ComponentOne

ToolTip for ASP.NET AJAX

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)

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

Table of Contents

ComponentOne ToolTip for ASP.NET AJAX Overview 1
What's New in ToolTip for ASP.NET AJAX 1
Revision History 1
What's New in 2009 v3 1
What's New in 2009 v2 2
Installing Studio for ASP.NET AJAX 2
Studio for ASP.NET AJAX Setup Files 2
System Requirements 2
Uninstalling Studio for ASP.NET AJAX 3
Deploying your Application in a Medium Trust Environment 3
End-User License Agreement 6
Licensing FAQs 6
What is Licensing? 6
How does Licensing Work? 7
Common Scenarios 7
Troubleshooting 9
Technical Support 11
Redistributable Files 12
About This Documentation 12
Namespaces 12
Creating an AJAX-Enabled ASP.NET Project 13
Adding the C1ToolTip Component to a Project 15

Key Features 16

C1ToolTip Quick Start 19
Step 1 of 4: Adding C1ToolTip to the Page 19
Step 2 of 4: Creating and Assigning a C1ToolTip to a Control 19
Step 3 of 4: Customizing the C1ToolTip Control 19
Step 4 of 4: Running the Application 20

ToolTip for ASP.NET AJAX Top Tips 20

C1ToolTip Design-Time Support 21
C1ToolTip Smart Tag 21
C1ToolTip Context Menu 22
C1ToolTip Designer Form 22

C1ToolTip Appearance 25
C1ToolTip Visual Styles 25
Custom Visual Styles 25
Using Templates to Embed HTML Controls in C1ToolTip 26
Animation Effects 27

Working with the Client-Side C1ToolTip 27
Client-Side Properties 27
Client-Side Methods 27
Client-Side Events 28

ToolTip for ASP.NET AJAX Samples 28

ToolTip for ASP.NET AJAX Task-Based Help 29
Adding Custom Visual Styles 29
Automatically Adding a ToolTip to Page Elements 31

Creating and Assigning a C1ToolTip to a Control 31
Creating and Assigning a C1ToolTip to Multiple Controls 32
Setting the Animation Effects for C1ToolTip 34
Using Built-in Visual Styles 34
Changing the Location of the C1ToolTip 35

ComponentOne ToolTip for ASP.NET AJAX Overview

ComponentOne ToolTip for ASP.NET AJAX allows you to load rich ToolTips on demand, keeping your Web pages small for quick load time and your users happily informed. Embed any HTML or custom controls, images, and text in a C1ToolTip. With **C1ToolTip**'s AutoTooltipify property, you can even have a tooltip automatically added to all elements on your page with a **Title** property.

Getting Started

To get started, review the following topics:

- Key Features (page 16)
- Quick Start (page 19)
- Samples (page 28)

What's New in ToolTip for ASP.NET AJAX

There were no new features for **ToolTip for ASP.NET AJAX** in the 2010 v1 release.

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

Revision History

The revision history provides recent enhancements to **ComponentOne ToolTip for ASP.NET AJAX**.

What's New in 2009 v3

You can now specify the position and animation of the C1ToolTip callout box using the following properties: CalloutSide, CalloutOffSet, CalloutAnimationDuration, and CalloutEasing. For more information on using these properties, see Changing the Location of the C1ToolTip (page 35).

Class Members

The following members have been added to **ToolTip for ASP.NET AJAX** in the 2009 v3 release:

Member	Description
CalloutSide Property	Gets or sets on which side of the element the callout will appear.
CalloutOffSet Property	Gets or sets the number of spaces to offset the callout box.
CalloutAnimationDuration Property	Determines how long, in milliseconds, the animation of the callout will last.
CalloutEasing Property	Gets or sets the easing that is applied to the callout animation.

What's New in 2009 v2

The following features were added in the 2009 v2 release:

ToolTip for ASP.NET AJAX was added to the **ComponentOne Studio Enterprise** and **ComponentOne Studio for ASP.NET AJAX** suites for 2009 v2.

Installing Studio for ASP.NET AJAX

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

Studio for ASP.NET AJAX Setup Files

The **ComponentOne Studio for ASP.NET AJAX** 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 online documentation for Studio for ASP.NET AJAX 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 7/Vista machines:

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

Windows 7/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.
Studio for ASP.NET\C1WebUi	Contains a readme.txt file and the folders that make up the Control Explorer and other samples.

Samples can be accessed from the **ComponentOne Sample Explorer**. To view samples, on your desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for ASP.NET | Control Explorer**.

System Requirements

System requirements for **ComponentOne Studio for ASP.NET AJAX** 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 or later
	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

Uninstalling Studio for ASP.NET AJAX

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

1. Open the **Control Panel** and select the **Add or Remove Programs** (**Programs and Features** in Vista/Windows 7).
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 (page 4).

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 file 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 (page 4).
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 (page 3) 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 (page 3) topic to create or modify a policy file before completing the following.

ReflectionPermission

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

To add reflection permission, complete the following:

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

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

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

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

4. Save and close the web_mediumtrust.config file.

OleDbPermission

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

To add OleDbPermission, complete the following steps:

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

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

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

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

```
</NamedPermissionSets>
```

4. Save and close the web_mediumtrust.config file.

FileIOPermission

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

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

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

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

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

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

4. Save and close the web_mediumtrust.config file.

End-User License Agreement

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

Licensing FAQs

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

What is Licensing?

