built-In Animated Effects**

  C1Expander includes twenty-six different built-in animations that can be set when the control is expanded or collapsed using the ExpandAnimation and CollapseAnimation properties.

- **Customized Content**

  You can add images, text, and controls – standard and third-party controls – to the **C1Expander** control.



- **CSS Styling**

  C1Expander includes CSS supported styling so that you can use cascading style sheets to easily style the **C1Expander** control to match the design of your current Web site.

- **Client-Side Object Model**

  The **C1Expander** control's client-side object model is exposed so that you can easily customize the control with client-side script.

- **Template Support**

  You can add templates to the content area of the dialog window through the **Content** property. Dynamic Templates can be used in the content area of the dialog window for achieving rich presentation of the **C1Expander**.

- **External Content**

  You can show external content in the control using the **ContentUrl** property. This means that you can have the content of another Web page in your project or even the content of a Web site outside of your project appear in a **C1Expander**. For more information see Displaying External Content.

- **AJAX Support**

  Built-in AJAX support lets users interact with the **C1Expander** control without the control performing a postback operation back to the server.

- **Set Postbacks**

Use the **AutoPostBack** property to determine whether **C1Expander** should perform a postback to the server each time a user interacts with the control.

- **Browser Support**

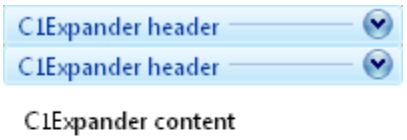
  C1Expander includes support for the Internet Explorer (6.0 or later), Firefox (2 or later), and Safari Web browsers.

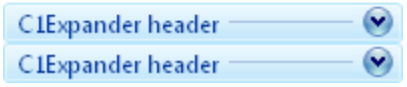
- **XHTML Compliant**

  C1Expander provides complete XHTML compliance. The output that is generated is fully XHTML 1.1 compliant.

- **Visual Styles**

  Use the built-in Visual Styles to quickly change the **C1Expander** control's appearance. Built-in styles include Office 2007 and Vista styles.





Expander for ASP.NET Quick Start

In this quick start you'll explore the functionality of the **C1Expander** control. **C1Expander** is similar to the **C1HeaderContentControl** control but provides the ability to expand and contract content with several built-in animation effects.

Step 1 of 5: Adding C1Expander to the Page

In this step you'll create and set up a Web site and add eight **C1Expander** controls.

Complete the following steps to add **C1Expander** controls to a Web site:

1. Create a new ASP.NET Ajax-Enabled Web site project. See Creating an AJAX-Enabled ASP.NET Project for details.

   Note that as you've created an AJAX-Enabled Web site, a **ScriptManager** control initially appears on the page.

2. In Source view add the following DIV tags to the body of your page, between the page's initial `<div></div>` tags:

```
<div id="row1" style="width: 480px; float:none; clear:both;">
    <div id="box1" style="float:left;">
           
    </div>
    <div id="box2" style="float:left;">
           
    </div>
    <div id="box3" style="float:left;">
           
    </div>
    <div id="box4" style="float:left;">
           
    </div>
</div>
<div id="row2" style="width: 480px; float:none; clear:both;">
    <div id="box5" style="float:left;">
           
    </div>
    <div id="box6" style="float:left;">
           
    </div>
    <div id="box7" style="float:left;">
           
    </div>
    <div id="box8" style="float:left;">
           
    </div>
</div>
```

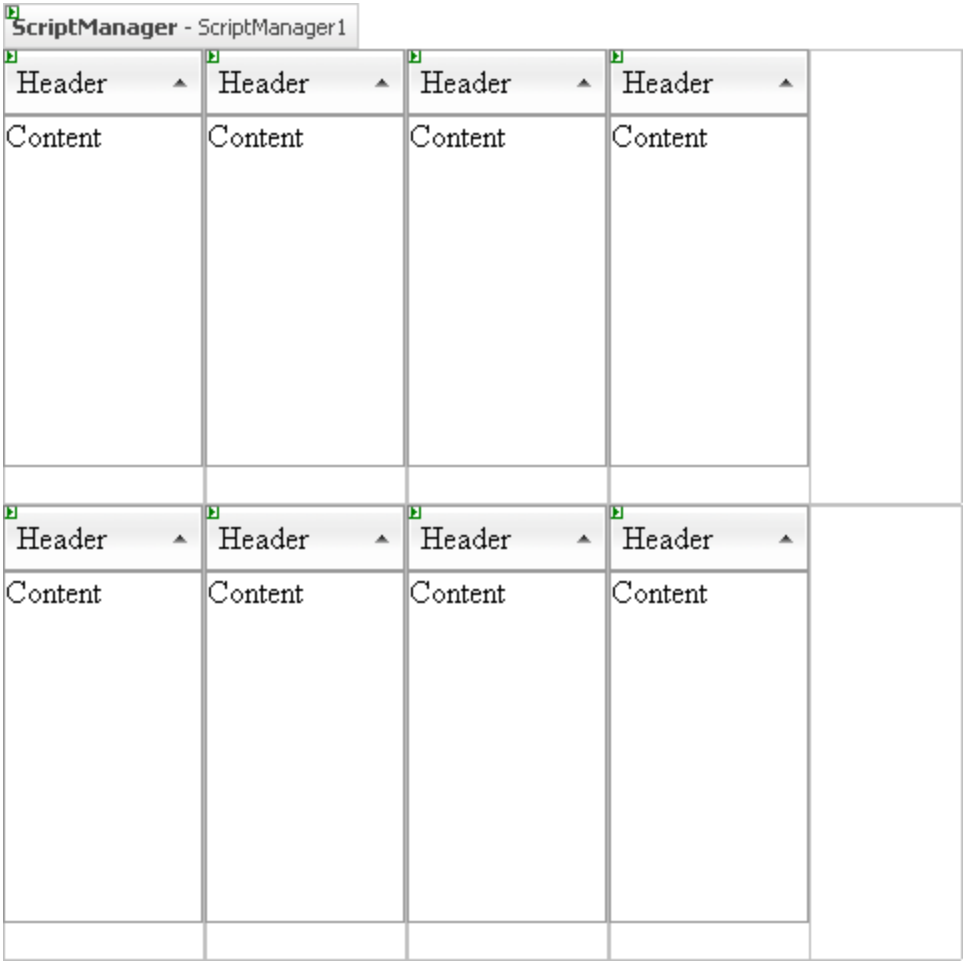
   This will create two rows, each containing four small boxes. Note that extra spaces were added within each box as placeholders.

3. Click the **Design** tab located below the Document window to switch to Design view.

   Your page should look similar to the following image:

4. Click in the top-left box to select it, navigate to the Visual Studio Toolbox, and double-click the **C1Expander** control to add the control to the box. Repeat this step with each box to add a total of eight **C1Expander** controls to the page, one in each box (**C1Expander1** to **C1Expander 4** in the top row and **C1Expander5** to **C1Expander8** in the bottom row).
5. Delete the extra spaces used as placeholders in each box.

   Your page will look similar to the following:



6. Run your application and observe that clicking on any **C1Expander**'s header will collapse the **C1Expander** control. Clicking on a collapsed **C1Expander**'s header will expand the control.

   For example, when all of the controls are collapsed the page will look at follows:

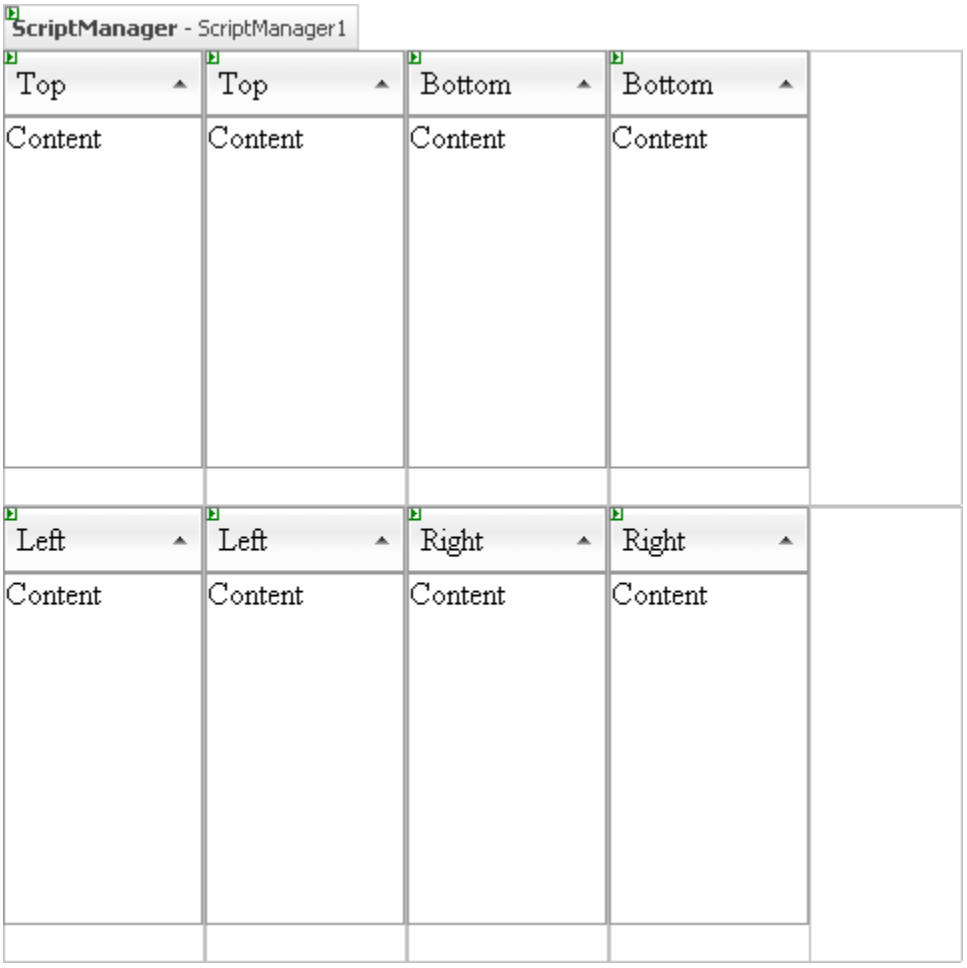
In this step you added C1Expander controls to the form. In the next step of the quick start, you'll add content to those controls.