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

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

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

How does Licensing Work?

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

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

When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system. (Users can also enter the serial number by clicking the **License** button on the **About Box** of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog box.)

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

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

These files are automatically added to the project.

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

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

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

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

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

Common Scenarios

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

Creating components at design time

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

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

Creating components at run time

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

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

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

Adding an instance of the component to a form, then removing that component, is just a simple way of adding a line with the component strong name to the licenses.licx file. If desired, you can do this manually using notepad or Visual Studio itself by opening the file and adding the text. When Visual Studio recreates the application resources, the component will be queried and its run-time license added to the appropriate assembly resource.

Inheriting from licensed components

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

- Add a LicenseProvider attribute to the component.

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

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

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

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

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

Using licensed components in console applications

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

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

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

Using licensed components in Visual C++ applications

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

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

1. Build the C++ project as usual. This should create an .exe file and also a licenses.licx file with licensing information in it.
2. Copy the licenses.licx file from the app directory to the target folder (Debug or Release).
3. Copy the C1Lc.exe utility and the licensed dlls to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use C1Lc.exe to compile the licenses.licx file. The command line should look like this:
   ```
          c1lc /target:MyApp.exe /complist:licenses.licx
   /i:C1.Win.C1FlexGrid.dll
   ```
5. Link the licenses into the project. To do this, go back to Visual Studio, right-click the project, select properties, and go to the Linker/Command Line option. Enter the following:
   ```
   /ASSEMBLYRESOURCE:Debug\MyApp.exe.licenses
   ```
6. Rebuild the executable to include the licensing information in the application.

Using licensed components with automated testing products

Automated testing products that load assemblies dynamically may cause them to display license 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 AJAX 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
- C1.Web.UI.4.dll
- C1.Web.UI.Controls.4.dll

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

About This Documentation

Acknowledgements

Microsoft, Windows, Windows Vista, Visual Studio, and Microsoft Expression are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Firefox is a registered trademark of the Mozilla Foundation.

Safari is a registered trademark of Apple Inc.

ComponentOne

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

Corporate Headquarters

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

http://www.componentone.com

ComponentOne Doc-To-Help

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

Namespaces

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

The general namespace for ComponentOne Web products is **C1.Web**. The following code fragment shows how to declare a **C1ToolTip** using the fully qualified name for this class:

- Visual Basic

```
Dim ToolTip As C1.Web.UI.Controls.C1ToolTip
```

- C#

```
C1.Web.UI.Controls.C1ToolTip ToolTip;
```

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 C1ToolTip = C1.Web.UI.Controls.C1ToolTip
Imports MyToolTip = MyProject.Objects.C1ToolTip

Dim wm1 As C1ToolTip
Dim wm2 As MyToolTip
```

- C#

```
using C1ToolTip = C1.Web.UI.Controls.C1ToolTip;
using MyToolTip = MyProject.Objects.C1ToolTip;

 C1ToolTip wm1;
 MyToolTip 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 Tooltip for ASP.NET AJAX 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 C1ToolTip 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://www.asp.net/ajax/downloads/archive/. 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.

Note: If you are using Visual Studio 2010, see http://www.asp.net/ajax/ for more information on creating an AJAX-Enabled ASP.NET Project.

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
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 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.

  5. A new Web Forms project is created at the root of the Web server you specified. In addition, a new Web Forms page called Default.aspx is displayed and a **ScriptManager** control is placed on the form. The **ScriptManger** is needed to enable certain features of ASP.NET AJAX such as partial-page rendering, client-script functionality of the Microsoft AJAX Library, and Web-service calls.

Adding the C1ToolTip Component to a Project

When you install **ComponentOne Studio for ASP.NET AJAX**, the **Create a ComponentOne Visual Studio 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 AJAX 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 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 AJAX controls to the Toolbox

When you install **ComponentOne Studio for ASP.NET AJAX**, the **C1ToolTip** component will appear in the Visual Studio Toolbox customization dialog box.

To manually add the Studio for ASP.NET AJAX 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 AJAX 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 AJAX, 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.C1ToolTip. 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 AJAX Controls to the Form

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

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

Adding a Reference to the Assembly

To add a reference to the C1.Web. UI.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 AJAX** 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 (page 12) for more information.

Key Features

The following are some of the main features of C1ToolTip that you may find useful:

- **Automatically add C1ToolTips to your page**
  Use the AutoTooltipify property to automatically add a C1ToolTip to all items on your page with a **Title** property, giving your page a clean, consistent look. For more information, see Automatically Adding a ToolTip to Page Elements (page 31).
- **Embed HTML or custom controls within C1ToolTip**
  ComponentOne ToolTip for ASP.NET AJAX supports templates so you can easily embed any HTML controls or even your own custom controls inside of C1ToolTip. See Using Templates to Embed HTML Controls in C1ToolTip (page 26) for more information.

- **Dynamically load rich C1ToolTip content on demand**
  Keep page sizes small and manageable using **C1ToolTip's** load on demand feature. C1ToolTip provides an **C1ToolTip.OnAjaxUpdate** event, which triggers an AJAX call to the server when the user hovers over a particular tooltip element on the client. The event handler receives the element's ID, allowing the rich data of the C1ToolTip to be dynamically loaded. See the Ajax Update sample (page 28) installed with the product for a detailed example on using the load on demand feature.

- **Built-in visual styles**
  Choose from five built-in visual styles to quickly change the C1ToolTip control's appearance. Visual styles include: **ArcticFox**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, and **Vista**. See Using Built-in Visual Styles (page 34) for additional information.

- **Customize the C1ToolTip control with visual styles and CSS styling**
  Easily customize the C1ToolTip control by pointing to your own visual styles using the VisualStyle property.
  ToolTip for ASP.NET AJAX also includes CSS-supported styling so that you can use cascading style sheets to easily style the C1ToolTip control to match the design of your current Web site.
  See Adding Custom Visual Styles (page 29) for more information.

C1ToolTip Quick Start

This quick start will lead you through the creation of a Web form that uses the C1ToolTip control. This quick start was created using Visual Studio 2008. Note that the steps may be slightly different if you are using a previous version of Visual Studio.

Step 1 of 4: Adding C1ToolTip to the Page

In this topic, you will add a C1ToolTip control to the page.

1. Begin by creating an ASP.NET AJAX-Enabled Web Site (page 13). Note that if using Visual Studio 2008, you must add a **ScriptManager** control to the form. If using Visual Studio 2005, the **ScriptManager** control is automatically added to the form.
2. Click **View | Designer** to make sure you are in **Design** view.
3. Navigate to the Visual Studio Toolbox and double-click the C1ToolTip control to add it to your page.
4. Add an HTML **Input Button** control to your form as well. This is the control that will have a ToolTip. Notice in the Visual Studio Properties window (**View | Properties Window**) that the **Button** control has an **Id** of **Button1**. This ID will be used to assign the ToolTip to the control.

Note: If the control is not appearing correctly, make sure you have a reference to the C1.Web.UI.Design.2.dll installed with the product in your project.

In the next step you will create a C1ToolTip and assign it to the **Button** control.

Step 2 of 4: Creating and Assigning a C1ToolTip to a Control

In this step you will create a simple ToolTip and assign it to the **Button** control.

1. Click the **C1ToolTip** smart tag (<) and select **Edit TargetControls** in the **C1ToolTip Tasks** menu. The **C1ToolTip Designer Form** opens.
2. Select **C1ToolTip1** on the **Edit** tab.
3. In the properties pane, enter "This is the text for my C1ToolTip." next to the Text property.
4. Click the **Add Child Item** button on the toolbar and select **ToolTipTargetControl**.
5. Select **C1ToolTipTargetControl** in the tree and in the properties pane, enter **Button1** next to the **C1ToolTip.TargetControlID** property.
6. Click **OK** to close the designer.

Now that you've created and assigned the ToolTip to a control, you can determine how the ToolTip appears.

Step 3 of 4: Customizing the C1ToolTip Control

In this step you will determine when the ToolTip appears and the animation used when it is shown.

1. Click the C1ToolTip smart tag (<) to open the **C1ToolTip Tasks** menu.
2. Click the drop-down arrow next to the **VisualStyle** property and select **Vista** from the list.
3. In the **C1ToolTip Tasks** menu, click **Edit TargetControls** to open the **C1ToolTip Designer Form**.

4. Select **C1ToolTip1** on the **Edit** tab.
5. In the properties pane, click the drop-down arrow next to the ShowAnimation property and select **FadeIn**.
6. Click the drop-down arrow next to the ShowEvent property and select **OnClick**.
7. Click **OK** to close the designer.

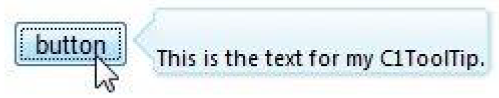
Now that you've customized the C1ToolTip control, let's run the application.

Step 4 of 4: Running the Application

In the previous steps of the quick start you created a simple application with a C1ToolTip control assigned to a **Button** control, and you customized the ToolTip's appearance and behavior. In this step, you'll run the application and explore some of the run-time interactions possible with the C1ToolTip control.

Complete the following steps:

1. In Visual Studio, select **Debug | Start Debugging** or press **F5** to run the application.
2. Click the button and notice how the ToolTip fades into view.



Congratulations! You've completed the C1ToolTip quick start. If you'd like to continue exploring **ToolTip for ASP.NET AJAX**, see the ToolTip for ASP.NET AJAX Samples (page 28) and take a look at the ToolTip for ASP.NET AJAX Task-Based Help (page 29) topics.

ToolTip for ASP.NET AJAX Top Tips

The following tips were compiled from frequently asked user questions posted in the Studio for ASP.NET AJAX forum.

Tip 1: Use the AutoTooltipify or ToolTipZoneID property to show a tooltip for the elements with a Title attribute.

You can replace a large number of common tooltips with C1Tooltips for any element with a **Title** attribute just by setting one property. See the Automatically Adding a ToolTip to Page Elements (page 31) topic for steps on how to do this.

Set the AutoTooltipify property to **True**, and the specified C1ToolTip will be shown for all elements with a "Title" attribute.

Set the ToolTipZoneID property to a specific element, and the element's children that have a **Title** attribute will also have a C1ToolTip.

Tip 2: Use the CalloutSide and CalloutOffset properties to customize the callout position.