Step 2 of 5: Adding Content to the C1Expander Control

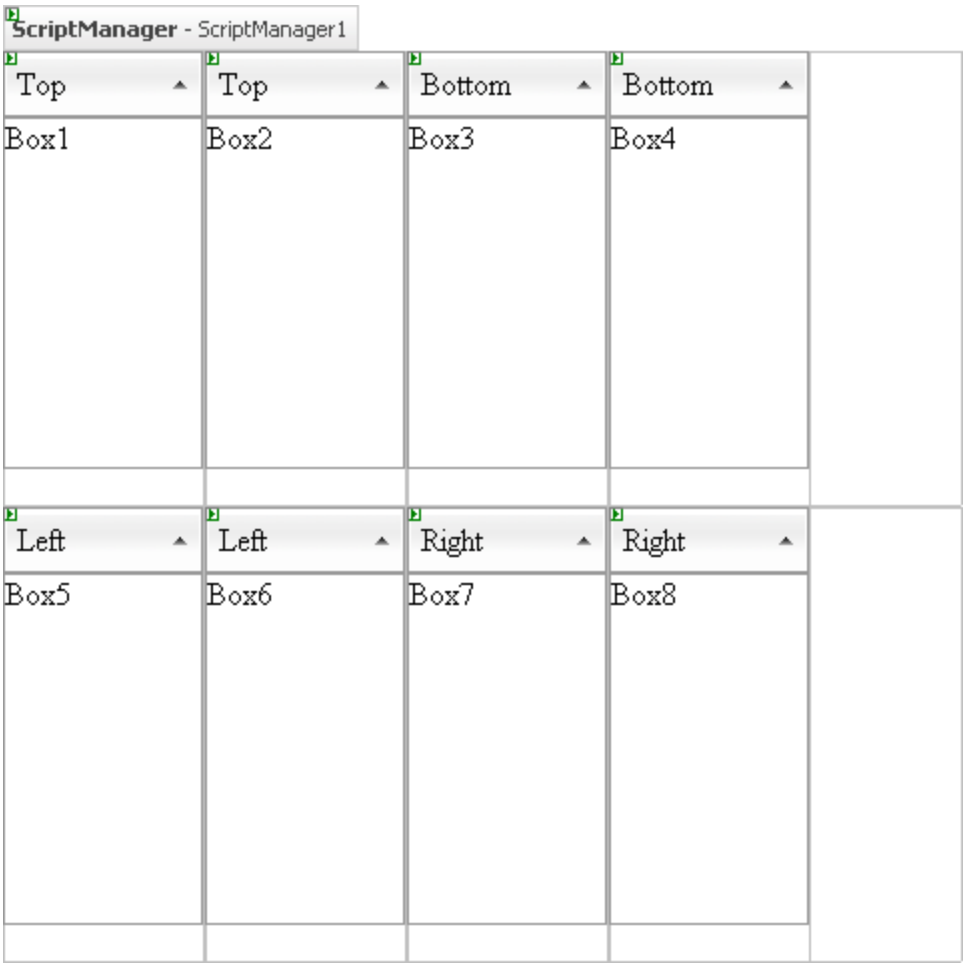
Adding content to the C1Expander control is as simple as clicking in the body of the control and typing text or adding controls. The following steps assume you've completed the Step 1 of 5: Adding C1Expander to the Page topic and added eight C1Expander controls to the page.

Complete the following steps to change the content of each C1Expander control's header and body:

1. In Design view click in the top-left **C1Expander** control's header, delete the default "Header" text and type in "Top".
2. Repeat the previous step to change the header text of each **C1Expander** control so that two controls in the first row have the heading "Top" and two have the heading "Bottom", and two controls in the second row have the heading "Left", and two have the heading "Right" as in the following image:



3. In Design view click in the top left C1Expander control's body, delete the default "Content" text and type in "Box1".
4. Repeat the previous step to change the body text of each C1Expander control so that they are numbered "Box1" to "Box4" across the first row and numbered "Box5" to "Box8" across the second row as in the following image:



In this step you added content to C1Expander controls. In the next step you'll customize the appearance and behavior of the controls.

Step 3 of 5: Customizing C1Expander's Appearance and Behavior

You can easily customize **C1Expander**'s appearance and behavior with **C1Expander**'s built-in design-time support. In the following steps you'll change the size of each **C1Expander** control and header, set the control's initial state as collapsed, and customize the expand direction of each control. The following steps assume you've completed the Step 2 of 5: Adding Content to the C1Expander Control topic.

Complete the following steps:

1. Click the top-left **C1Expander**'s smart tag (the control marked **Box1**) to open the **C1Expander Tasks** menu.
2. Set the following properties in the **C1Expander Tasks** menu:

- Uncheck the Expanded check box to set the Expanded property to **False** and set the **C1Expander**'s initial state to collapsed.
- Click the drop-down arrow next to ExpandDirection and select **Top** to set the **C1Expander**'s body to expand above the header.
- Set the **HeaderSize** to "30px" to increase the size of the header.
- Set the **Height** to "100px" to decrease the control's height.

3. Click the remaining **C1Expander**'s smart tags to open their **C1Expander Tasks** menu, and set the following properties:
   - Uncheck the Expanded check box to set the Expanded property to **False** and set the **C1Expander**'s initial state to collapsed.
   - Set the **HeaderSize** to "30px" to increase the size of the header.
   - Set the **Height** to "100px" to decrease the control's height.
   - Set the ExpandDirection property to reflect the text in each **C1Expander**'s header.

     For example the ExpandDirection of the first two **C1Expander** controls in the first row should be set to **Top** and the last two **C1Expander** controls in the first row should remain set to **Bottom**. The first two controls in the second row should be set to **Left** and the last two controls in the second row should be set to **Right**.
4. Run the application and observe that the **C1Expander** controls now initially appear collapsed and that, on clicking the header of each control, they expand in the direction indicated in the header of each control.

In this step you customized the appearance and behavior of the controls. In the next step you'll add animation effects to the controls.

Step 4 of 5: Adding Animation Effects to C1Expander

C1Expander includes over 25 built-in animation effects for when the control is expanded or collapsed. These effects can be set using the CollapseAnimation and ExpandAnimation properties. In the following steps you'll give each C1Expander control on the page different animation effects. The following steps assume you've completed the Step 3 of 5: Customizing C1Expander's Appearance and Behavior topic.

Complete the following steps:

1. Select the top left **C1Expander** control (**C1Expander1**), navigate to the Properties window and set its CollapseAnimation property to **ScrollOutToTop** and its ExpandAnimation property to **ScrollInFromTop**.
2. Repeat the previous step for each control setting the CollapseAnimation and ExpandAnimation properties for each control as you choose, for example set the CollapseAnimation and ExpandAnimation properties to the following:
   - Set **C1Expander2**'s CollapseAnimation property to **ScrollOutToBottom** and ExpandAnimation property to **ScrollInFromBottom**.
   - Set **C1Expander3**'s CollapseAnimation property to **FadeOut** and ExpandAnimation property to **FadeIn**.
   - Set **C1Expander4**'s CollapseAnimation property to **CloseHorizontally** and ExpandAnimation property to **OpenHorizontally**.
   - Set **C1Expander5**'s CollapseAnimation property to **ScrollOutToRight** and ExpandAnimation property to **ScrollInFromRight**.
   - Set **C1Expander6**'s CollapseAnimation property to **UnFold** and ExpandAnimation property to **Fold**.

- Set **C1Expander7**'s CollapseAnimation property to **Snake** and ExpandAnimation property to **Bounce**.
- Set **C1Expander8**'s CollapseAnimation property to **DropOutToTop** and ExpandAnimation property to **DropInFromTop**.

3. Run your application and click on each **C1Expander**'s header to expand the control and click on each expanded **C1Expander** control to collapse the control and view the various animation effects.

Congratulations, you've completed this quick start guide!

C1Expander Elements

This section provides a visual and descriptive overview of the elements that comprise the C1Expander control. The topics are categorized into the two distinct elements, the header element and the content element, that represent different aspects of the control.

Header Element

The **C1Expander**'s header area appears at the top of the control and initially includes a title with the text "Header". You can add content, including text, HTML content, images, and other controls, to the header area of the **C1Expander** using the header template. Elements in the header area of the control can be added and moved on the control through a simple drag-and-drop operation.

The following image labels the header area of the **C1Expander** control:

Note the expand/collapse indicator that appears and indicates the header's expand and collapse direction.