The CalloutSide property is used to determine on which side of the C1ToolTip the callout will appear. The CalloutOffset property is used to specify the offset value from the left or top edge. If the CalloutOffset property is set, the callout will move to the new position using animation.

C1ToolTip Design-Time Support

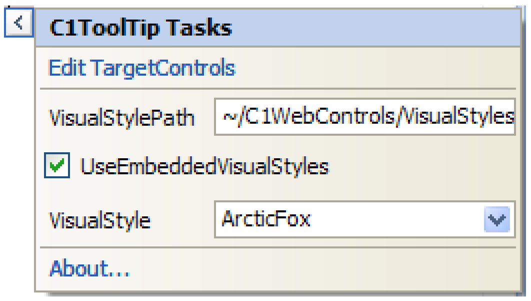
The following sections describe how to use **C1ToolTip**'s design-time environment to configure the C1ToolTip control.

C1ToolTip Smart Tag

The C1ToolTip control includes a smart tag (<) in Visual Studio. A smart tag represents a shortcut tasks menu that provides the most commonly used properties in C1ToolTip.

The C1ToolTip control provides quick and easy access to common properties through its smart tag.

To access the **C1ToolTip Tasks** menu, click the smart tag in the upper-right corner of the C1ToolTip control.



VisualStylePath ~/C1WebControls/VisualStyles

☑ UseEmbeddedVisualStyles

VisualStyle ArcticFox

The **C1ToolTip Tasks** menu operates as follows:

- **Edit TargetControls**

  Clicking **Edit TargetControls** opens the **C1ToolTip Designer Form** where ToolTip text and behavior can be defined and ToolTips can be assigned to specific controls. You can also preview **C1ToolTips** here.

- **VisualStylePath**

  The VisualStylePath property specifies the path to the folder containing built-in visual styles by default. If you want to use a custom style, you can change the VisualStylePath here. See Adding Custom Visual Styles (page 29) for more information on using custom styles.

- **UseEmbeddedVisualStyles**

  The UseEmbeddedVisualStyles check box is checked by default so that the built-in visual styles can be used. If you want to use a custom visual style, uncheck this check box. See Adding Custom Visual Styles (page 29) for more information on using custom styles.

- **VisualStyle**

  Clicking the **VisualStyle** drop-down arrow allows you to select from various built-in visual styles. See Using Built-in Visual Styles (page 34) for more information.

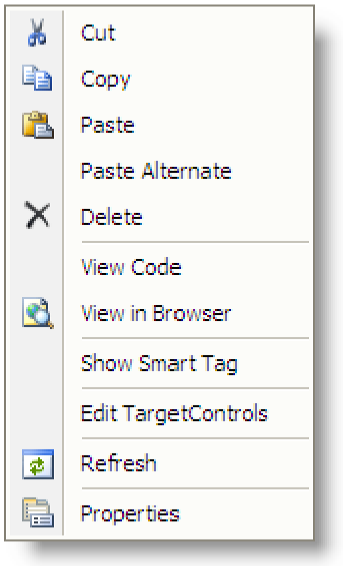
- **About**

  Displays the **About ComponentOne ToolTip** dialog box, which is helpful in finding the version number of the product and online resources such as how to purchase a license, how to contact ComponentOne, or view ComponentOne product forums.

C1ToolTip Context Menu

C1ToolTip has additional commands available on the context menu that Visual Studio provides for all .NET and ASP.NET controls.

Right-click anywhere on the C1ToolTip control to display the context menu:



The context menu commands operate as follows:

- **Edit TargetControls**

  Clicking **Edit TargetControls** opens the **C1ToolTip Designer Form** where ToolTip text and behavior can be defined and ToolTips can be assigned to specific controls. You can also preview **C1ToolTips** here.