You can use the following element to customize the header area of the **C1Expander** control:

- **Header**

Header

You can add content to the header area of the control at design time. To add content, simply click in the header area of the control and begin typing, or add images or controls to the header area as you would normally.

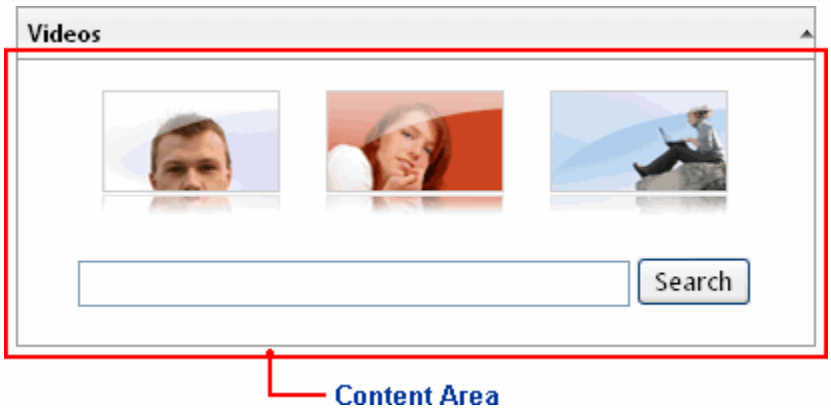
When you add content to the **C1Expander**'s header and switch to Source view you will notice that your content appears inside `<Header>` tags inside the `<cc1:C1Expander>` tags:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px" Width="100px">
    <Header>
        Header
    </Header>
    <Content>
        Content
    </Content>
</cc1:C1Expander>
```

Content Element

The **C1Expander**'s content area initially consists of an empty area with the text "Content". In the content area you can add rich text through custom HTML content, URL links through the **ContentUrl** property, and add arbitrary controls through the content template. Elements in the content area of the control can be added and moved on the control through a simple drag-and-drop operation.

The following image labels the content area of the **C1Expander** control:



You can use the following elements to customize the content area of the **C1Expander** control:

- **Content**
- **ContentUrl**

Content

You can add content to the control at design time. To add content, simply click in the content area of the control and begin typing, or add images or controls to the content areas as you would normally.

When you add content to the **C1Expander** control and switch to Source view you will notice that your content appears inside `<Content>` tags inside the `<cc1:C1Expander>` tags:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px" Width="100px">
    <Header>
        Header
    </Header>
    <Content>
        Content
    </Content>
</cc1:C1Expander>
```

ContentUrl

You can use the **ContentUrl** property to set external content to appear within the content area of the **C1Expander** control.

Design-Time Support

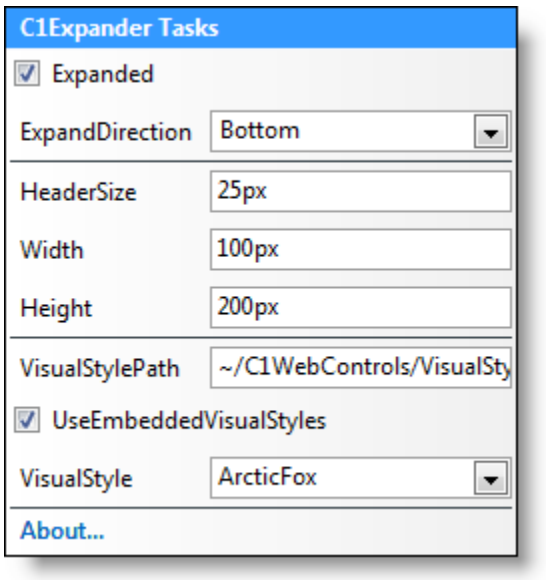
The following sections describe how to use **C1Expander**'s design-time environment to configure the **C1Expander** control.

C1Expander Smart Tag

In Visual Studio, the **C1Expander** control includes a smart tag. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in **C1Expander**.

The **C1Expander** control provides quick and easy access to common properties through its smart tag.

To access the **C1Expander Tasks** menu, click on the smart tag in the upper-right corner of the **C1Expander** control. This will open the **C1Expander Tasks** menu.



The **C1Expander Tasks** menu operates as follows:

- **Expanded**

  Selecting the **Expanded** check box sets the Expanded property to **True** and the **C1Expander** control will initially appear expanded on page load. By default the Expanded is selected. If you want the control to appear collapsed initially uncheck this check box.

- **Expand Direction**

  Selecting the **ExpandDirection** drop-down list allows you to change the direction the **C1Expander** control will expand. The ExpandDirection property can be set to **Top**, **Right**, **Bottom**, and **Left**. By default the ExpandDirection property is set to **Bottom**.

- **HeaderSize**

  Changing the **HeaderSize** value changes the height of the header using the **HeaderSize** property. By default the **HeaderSize** property and the height of the header is set to 25 pixels.

- **Width**

  Changing the **Width** value changes the width of the **C1Expander** control using the **Width** property. By default the **Width** property and the width of the control is set to 100 pixels.

- **Height**

  Changing the **Height** value changes the height of the **C1ExpanderControl** control using the **Height** property. By default the **Height** property and the height of the control is set to 200 pixels.

- **VisualStylePath**

The **VisualStylePath** property specifies the location of the visual styles used for the control. By default, embedded visual styles are located in **~/C1WebControls/VisualStyles**. If you create a custom style, add it to this location **~/VisualStyles/StyleName/C1Expander/styles.css**, set the **VisualStylePath** property to **~/VisualStyles**, and set the **VisualStyle** property to **StyleName** (assuming that **StyleName** is the name used to define the style in the style.css file). Uncheck the **UseEmbeddedVisualStyles** property.

- **UseEmbeddedVisualStyles**

  This check box is checked by default so that the internal visual styles, such as **ArcticFox** and **Vista** can be used. If you want to use your own custom styles, uncheck this check box and specify the location of your visual styles using the **VisualStylePath** property.

- **Visual Style**

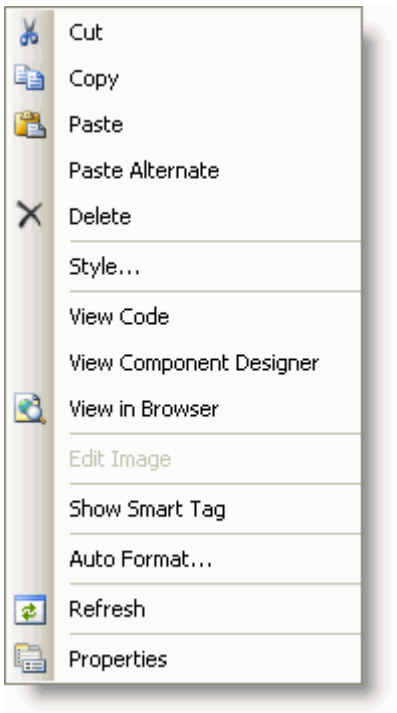
  Clicking the **Visual Style** drop-down box allows you to select from various visual schemes. For more information about available visual styles, see Visual Styles.

- **About**

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

C1Expander Context Menu

Right-click anywhere on the list to display the C1Expander context menu, which is a context menu that Visual Studio provides for all .NET controls, although the C1Expander context menu has a few extra features.



The context menu commands operate as follows:

- **Show Smart Tag**

  Clicking this item shows the **C1Expander Tasks** menu. For more information on how to use the smart tag and available features in the Tasks menu, see C1Expander Smart Tag.

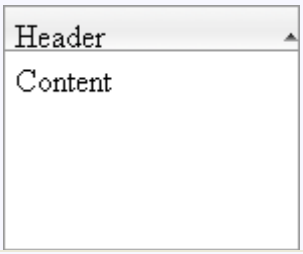
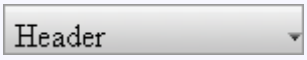
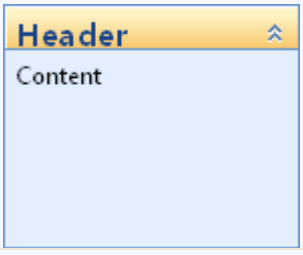
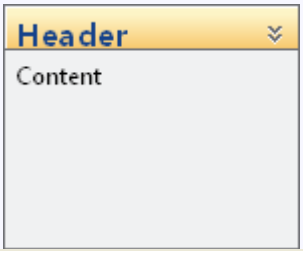
Expander for ASP.NET Appearance

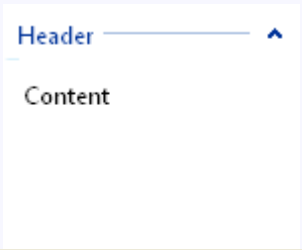
There are several options for customizing the appearance of the C1Expander control. The following sections describe how to change the appearance of the control through built-in Visual Styles as well as how to customize other elements of the C1Expander control.

Visual Styles

C1Expander includes Visual Styles allowing you to easily change the control's appearance. The control includes several built-in visual styles, including Vista and Office 2007 styles, to quickly style your application. You can easily change Visual Styles from the **C1Expander Tasks** menu, from the Properties window, and in code. For more information on changing Visual Styles see the Setting the Visual Style topic.

The following Visual Styles are included in **Expander for ASP.NET**:

Visual Style	Expanded Preview	Collapsed Preview
ArcticFox (default)		
Office2007Black		
Office2007Blue		
Office2007Silver		

Vista		

Customizing the Control's Appearance

If you choose to completely customize the appearance of the C1Expander you may not wish to use any of the available built-in Visual Styles. In that case, to override any visual styles with your own custom appearance, you will need to set the C1Expander.**UseEmbeddedVisualStyles** property to **False**.

By default C1Expander.**UseEmbeddedVisualStyles** property is **True** and Visual Styles are used. Any customizations you make while using Visual Styles will simply set specific elements in the control's appearance on top of the current Visual Style. To start customizing the control's appearance from scratch set C1Expander.**UseEmbeddedVisualStyles** to **False** and set your own styles.

Header and Content Templates

The content of the header and content areas of the C1Expander control can be controlled by using templates. C1Expander has two special properties, **Header**, and **Content**, that can be used to apply templates to the header and content areas of the C1Expander control. Header and content templates are useful for additionally customizing your C1Expander control to your application and for adding content to the header and content areas of the control.

Expanding and Collapsing C1Expander

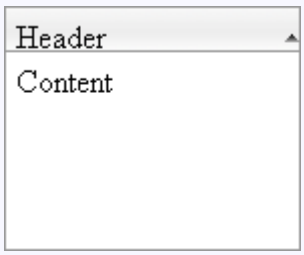
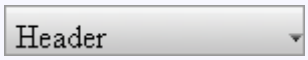
There are several options for customizing the way that the C1Expander control appears and how the expanding and collapsing animations of the control are handled. The following sections describe how to set the initial expand state, set the direction to which the control expands, different animation expand and collapse effects and expand and collapse duration and easing.

Initial Expand State

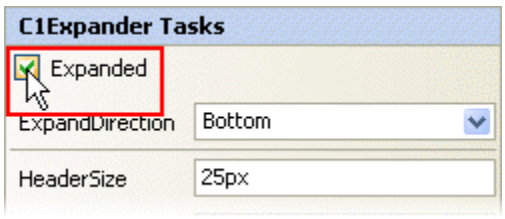
You can choose how the C1Expander control appears on the form initially using the Expanded property. By default the Expanded property is set to **True** and the control initially appears expanded when the page is loaded. You can set the Expanded property to **False** to set the control to initially appear collapsed when the page is loaded.

So, for example, in the **ArcticFox** theme:

Expanded Setting	Initial State Preview

True (default)	
False	

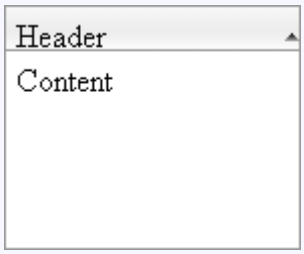
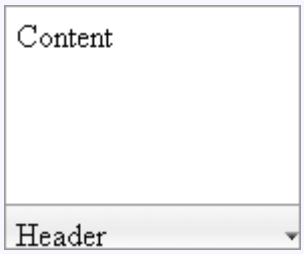
You can set the Expanded property in the **C1Expander Tasks** menu, in the Properties window, in Source view, and in Code. To set the Expanded property in the **C1Expander Tasks** menu, uncheck or check the **Expanded** box:

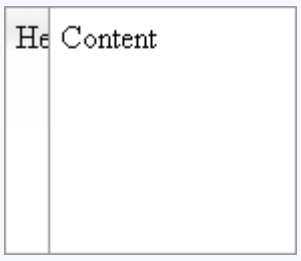
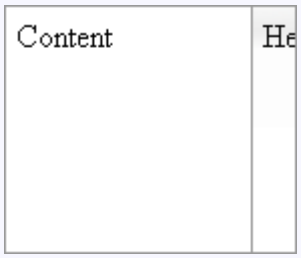


Expand Direction

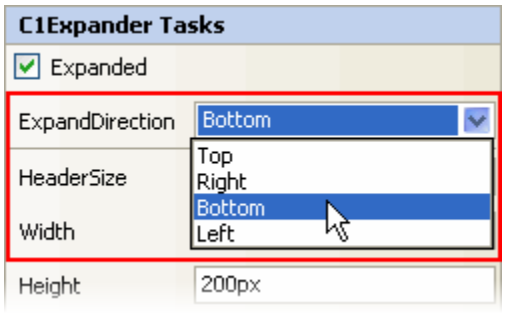
C1Expander includes the option to specify the expand direction using the ExpandDirection property. In addition to setting how the direction the control expands, changing the ExpandDirection also changes the header's orientation to the content area of the control. By default the ExpandDirection property is set to **Bottom** and the control expands from top to bottom.

So, for example, in the **ArcticFox** theme:

ExpandDirection Setting	Preview
Bottom (default)	
Top	

Right	
Left	

You can set the ExpandDirection property in the **C1Expander Tasks** menu, in the Properties window, in Source view, and in Code. To set the ExpandDirection property in the **C1Expander Tasks** menu, select the **ExpandDirection** drop-down box:



Expand and Collapse Animation

C1Expander includes twenty-six built-in animation options that allow you to completely customize interaction with the C1Expander control. You can change how the control expands and collapses by setting the ExpandAnimation and CollapseAnimation properties. By default the ExpandAnimation and CollapseAnimation properties are set to **None** and the control expands and collapses without an animation effect used.

Sample Project Available

For a demonstration of each animation effect possible in the C1Expander control, see the **Animation** page of the **ControlExplorer** sample.

The following table describes each animation effect choice. Note that while you can use all animations to expand or collapse the control, the terms "expands" and "collapses" were used to indicate which selection is suggested for each animation:

Name	Description
None (default)	No animation used.
FadeIn	Expands body of the control so that it appears to fade in.
FadeOut	Collapses the body of the control, so that it appears to fade out.
ScrollInFromTop	Expands the body of the control, scrolling into view from the top.
ScrollInFromRight	Expands the body of the control, scrolling into view from the right.
ScrollInFromBottom	Expands the body of the control, scrolling into view from the bottom.
ScrollInFromLeft	Expands the body of the control, scrolling into view from the left.
ScrollOutToTop	Collapses the body of the control, scrolling out of view to the top.
ScrollOutToRight	Collapses the body of the control, scrolling out of view to the right.
ScrollOutToBottom	Collapses the body of the control, scrolling out of view to the bottom.
ScrollOutToLeft	Collapses the body of the control, scrolling out of view to the left.
Fold	Collapses the body of control vertically and then horizontally so it appears to fold.
UnFold	Expands the body of control horizontally and then vertically so it appears to unfold.
OpenVertically	Expands the body of control vertically from the center of the body area.
CloseVertically	Collapses the body of control vertically from the center of the body area.
OpenHorizontally	Expands the body of control horizontally from the center of the body area.
CloseHorizontally	Collapses the body of control horizontally from the center of the body area.
Shake	Expands or Collapses the body of control with a horizontal shaking motion.
Bounce	Expands or Collapses the body of control with a vertical bouncing motion.
DropInFromTop	Expands the body of the control from below the control to the top.
DropInFromRight	Expands the body of the control from the left of the control to the right.

DropInFromBottom	Expands the body of the control from above the control to the bottom.
DropInFromLeft	Expands the body of the control from the right of the control to the left.
DropOutToTop	Collapses the body of the control out to above the control.
DropOutToRight	Collapses the body of the control out to the right of the control.
DropOutToBottom	Collapses the body of the control out to below the control.
DropOutToLeft	Collapses the body of the control out to the left of the control.

Expand and Collapse Transitions

C1Expander includes thirty-one built-in animation transition options that allow you to customize how animation effects are transitioned in the C1Expander control. You can change how the control expands and collapses by setting the ExpandEasing and CollapseEasing properties. By default the ExpandEasing and CollapseEasing properties are set to **EaseLinear** and the control expands and collapses with a smooth linear transition effect.

Sample Project Available

For a demonstration of each transition effect possible in the C1Expander control, see the **Animation** page of the **ControlExplorer** sample.

The following table describes each transition effect choice:

Name	Description
EaseLinear (default)	Linear easing. Moves smoothly without acceleration or deceleration.
EaseOutElastic	Elastic easing out. Starts quickly and then decelerates.
EaseInElastic	Elastic easing in. Starts slowly and then accelerates.
EaseInOutElastic	Elastic easing in and out. Starts slowly, accelerates, and then decelerates.
EaseOutBounce	Bouncing easing out. Starts quickly and then decelerates.
EaseInBounce	Bouncing easing in. Starts slowly and then accelerates.
EaseInOutBounce	Bouncing easing in and out. Starts slowly, accelerates, and then decelerates.
EaseOutExpo	Exponential easing out. Starts quickly and then decelerates.

EaseInExpo	Exponential easing in. Starts slowly and then accelerates.
EaseInOutExpo	Exponential easing in and out. Starts slowly, accelerates, and then decelerates.
EaseOutQuad	Quadratic easing out. Starts quickly and then decelerates.
EaseInQuad	Quadratic easing in. Starts slowly and then accelerates.
EaseInOutQuad	Quadratic easing in and out. Starts slowly, accelerates, and then decelerates.
EaseOutSine	Sinusoidal easing out. Starts quickly and then decelerates.
EaseInSine	Sinusoidal easing in. Starts slowly and then accelerates.
EaseInOutSine	Sinusoidal easing in and out. Starts slowly, accelerates, and then decelerates.
EaseOutCirc	Circular easing out. Starts quickly and then decelerates.
EaseInCirc	Circular easing in. Starts slowly and then accelerates.
EaseInOutCirc	Circular easing in and out. Starts slowly, accelerates, and then decelerates.
EaseOutCubic	Cubic easing out. Starts quickly and then decelerates.
EaseInCubic	Cubic easing in. Starts slowly and then accelerates.
EaseInOutCubic	Cubic easing in and out. Starts slowly, accelerates, and then decelerates.
EaseOutQuint	Quintic easing out. Starts quickly and then decelerates.
EaseInQuint	Quintic easing in. Starts slowly and then accelerates.
EaseInOutQuint	Quintic easing in and out. Starts slowly, accelerates, and then decelerates.
EaseOutBack	Back easing out. Starts quickly and then decelerates.
EaseInBack	Back easing in. Starts slowly and then accelerates.
EaseInOutBack	Back easing in and out. Starts slowly, accelerates, and then decelerates.
EaseOutQuart	Quartic easing out. Starts quickly and then decelerates.
EaseInQuart	Quartic easing in. Starts slowly and then

	accelerates.
EaseInOutQuart	Quartic easing in and out. Starts slowly, accelerates, and then decelerates.