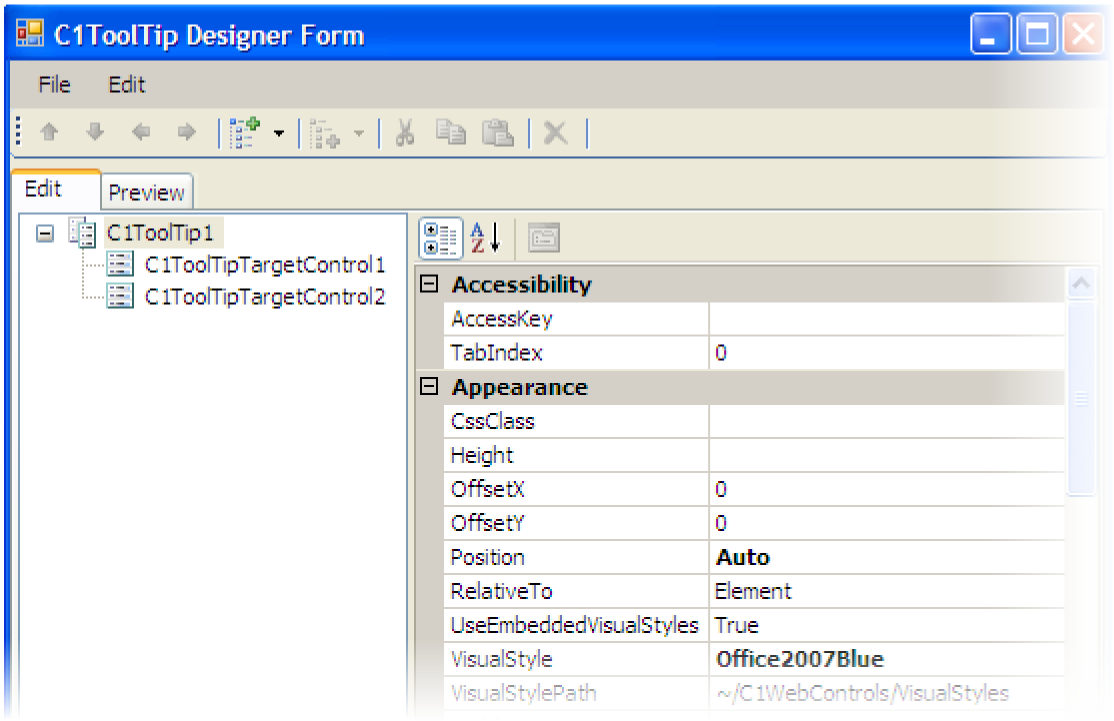
C1ToolTip Designer Form

The **C1ToolTip Designer Form** is a designer used to define, format, preview, and assign **C1ToolTips** to specific controls. The **C1ToolTip Designer Form** is similar to the Properties window as it allows programmers to modify the control visually. However, it allows you set the text for the ToolTip, set the ToolTip properties, assign the ToolTip to a control, and preview the ToolTip appearance, all within the form.

In this topic you will become familiar with the **C1ToolTip Designer Form's** design interface so you can use the commands within it to edit C1ToolTip with minimal effort and time.

To open the **C1ToolTip Designer Form**, click **Edit TargetControls** from the **C1ToolTip Tasks** menu.

The **C1ToolTip Designer Form** contains a menu, toolbar, **Edit** tab, **Preview** tab, and properties pane.



Accessibility	
AccessKey	
TabIndex	0
Appearance	
CssClass	
Height	
OffsetX	0
OffsetY	0
Position	**Auto**
RelativeTo	Element
UseEmbeddedVisualStyles	True
VisualStyle	**Office2007Blue**
VisualStylePath	~/C1WebControls/VisualStyles

Edit Tab

Click the **Edit** tab and select the C1ToolTip want to edit. You can assign a C1ToolTip to a specific control here, too.

Preview Tab

Click the **Preview** tab to view how the C1ToolTip will look at run time.

Properties Pane

The **C1ToolTip Designer Form** properties pane is almost identical to the Visual Studio Properties window. Simply select a C1ToolTip or C1ToolTipTargetControl and set the desired properties here.

Command Buttons

The command buttons are summarized in the following table:

Button	Description
OK	Clicking **OK** applies the new settings to the C1ToolTip control.
Cancel	Clicking **Cancel** closes the **C1ToolTip Designer Form**, cancelling the new settings and applying the default settings to the C1ToolTip control.

C1ToolTip Designer Form Menu

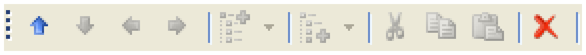
The **C1ToolTip Designer Form** menu contains the following menu items and subitems:

Menu Item	Submenu Item	Description
File	Load from XML	Loads the formatting for a C1ToolTip control from an .xml file.
	Save as XML	Saves the current formatting of the C1ToolTip control to an .xml

		file.
	Exit	Closes the **C1ToolTip Designer Form**.
Edit	Insert Item	Inserts a new C1ToolTipTargetControl item at the specified place in the list of C1ToolTipTargetControl items.
	Add Child	Adds a new C1ToolTipTargetControl item as a child of the C1ToolTip or of another C1ToolTipTargetControl item.
	Cut	Cuts the selected C1ToolTipTargetControl item to be moved in the list of **C1ToolTipTargetControls**.
	Copy	Copies the selected C1ToolTipTargetControl item.
	Paste	Pastes a C1ToolTipTargetControl item at the specified location in the list of C1ToolTipTargetControls.
	Delete	Removes the selected C1ToolTipTargetControl item.
	Rename	Allows you to change the name of the C1ToolTipTargetControl item.

C1ToolTip Designer Form Toolbar

The toolbar for the **C1ToolTip Designer Form** appears like the following:



The table below describes each button in the toolbar:

Button	Name	Description
⬆	Move Item Up	Moves the selected C1ToolTipTargetControl item up in the list of toolbar items.
⬇	Move Item Down	Moves the selected C1ToolTipTargetControl item down in the list.
⬅	Move Item Left	Moves the selected C1ToolTipTargetControl item to the left in the hierarchy.
➡	Move Item Right	Moves the selected C1ToolTipTargetControl item to the right in the hierarchy.
☰	Add Child Item	Inserts a C1ToolTipTargetControl item as a child of the C1ToolTip control or of another C1ToolTipTargetControl item.
☰	Insert Item	Inserts a toolbar item at the specified location in the list of C1ToolTipTargetControl items.
✂	Cut	Cuts the selected C1ToolTipTargetControl items to be moved in the list of **C1ToolTipTargetControls**.
⧉	Copy	Copies the selected C1ToolTipTargetControl item.
📋	Paste	Pastes a toolbar item at the specified location in the list of C1ToolTipTargetControl items.
✖	Delete	Removes the selected C1ToolTipTargetControl item.

C1ToolTip Appearance

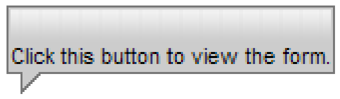
The following sections describe how to change the appearance of C1ToolTip using built-in visual styles as well as how to use your own custom styles.

C1ToolTip Visual Styles

ComponentOne ToolTip for ASP.NET AJAX provides five built-in visual styles, allowing you to automatically format the C1ToolTip control. The styles include the following: **ArcticFox**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, and **Vista**. For more information, see Using Built-in Visual Styles (page 34).

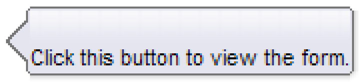
ArcticFox Style

The following image displays the **ArcticFox** style. This is the default format of the C1ToolTip control:



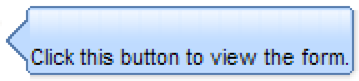
Office2007Black Style

The following image displays the **Office2007Black** style:



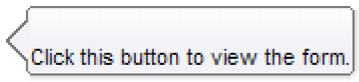
Office2007Blue Style

The following image displays the **Office2007Blue** style:



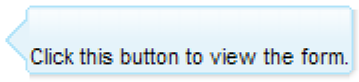
Office2007Silver Style

The following image displays the **Office2007Silver** style:



Vista Style

The following image displays the **Vista** style:

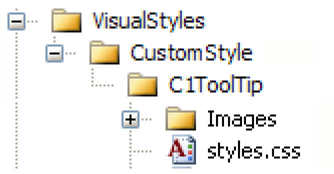


Custom Visual Styles

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

Tip: The easiest way to create a custom visual styles is by modifying one of the control's pre-existing visual styles. You can find the .css sheets and images for the **ToolTip for ASP.NET AJAX** visual styles within the installation directory at C:\Program Files\ComponentOne\Studio for ASP.NET\C1WebUI\VisualStyles. If you use one of the existing .css sheets, you must change any instances of the default file name, for example **_ArcticFox** to your new style name, for example **_CustomStyle**, in order for your custom style to work.

Before adding your .css file and images, you will have to create a hierarchy of folders, the last of which will hold your files. On the top-level of your project, create a folder named "VisualStyles". Underneath the **VisualStyles** folder, create a sub-folder bearing the theme name (such as "CustomStyle), and then, beneath that, create a sub-folder named "C1ToolTip". The image folder and .css file should be placed underneath the **C1ToolTip** folder. The result will resemble the following:



This structure of these folders is very important; **ToolTip for ASP.NET AJAX** will always look for the **~/VisualStyles/StyleName/C1ToolTip/styles.css** path, as it is the default visual style path for the control.

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

For more information on customizing the appearance of **ToolTip for ASP.NET AJAX** controls, see Adding Custom Visual Styles (page 29).

Using Templates to Embed HTML Controls in C1ToolTip

Templates can be used to embed other HTML controls in the C1ToolTip control. The templates are useful for making the C1ToolTip more coherent to your application and for displaying additional information, such as images or text.

To use a template, complete the following steps. In this example, we'll add an empty **TextBox** control to the C1ToolTip.

1. Click the **Source** tab to view the source markup for your project.
2. Add the <Template> tag within the **C1ToolTip** tags so it looks like the following:

```
<cc1:C1ToolTip ID="C1ToolTip1" runat="server" VisualStyle="ArcticFox"
AutoClose="false" VisualStylePath="~/VisualStyles">
      <Template>
          <asp:TextBox runat="server" ID="abc"></asp:TextBox>
      </Template>
      <TargetControls >
          <cc1:C1ToolTipTargetControl TargetControlID="Button1"/>
          <cc1:C1ToolTipTargetControl TargetControlID="Button2"/>
      </TargetControls>
</cc1:C1ToolTip>
```

Animation Effects

C1ToolTip includes twenty-six different built-in animation options that allow you to customize how the C1ToolTip appears or disappears on a Web page. You can set the animation using the ShowAnimation and HideAnimation properties. By default, these properties are set to **None**, so there is no animation effect. You can change the transition for the animation using the ShowEasing and HideEasing properties. By default, these properties are set to **EaseLiner**, so the control appears and hides with a smooth linear transition effect.

The Setting the Animation Effects for C1ToolTip (page 34) topic shows you how to set the ShowAnimation, ShowEasing and ShowDuration properties.

Sample Project Available

For a demonstration of each transition effect possible in the C1ToolTip control, see the **C1ToolTip Animation** page of the **ControlExplorer** sample.

Working with the Client-Side C1ToolTip

When a C1ToolTip control is rendered, an instance of the client-side ToolTip will be created automatically. This means that you can enjoy the convenience of accessing properties and methods of the C1ToolTip control without having to postback to the server.

For example, suppose a C1ToolTip control with the name, **C1ToolTip1** is hosted on the Web page. When the page is rendered, a corresponding client object with the name **C1ToolTip1_Client** will be created.

You can get the reference for a C1ToolTip using the following client-side code:

```
var controlObj = Sys.Application.findComponent("<%=C1ToolTip1.ClientID%>");
```

Using **C1ToolTip**'s client-side code, you can implement many features in your Web page without the need to send information to the Web server which takes time. Thus, using **C1ToolTip**'s client-side object model can increase the efficiency of your Web site.