Expand and Collapse Duration

You can set how long each C1Expander expand and collapse animation effect takes by using the ExpandDuration and CollapseDuration properties. The unit of time used for specifying animation effect duration is in milliseconds, and the default setting for both the ExpandDuration and CollapseDuration properties is **500** milliseconds (so half a second). Increase this value for a longer animation effect, and decrease this number for a shorter animation effect.

Sample Project Available

For a demonstration of setting the ExpandDuration and CollapseDuration properties, see the **Animation** page of the **ControlExplorer** sample.

Client-Side Functionality

C1Expander includes a robust client-side object model, where a majority of server-side properties can be set on the client-side and client-side events are described in the properties window.

When a C1Expander control is rendered, an instance of the client side control will be created automatically. This means that you can enjoy the convenience of accessing properties and methods of the C1Expander control without having to postback to the server.

For example, suppose a C1Expander control with name **C1Expander1** is hosted on Web page; when the page is rendered, a corresponding client side object will be created. Use the following syntax to get the client object:

```
$get("C1Expander1").control
```

OR

```
$find("C1Expander1")
```

By using **C1Expander**'s client-side functionality, you can implement many features in your Web page without the need to take up time by sending information to the Web server. Thus, using client-side methods and events can increase the efficiency of your Web site.

Client-Side Properties

The following conventions are used when accessing client object properties:

- Server properties on the client are implemented as a pair of Get- and Set- methods.
- Method names must start with "get_" (Get-method) and "set_" (Set-method) followed with the server property name. The first letter of the server property name must be lowercase (camel case).

For example in the code below the C#, Visual Basic, and JavaScript examples are equivalent:

- Visual Basic

```
Dim s As String = C1Expander1.AccessKey
C1Expander1.AccessKey = s
```

- C#

```
string s = C1Expander1.AccessKey;
C1Expander1.AccessKey = s;
```

- JavaScript

```
var oExpander = $get("C1Expander1").control;
var s = oExpander.get_accessKey();
oExpander.set_accessKey(s);
```

For an example of setting the **Height** and **Width** properties at run time, see Setting the Height and Width at Run Time.

C1Expander includes a rich client-side object model in which several properties can be set on the client side. For information about these client-side methods and what properties can be set on the client side, see the C1Expander Reference.

Expander 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# Sample

The following pages within the **ControlExplorer** sample installed with **ComponentOne Studio for ASP.NET** detail the C1Expander control's functionality:

Sample	Description
Animation	Displays various animation effects as well as expand and collapse transitions, durations, and delays.
ContentUrl	Demonstrates the **ContentUrl** property, allowing you to display external content in the C1Expander control.
ExpandDirection	Shows the four different expand directions included in C1Expander: top, right, bottom, and left.
VisualStyles	Displays the built-in C1Expander control Visual Styles including ArcticFox, Office2007Black, Office2007Blue, Office2007Silver, and Vista.

Expander 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. By following the steps outlined in the help, you will be able to create projects demonstrating a variety of C1Expander's features, and get a good sense of what the C1Expander control can do.

Each topic provides a solution for specific tasks using the C1Expander control. Each task-based help topic also assumes that you have created a new ASP.NET AJAX-Enabled project and added references to the appropriate assemblies. For additional information on this topic, see Creating an AJAX-Enabled ASP.NET Project.

Note: Expander for ASP.NET requires Microsoft ASP.NET AJAX Extensions installed and a **ScriptManager** on the page before the C1Expander control is placed on the page. You must create an ASP.NET AJAX-Enabled Project so that the **ScriptManager** and Microsoft AJAX Extensions are included on the page. For more information, see Creating an AJAX-Enabled ASP.NET Project. For more information about Microsoft ASP.NET AJAX Extensions, see http://ajax.asp.net/. For information about the **ScriptManager**, see MSDN.

Creating a C1Expander in Code

Creating a C1Expander control in code is fairly simple. In the following steps you'll add a **PlaceHolder** control to the page, add an import statement, add and customize the C1Expander, and add the control to the **PlaceHolder**.

Complete the following steps:

1. In Design view, navigate to the Visual Studio Toolbox and add a **PlaceHolder** control to your page.
2. Double-click the page to create the **Page_Load** event and switch to Code view.
3. Add the following statement to the top of the Code Editor to import the appropriate namespace:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1Expander
     ```
   - C#
     ```
     using C1.Web.UI.Controls.C1Expander;
     ```
4. Add the following code to the **Page_Load** event to create and customize the **C1Expander** control.
   - Visual Basic
     ```
     ' Create a new C1Expander.
     Dim C1E As New C1Expander
     ' Set the control's size, appearance, and content.
     C1E.VisualStyle = "Office2007Blue"
     C1E.Height = 200
     C1E.Width = 200
     C1E.ContentUrl = "http://www.wikipedia.com/"
     ' Add the C1Expander to the PlaceHolder control.
     PlaceHolder1.Controls.Add(C1E)
     ```
   - C#
     ```
     // Create a new C1Expander.
     C1Expander C1E = new C1Expander();
     // Set the control's size, appearance, and content.
     C1E.VisualStyle = "Office2007Blue";
     C1E.Height = 200;
     C1E.Width = 200;
     C1E.ContentUrl = "http://www.wikipedia.com/";
     // Add the C1Expander to the PlaceHolder control.
     PlaceHolder1.Controls.Add(C1E);
     ```

Run your application and observe:

The C1Expander control was created and displays external content:

Setting Post Back

You can easily set whether or not C1Expander automatically posts back to the server by using the **AutoPostBack** property. By default the **AutoPostBack** property is set to **False** and the **C1Expander** control does not automatically post back to the server; to allow **C1Expander** to post back to the server, set **AutoPostBack** to **True**.

In Source View

In Source view add `AutoPostBack="True"` to the `<cc1:C1Expander>` tag so it appears similar to the following:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px" Width="100px" AutoPostBack="True">
```

In Design View

In Design view select the **C1Expander** control and in the Properties window set the **AutoPostBack** property to **True**.

In Code

Add the following code to the **Page_Load** event to set the **AutoPostBack** property to **True**:

- Visual Basic

```
Me.C1Expander1.AutoPostBack = True
```

- C#

```
this.C1Expander1.AutoPostBack = true;
```

Suppressing Header Post Back

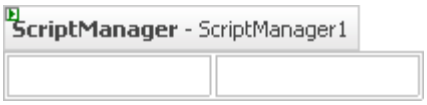
You can easily set whether or not the header of the C1Expander automatically posts back to the server by using the **SuppressHeaderPostbacks** property. By default the **SuppressHeaderPostbacks** property is set to **False** and any client-size click handlers in the header are not suppressed; to suppress client-size click handlers in the header of a **C1Expander**, set **SuppressHeaderPostbacks** to **True**.

In this topic you'll create two **C1Expander**s with hyperlinks in the header area, one **C1Expander** with **SuppressHeaderPostbacks** set to **True** and one by default with **SuppressHeaderPostbacks** set to **False**.

Complete the following steps:

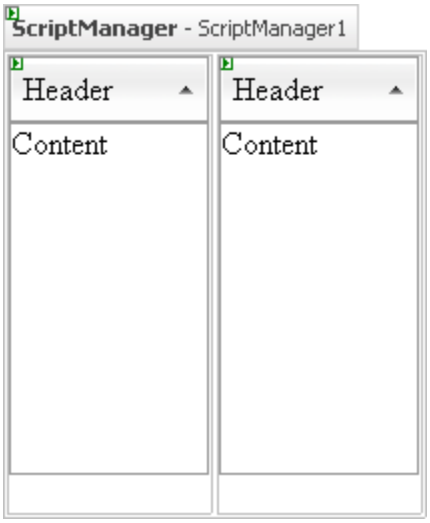
1. Select **Layout | Inset Table** and in the **Insert Table** dialog box create a table with one row and two columns.

   Your page should now look similar to the following:



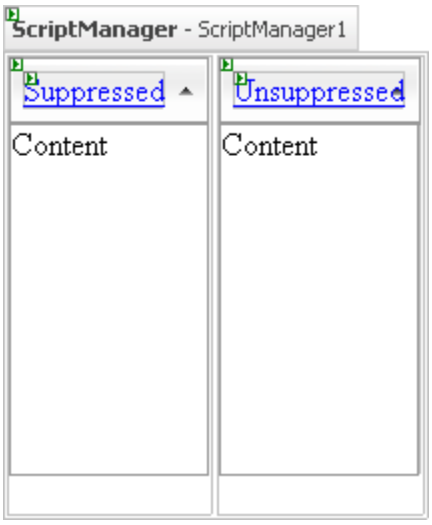
2. Click within the left cell of the table and in the Visual Studio Toolbox double-click the **C1Expander** to add the control to the left cell.
3. Click within the right cell of the table and in the Visual Studio Toolbox double-click the **C1Expander** to add the control to the right cell.

   The page should now look similar to the following:



4. Delete the default "Header" text within each **C1Expander**'s header and from the Toolbox add a **HyperLink** control to each header.
5. Select each **HyperLink** control and set the following properties in the Properties window:
   - Set both the **HyperLink** controls' **NavigateUrl** properties to "http://www.componentone.com/".
   - Set both the **HyperLink** controls' **Target** properties to "_blank".
   - Set the left-side **HyperLink** control's **Text** property to "Suppressed".
   - Set the right-side **HyperLink** control's **Text** property to "Unsuppressed".

   The page will now look similar to the following:



6. Select the left **C1Expander** and in the Properties window set its **SuppressHeaderPostbacks** property to **True**.

   By default the right-side **C1Expander**'s **SuppressHeaderPostbacks** property will remain set to **False**.

Run your application and observe:

1. Click the hyperlink in the left **C1Expander**'s header. The hyperlink does not open.

   Note that you can right-click the link to manually open the Web site, but because the **SuppressHeaderPostbacks** property was set to **True**, client-size click handlers in the header of the **C1Expander** were suppressed.

2. Click the hyperlink in the right **C1Expander**'s header. The hyperlink opens the ComponentOne Web site in a new page as expected.

Adding Keyboard Support

The **C1Expander** control lets you easily add keyboard accessibility to the control. You can use **AccessKey** property to set how the user navigates to the control and through your user interface.

In the following examples you'll set the **AccessKey** property to **e** so that pressing the ALT+E key combination at run time brings the **C1Expander** control into focus.

In Source View

In Source view add `AccessKey="e"` to the `<cc1:C1Expander>` tag so it appears similar to the following:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px" Width="100px" AccessKey="e">
```