The following topics describe the available client-side properties, methods, and events.

Client-Side Properties

The following conventions are used when accessing the 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).

Client-Side Methods

C1ToolTip includes a rich client-side object model in which several methods can be used on the client side. For information about these client-side methods and what properties can be set on the client side, see the C1ToolTip reference section of this documentation.

Client-Side Events

C1ToolTip includes several client-side events that allow you to manipulate the C1ToolTip control, such as hiding or showing the ToolTip or performing an action before showing or hiding the ToolTip.

You can access these client-side events from the Properties window. To create a new client-side event using the Properties window, select the drop-down arrow next to a client-side event and select **Add new client side handler**.

Once you've added the client-side handler, Visual Studio will add a script to the Source view. That script will resemble the following:

```
<script type="text/javascript" id="ComponentOneClientScript1">
function C1ToolTip1_OnClientBeforeHide(args){
  //
  // Put your code here.
  //
};
</script>
```

Each of the client-side events requires two parameters: the **ID** that identifies the sender C1ToolTip, in this example, **C1ToolTip1**, and an **eventArgs** that contains the data of the event.

You can use the sever-side properties, listed in the **Client Side Event** table, to specify the name of the JavaScript function that will respond to a particular client-side event.

The following table lists the events that you can use in your client scripts. These properties are defined on the server side, but the actual events or the name you declare for each JavaScript function are defined on the client side.

Client Side Event table

Event Server-Side Property Name	Description
OnClientBeforeHide	Fires on client side before the ToolTip disappears.
OnClientBeforeShow	Fires on client side before the ToolTip appears.
OnClientHide	Fires on client side when the ToolTip is hidden.
OnClientShow	Fires on client side when the ToolTip is shown.

ToolTip for ASP.NET AJAX 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.

Samples can be accessed from the **ComponentOne Sample Explorer**. To view samples, on your desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for ASP.NET | Control Explorer**.

The following pages within the ControlExplorer sample installed with **ComponentOne Studio for ASP.NET AJAX** detail the C1ToolTip control's functionality:

C# Samples

Sample	Description

VisualStyles	This sample displays the built-in C1ToolTip control visual styles including: **ArcticFox**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, and **Vista**.
Animation	This sample shows how you can set the ShowAnimation, ShowEasing, ShowDuration, ShowDelay, HideAnimation, HideEasing, HideDuration, and HideDelay properties.
Ajax Update	This sample demonstrates the AjaxUpdate effects that can be applied to the C1ToolTip control.
Client Side	This sample shows the client side event effects for the C1ToolTip control.
Scrolling	This sample shows how the C1ToolTip can be scrolled: when the user clicks the scroll button, when the user hovers over the scroll button, when the user hovers over the ToolTip and moves the mouse, and with scrollbars.
Server Api	This sample demonstrates the various server APIs that can be applied to the C1ToolTip control.

ToolTip for ASP.NET AJAX Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET. By following the steps outlined in the Help, you will be able to create projects demonstrating a variety of **ToolTip for ASP.NET AJAX** features and get a good sense of what **ToolTip for ASP.NET AJAX** can do.

Each task-based help topic also assumes that you have created a new AJAX-enabled ASP.NET project. **ComponentOne ToolTip for ASP.NET AJAX** 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 C1ToolTip 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. For additional information on this topic, see Creating an AJAX-Enabled ASP.NET Project (page 13).

Adding Custom Visual Styles

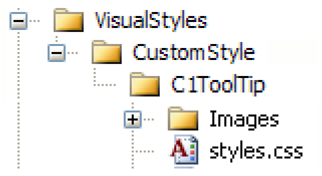
You can use the VisualStyle, **VisualStylePath**, and **UseEmbeddedVisualStyles** properties to create a custom visual style for your C1ToolTip. For more information on custom visual styles, see Custom Visual Styles (page 25).

To add a custom visual style, best practice is to copy one of the existing visual styles and customize it. In this example we will use the **C1ToolTip ArcticFox** style.

Adding Custom Visual Styles in Design View

To add a custom visual style, complete the following steps:

1. Copy the theme folder C:\Program Files\ComponentOne\Studio for ASP.NET\C1WebUI\VisualStyles\ArcticFox\C1ToolTip to a new folder in your Visual Studio project so the folder structure is: ~\VisualStyles\CustomStyle\C1ToolTip.



2. Open the styles.css file in the ~/VisualStyles/CustomStyle/C1ToolTip folder and replace any instance of "ArcticFox" with "CustomStyle". You can modify the CSS definition to customize the appearance.
3. Locate the first instance of **.C1ToolTip_CustomStyle** and add a font-size and font-family attribute so it looks like the following:
```
.C1ToolTip_CustomStyle
{
    clear: none;
    font-size: small;
    font-family: Arial, Verdana;
}
```
4. Save and close the styles.css file.
5. Add a button control and C1ToolTip control to your page.
6. Click the **C1ToolTip** smart tag () and select **Edit TargetControls** in the **C1ToolTip Tasks** menu. The **C1ToolTip Designer Form** opens.
7. Select **C1ToolTip1** on the **Edit** tab.
8. In the properties pane, next to the Text property, enter "This is my tooltip."
9. Click the **Add Child Item** button on the toolbar and select the **C1ToolTipTargetControl1** item in the tree.
10. In the properties pane, enter **Button1** next to the **C1ToolTip.TargetControlID** property.
11. Click **OK** to close the **C1ToolTip Designer Form**.
12. Select the C1ToolTip control on your form and click the smart tag to open the **C1ToolTip Tasks** menu again.
    a. Uncheck the **UseEmbeddedVisualStyles** property to set it to **False**.
    b. Make sure the **VisualStylePath** property is set to **~/VisualStyles**.
    c. Select **CustomVisualStyle (external)** from the **VisualStyle** property drop-down list.
13. Press F5 to run your project, move the mouse over the button, and notice the C1ToolTip text is small and in Arial font.