In Design View

In Design view, select the **C1Expander** control and in the Properties window set the **AccessKey** property to **e**.

In Code

Add the following code to the **Page_Load** event to set the **AccessKey** property to **e**:

- Visual Basic

```
Me.C1Expander1.AccessKey = "e"
```

- C#

```
this.C1Expander1.AccessKey = "e";
```

Setting the Visual Style

The control includes several built-in visual styles, including Vista and Office 2007 styles, to style your application. For more information about available styles, see Visual Styles. You can easily change Visual Styles in Source view, from the **C1Expander Tasks** menu, from the Properties window, and in code.

In Source View

In Source view add `VisualStyle="Vista"` to the `<cc1:C1Expander>` tag so it appears similar to the following:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px" Width="100px" VisualStyle="Vista">
```

From the Tasks Menu

You can access Visual Styles from the **C1Expander Tasks** menu:

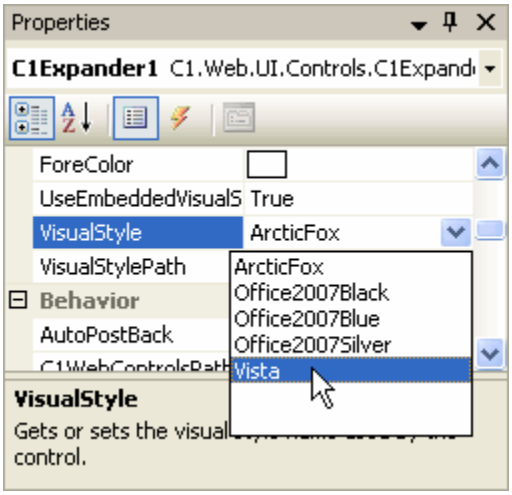
1. Click on the **C1Expander**'s smart tag to open the **C1Expander Tasks** menu.
2. Click the Visual Style drop-down box and select a style to apply, for example **Vista**.

   The style you choose will be applied to the control.

From the Properties Window

You can select a Visual Style to apply from the Properties window:

1. Click on the **C1Expander** to select it.
2. Navigate to the Properties window and select the drop-down arrow next to the **VisualStyle** property.
3. Select a style to apply, for example **Vista**.



The Visual Style you chose will be applied to the **C1Expander**.

In Code

Add the following code to the **Page_Load** event to set the **VisualStyle** property to **Vista**:

- Visual Basic

```
Me.C1Expander1.VisualStyle = "Vista"
```

- C#

```
this.C1Expander1.VisualStyle = "Vista";
```

Changing the Font

To further customize your application, you can easily set the font of the **C1Expander** to be similar to the font selected throughout your application. You can easily change the font in Source view, from the Properties window, and in code.

In Source View

In Source view add text to the `<cc1:C1Expander>` tag so it appears similar to the following:

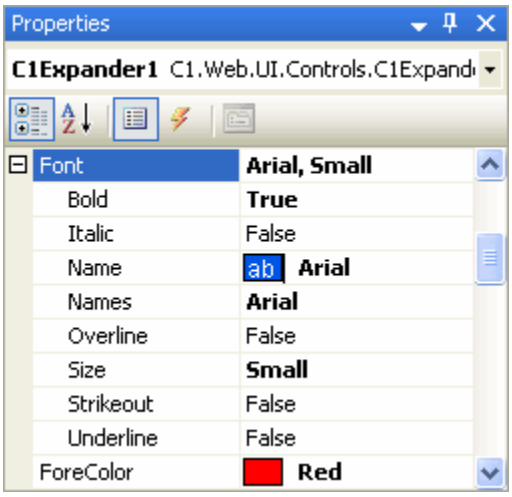
```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px" Width="100px" Font-Bold="True" Font-Names="Arial" Font-
Size="Small" ForeColor="Red" >
```

When you run your application, the text in the body of the **C1Expander** control will now appear small, bold, red, and in the Arial font.

In Design View

You can customize the font's appearance from the Properties window:

1. Click on the **C1Expander** to select it.
2. Navigate to the Properties window, expand the **Font** node.
3. Set the following properties:
   - Set **Font.Bold** to **True**.
   - Set **Font.Name** to **Arial**.
   - Set **Font.Size** to **Small**.
   - Set **ForeColor** to **Red**.



When you run your application, the text in the body of the **C1Expander** control will now appear small, bold, red, and in the Arial font.

In Code

Add the following code to the **Page_Load** event to customize the font's appearance:

- Visual Basic

```
Me.C1Expander1.Font.Bold = True
Me.C1Expander1.Font.Name = "Arial"
```

```
Me.C1Expander1.Font.Size = FontUnit.Small
Me.C1Expander1.ForeColor = Color.Red
```

- C#

```
this.C1Expander1.Font.Bold = true;
this.C1Expander1.Font.Name = "Arial";
this.C1Expander1.Font.Size = FontUnit.Small;
this.C1Expander1.ForeColor = Color.Red;
```

Note: The **System.Drawing** namespace must be imported for the above code to work correctly.

When you run your application, the text in the body of the **C1Expander** control will now appear small, bold, red, and in the Arial font.

Setting the Content Background Color

You can customize the appearance of the C1Expander by using styles. You can create a new style front scratch or customize the built-in styles. In this topic you'll customize the **ArcticFox** theme by adding a style to change the background color of the control. The following topic assumes you've added a C1Expander to the page.

Complete the following steps:

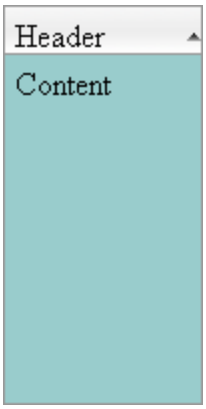
1. Click on the **C1Expander**'s smart tag to open the **C1Expander Tasks** menu.
2. From the Tasks menu, click the **VisualStyle** drop-down arrow and select the **ArcticFox** scheme. The style you'll add in the next steps will be added on top of this theme.
3. Switch to Source view.
4. Add the following style markup between the <head> and </head> tags at the top of the document:

```
<style>
.C1Expander_ArcticFox .C1eContentPanel {
   background: #99CCCC;;
   display:block;
   margin:0;
   padding:0;
}
</style>
```

   This will set the background color of the control's content area.

5. Save and run your project.

   The **C1Expander** control will appear in the **ArcticFox** theme and the control's content area will appear with a green background:



Resizing the Header

You can easily change the height of the header, by setting the **HeaderSize** property. By default the height of the header is set to **25px**. You can easily change the header's height in Source view, from the **C1Expander Tasks** menu, from the Properties window, or in code.

In Source View

In Source view change `HeaderSize="25px"` in the `<cc1:C1Expander>` tag to the size you wish to set the header's height to, for example:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="100px"
Height="200px" Width="100px">
```

The above will resize the header's height to 100 pixels tall.

From the Tasks Menu

You can set the header's height from the **C1Expander Tasks** menu:

1. Click on the **C1Expander**'s smart tag to open the **C1Expander Tasks** menu.
2. In the **HeaderSize** textbox, replace "25px" with the size you want the header to be, for example "100px".
3. Click outside of the **C1Expander Tasks** menu to close the menu and set the height of the header.

From the Properties Window

You can change the **HeaderSize** property to set the header's height in the Properties window:

1. Click on the **C1Expander** to select it.
2. Navigate to the Properties window and, if needed, expand the **Layout** node to locate the **HeaderSize** property.
3. Next to **HeaderSize**, replace "25px" with the size you want the header to be, for example "100px".
4. Press the ENTER key or click outside of the Properties window for the header height you set to be applied to the **C1Expander**.

In Code

Add the following code to the **Page_Load** event to set the **HeaderSize** property to 100 pixels:

- Visual Basic

```
Me.C1Expander1.HeaderSize = 100
```

- C#

```
this.C1Expander1.HeaderSize = 100;
```

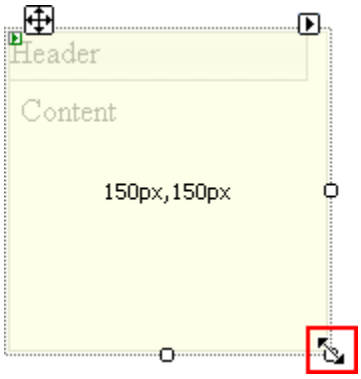
Resizing the Control

You can easily change the height and width on the **C1Expander** by setting the **Height** and **Width** properties. By default the height of the control is set to **200px** and the width of the control is set to **100px**. You can easily change the control's height and width in Design view, Source view, from the **C1Expander Tasks** menu, from the Properties window, or in code. See the Setting the Height and Width at Run Time topic for resizing the control on the client side.

In Design View

To resize the **C1Expander** control in Design view, click the bottom-right corner of the control and perform a drag-and-drop operation to set the control's size.

For example, in the following image the control has been resized to 150 pixels wide and 150 pixels tall:



In Source View

In Source view change `Height="200px" Width="100px"` in the `<cc1:C1Expander>` tag to the size you wish to set the control's height and width to, for example:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="150px" Width="150px">
```

The above will resize the control to 150 pixels tall and 150 pixels wide.

From the Tasks Menu

You can set the control's height and width from the **C1Expander Tasks** menu:

1. Click on the **C1Expander**'s smart tag to open the **C1Expander Tasks** menu.
2. In the **Width** textbox, replace "100px" with the size you want the control's width to be, for example "150px".
3. In the **Height** textbox, replace "200px" with the size you want the control's height to be, for example "150px".
4. Click outside of the **C1Expander Tasks** menu to close the menu and set the width and height of the control.

From the Properties Window

You can change the **Height** and **Width** properties to set the control's height and width in the Properties window:

1. Click on the **C1Expander** to select it.
2. Navigate to the Properties window and if needed expand the **Layout** node to locate the **Height** and **Width** properties.
3. Next to **Height**, replace "200px" with the size you want the control's height to be, for example "150px".

4. Next to **Width**, replace "100px" with the size you want the control's width to be, for example "150px".
5. Press the ENTER key or click outside of the Properties window for the height and width you set to be applied to the **C1Expander** control.

In Code

Add the following code to the **Page_Load** event to set the **Height** and **Width** properties to 150 pixels:

- Visual Basic
```
Me.C1Expander1.Height = 150
Me.C1Expander1.Width = 150
```
- C#
```
this.C1Expander1.Height = 150;
this.C1Expander1.Width = 150;
```

Displaying External Content

You can use the **ContentUrl** property to display external content in the **C1Expander** control. This means that you can have the content of another Web page in your project or even the content of a Web site outside of your project appear in a **C1Expander**. To set the **ContentUrl** property at run time see, Setting the ContentUrl Property and Header at Run Time. You can easily set the **ContentUrl** property in Source view, from the Properties window, or in code.

In Source View

In Source view change add `ContentUrl` in the `<cc1:C1Expander>` tag to set the external URL to display in the **C1Expander** control, for example:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px" Width="100px" ContentUrl="http://www.google.com/">
```

The above will set the Google Web site to appear in the **C1Expander**.

From Design View

You can change the **ContentUrl** property to the URL to be displayed in the **C1Expander** via the Properties window:

1. Click on the **C1Expander** to select it.
2. Navigate to the Properties window and if needed expand the **Misc** node to locate the **ContentUrl** property.
3. Next to **ContentUrl**, type the URL for the page you wish to appear in the control, for example "http://www.google.com/".
4. Press the ENTER key or click outside of the Properties window for **ContentUrl** you set to be applied to the **C1Expander** control.

In Code

Add the following code to the **Page_Load** event to set the **ContentUrl** property to display external content in the **C1Expander** control:

- Visual Basic
```
Me.C1Expander1.ContentUrl = "http://www.google.com/"
```
- C#
```
this.C1Expander1.ContentUrl = "http://www.google.com/";
```

Setting the ContentUrl Property and Header at Run Time

The **ContentUrl** property allows you display external content, such as internal or external Web pages, in the **C1Expander** control. In this topic you'll set the **C1Expander** control's **ContentUrl** property and header content to change on the client side. If, instead, you'd like set the **ContentUrl** property in Source view, from the Properties window, or in code, see the Displaying External Content topic.

To set the control's header content and **ContentUrl** property at run time, create a new AJAX-Enabled Web site project and complete the following steps:

1. Navigate to the Visual Studio Toolbox and double-click the **C1Expander** icon to add the control to your page.
2. Click on the **Source** tab to switch to Source view, and note that the body of the page looks similar to the following:

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px"
Width="100px"><Header>Header</Header><Content>Content</Content></cc1:C1
Expander>
    </div>
    </form>
</body>
```

3. In the `<cc1:C1Expander>` tag, delete the default "Content" text between the `<Content></Content>` tags.
4. In the `<cc1:C1Expander>` tag, change `Height="200px" Width="100px"` to `Height="250px" Width="350px"` so the tag appears similar to the following:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="250px"
Width="350px"><Header>Header</Header><Content></Content></cc1:C1Expande
r>
```

   This will resize the C1Expander control.

5. Add the following tags just below the `</cc1:C1Expander>` tag to add a button and textbox to set the **ContentUrl** property.

```
<br /><input id="TxtUrl" type="text" value="http://www.google.com" />
<input id="Button3" type="button" value="Set ContentUrl"
onclick="LoadContentUrl()"/>
```

6. At the top of the page, add the following JavaScript just above the opening `<html>` tag:

```
<script language="javascript" type="text/javascript">
function LoadContentUrl()
{
var oExpander =
Sys.Application.findComponent("<%=C1Expander1.ClientID%>");
oExpander.set_contentUrl(document.getElementById('TxtUrl').value);
oExpander.get_headerPanel().get_contentElement().innerHTML =
document.getElementById('TxtUrl').value;
}
</script>
```

   This function will change the **C1Expander**'s **ContentUrl** property and the title in the header of the control when the button on the page is pressed.

7. Switch to Design view and note that the page now looks similar to the following:

Run the project and observe:

1. Click the **Set ContentUrl** button, the page will look similar to the following:

2. Enter a new URL in the textbox and click the **Set ContentUrl** button again.

   The Web site you have just entered will be loaded into the **C1Expander** control.

Setting the Height and Width at Run Time

You can easily change the height and width on the **C1Expander** by setting the **Height** and **Width** properties. By default the height of the control is set to **200px** and the width of the control is set to **100px**. In this topic you'll be able to resize the control at run time. For information about changing the control's height and width in Design view, Source view, and in code, instead, see Resizing the Control.

To set the control's **Height** and **Width** properties at run time, create a new AJAX-Enabled Web site project and complete the following steps:

1. Navigate to the Visual Studio Toolbox and double-click the **C1Expander** icon to add the control to your page.
2. Click on the **Source** tab to switch to Source view, and note that the body of the page looks similar to the following:

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px"
Width="100px"><Header>Header</Header><Content>Content</Content></cc1:C1
Expander>
    </div>
    </form>
</body>
```

3. Add the following tags just below the `</cc1:C1Expander>` tag to add a button and textboxes to set the **Height** and **Width** properties.

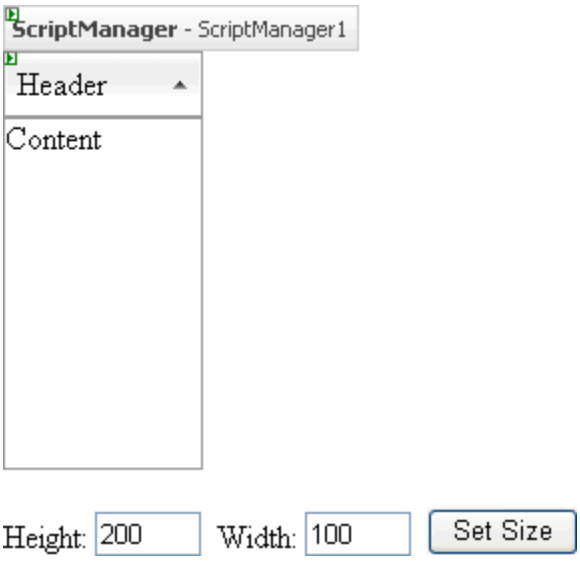
```
<br />Height: <input id="TxtHeight" type="text" size="3"
value="200"/>  Width: <input id="TxtWidth" type="text" size="3"
value="100"/>  <input id="Button3" type="button" value="Set Size"
onclick="SetSize()"/>
```

4. At the top of the page, add the following JavaScript just above the opening `<html>` tag:

```
<script language="javascript" type="text/javascript">
function SetSize()
{
var oExpander =
Sys.Application.findComponent("<%=C1Expander1.ClientID%>");
oExpander.set_height(parseInt(document.getElementById('TxtHeight').valu
e));
oExpander.set_width(parseInt(document.getElementById('TxtWidth').value)
);
}
</script>
```

   This function will change the **C1Expander** 's **Height** and **Width** properties when the button on the page is clicked.

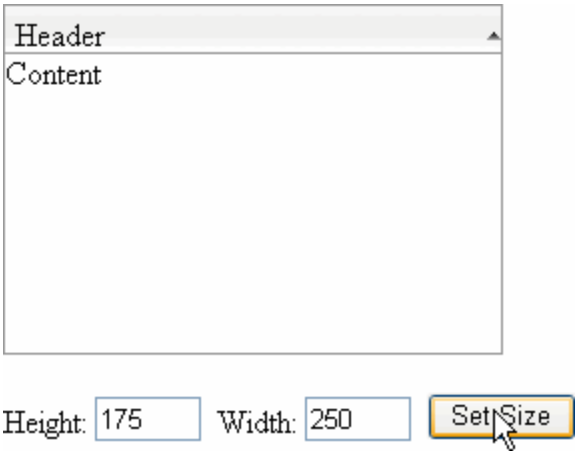
5. Switch to Design view and note that the page now looks similar to the following:



Run the project and observe:

1. Enter **175** in the **Height** textbox and **250** in the **Width** textbox and click the **Set Size** button.

   The control will resize and appear similar to the following.



2. Enter different values in the textboxes and click the **Set Size** button again. The control will resize once again.

Setting the Expand Direction at Run Time

You can change the direction that the **C1Expander** expands and collapses and the location of the **C1Expander**'s header by setting the ExpandDirection property. By default the ExpandDirection of the control is set to **Bottom** and the header appears at the top of the control (the control expands out from the header location, so expands to the bottom). For more information, see the Expand Direction topic. In this topic you'll create radio buttons that can be selected at run time to change the control's ExpandDirection.

To set the control's ExpandDirection property at run time, create a new AJAX-Enabled Web site project and complete the following steps:

1. In Design View, navigate to the Visual Studio Toolbox and double-click the **C1Expander** icon to add the control to your page.
2. Click on the **Source** tab to switch to Source view, and note that the body of the page looks similar to the following:

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="200px"
Width="100px"><Header>Header</Header><Content>Content</Content></cc1:C1
Expander>
    </div>
    </form>
</body>
```

3. In the `<cc1:C1Expander>` tag, change `Height="200px" Width="100px"` to `Height="300px" Width="300px"` so the tag appears similar to the following:

```
<cc1:C1Expander ID="C1Expander1" runat="server" HeaderSize="25px"
Height="300px"
Width="300px"><Header>Header</Header><Content>Content</Content></cc1:C1
Expander>
```

   This will resize the C1Expander control.

4. Add the following content just below the `</cc1:C1Expander>` tag to add four radio buttons that will set the ExpandDirection property.

```
<p>Select an Expand Direction:</p>
<div style="clear:none;float:left;width:100px;">
    <input type="radio" name="ExpandDirectionGroup"
id="RadioButton_ExpandDirection_Top"
onclick="C1ExpanderSampleOptionSelected(this)" />
    <label for="RadioButton_ExpandDirection_Top">Top</label>
</div>
<div style="clear:none;float:left;width:100px;">
    <input type="radio" name="ExpandDirectionGroup"
id="RadioButton_ExpandDirection_Right"
onclick="C1ExpanderSampleOptionSelected(this)" />
    <label for="RadioButton_ExpandDirection_Right">Right</label>
</div>
<div style="clear:none;float:left;width:100px;">
    <input checked="checked" type="radio" name="ExpandDirectionGroup"
id="RadioButton_ExpandDirection_Bottom"
onclick="C1ExpanderSampleOptionSelected(this)" />
    <label for="RadioButton_ExpandDirection_Bottom">Bottom</label>
</div>
<div style="clear:none;float:left;width:100px;">
    <input type="radio" name="ExpandDirectionGroup"
id="RadioButton_ExpandDirection_Left"
onclick="C1ExpanderSampleOptionSelected(this)" />
    <label for="RadioButton_ExpandDirection_Left">Left</label>
</div>
```

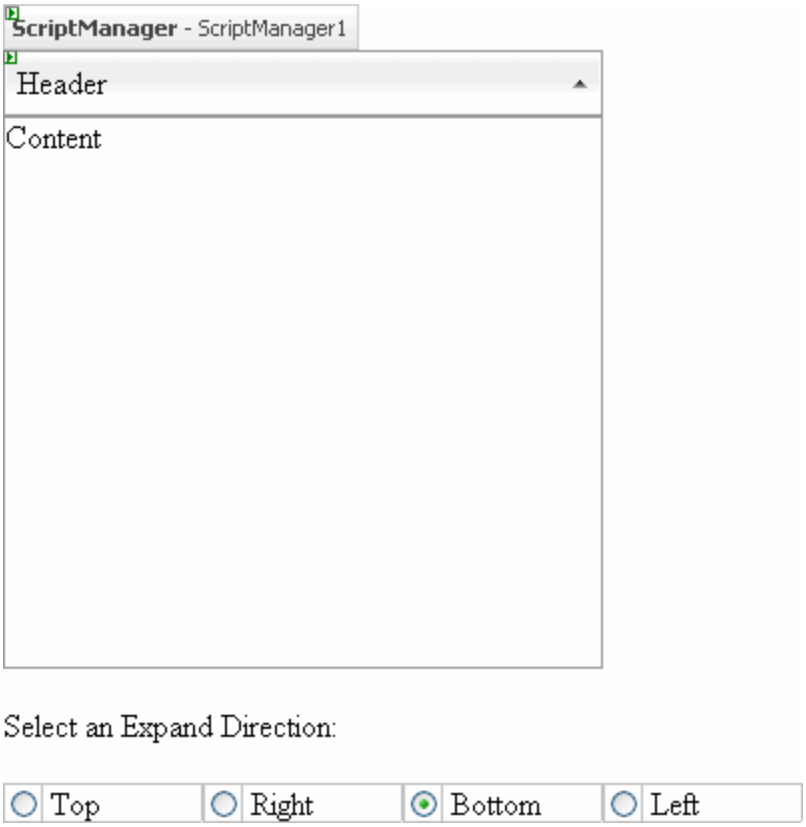
5. At the top of the page, add the following JavaScript just above the opening `<html>` tag:

```
<script language="javascript" type="text/javascript">
function C1ExpanderSampleOptionSelected(optionElem) {
```

```
    var oExpander =
Sys.Application.findComponent("<%=C1Expander1.ClientID%>");
    var s = optionElem.id;
    switch(s){
        case "RadioButton_ExpandDirection_Top":

oExpander.set_expandDirection(C1.Web.UI.Controls.C1Expander.C1ExpandDir
ection.top);
        break;
        case "RadioButton_ExpandDirection_Right":

oExpander.set_expandDirection(C1.Web.UI.Controls.C1Expander.C1ExpandDir
ection.right);
        break;
        case "RadioButton_ExpandDirection_Bottom":

oExpander.set_expandDirection(C1.Web.UI.Controls.C1Expander.C1ExpandDir
ection.bottom);
        break;
        case "RadioButton_ExpandDirection_Left":

oExpander.set_expandDirection(C1.Web.UI.Controls.C1Expander.C1ExpandDir
ection.left);
        break;
    }
}
</script>
```

This function will change the **C1Expander** 's ExpandDirection property when a radio button on the page is clicked.

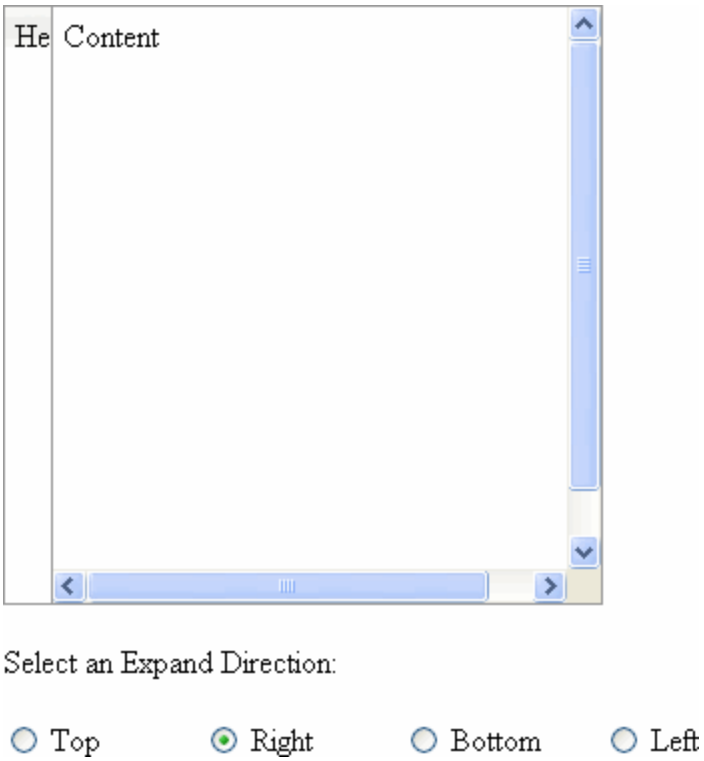
6. Switch to Design view and note that the page now looks similar to the following:



Run the project and observe:

1. Click the **Right** radio button.

   The expand direction of the control will change, and it will appear similar to the following:



2. Click the control's header to collapse the control.
3. Click a different radio button; the expand direction of the control will change again.
4. Click the control's header to view the direction in which the control expands.