Adding Custom Visual Styles in Source View

To add a custom visual style in Source view, complete the following steps:

1. Complete steps 1 through 4 under **Adding Custom Visual Styles in Design View**.
2. Enter Source view and enter `VisualStyle="CustomStyle"`, `VisualStylePath="~/VisualStyles"`, and `UseEmbeddedVisualStyles="False"` into the `<cc1:C1ToolTip>` tag. Your XHTML will resemble the following:
```
<cc1:C1ToolTip ID="C1ToolTip1" runat="server" Height="22px"
VisualStyle="CustomStyle" VisualStylePath="~/VisualStyles"
UseEmbeddedVisualStyles="False" Width="155px" />
```

Adding Custom Visual Styles in Code

To add a custom visual style, complete the following steps:

1. Complete steps 1 through 4 under **Adding Custom Visual Styles in Design View**.
2. Double-click the Web project to place a **Page_Load** event in the code editor.
3. Set the **UseEmbeddedVisualStyles** property to **False** by adding the following code to the **Page_Load** event:
    - Visual Basic
```
C1ToolTip1.UseEmbeddedVisualStyles = False
```

- C#
```
C1ToolTip1.UseEmbeddedVisualStyles = false;
```

4. Change the **VisualStylePath** property:

   - Visual Basic
   ```
   C1ToolTip1.VisualStylePath = "~/VisualStyles"
   ```

   - C#
   ```
   C1ToolTip1.VisualStylePath = "~/VisualStyles";
   ```

5. Select the custom visual style:

   - Visual Basic
   ```
   C1ToolTip1.VisualStyle = "CustomStyle"
   ```

   - C#
   ```
   C1ToolTip1.VisualStyle = "CustomStyle";
   ```

6. Run the program and observe that the C1ToolTip control has adopted your custom visual style.

Automatically Adding a ToolTip to Page Elements

You can automatically add a C1ToolTip to all elements on your page that have a **Title** property using the AutoTooltipify property. This topic assumes you already have a C1ToolTip created.

1. Select the C1ToolTip control on your page.
2. Click the smart tag to open the **C1ToolTip Tasks** menu and click **Edit TargetControls**. The **C1ToolTip Designer Form** opens.
3. On the **Edit** tab, select **C1ToolTip1**.
4. In the properties pane, enter the text for your ToolTip next to the Text property.
5. Click the drop-down arrow next to the AutoTooltipify property and select **True**.
6. Click **OK** to close the designer. When you run your project, the ToolTip appears for each of the elements on the page that have a **Title** property.

Creating and Assigning a C1ToolTip to a Control

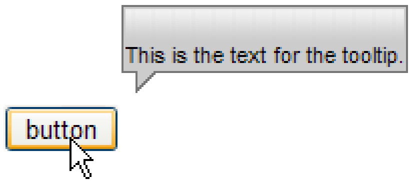
Creating a simple ToolTip and specifying the control it belongs to is a quick and easy process. Follow these steps:

In Design View

This example shows a C1ToolTip for one button control in **Design** view.

1. Add a button control and C1ToolTip control to your page.
2. Click the **C1ToolTip** smart tag (<) and select **Edit TargetControls** in the **C1ToolTip Tasks** menu. The **C1ToolTip Designer Form** opens.
3. Select **C1ToolTip1** on the **Edit** tab.
4. In the properties pane, enter the text for your ToolTip next to the Text property.
5. Click the drop-down arrow next to the Position property and select where you want the ToolTip to appear in relation to the control that will show it.
6. Click the **Add Child Item** button on the toolbar and select the **C1ToolTipTargetControl** item in the tree.
7. In the properties pane, enter the ID for the control that you want to show the ToolTip next to the **C1ToolTip.TargetControlID** property, or **Button1** in this case.

8. Click **OK** to close the designer. When you run your project, the ToolTip text will appear in the specified position.



In Source View

This example shows a C1ToolTip for one button control in **Source** view.

1. Create a C1ToolTip by entering the following markup within the **<form>** tags of the page, after the **ScriptManager** tags. Here is an example:
```
<cc:C1ToolTip ID="C1ToolTip1" runat="server" HideDelay="0" Position="Auto"
ShowAnimation="FadeIn" ShowDelay="0" Text="This is the text for my
C1ToolTip." VisualStyle="Vista">
</cc1:C1ToolTip>
```
2. Add a button control to your page by adding the following markup under the C1ToolTip.
```
<input id="Button1" type="button" value="button" />
```
3. Add the **TargetControl** using markup like the following, which should appear within the **C1ToolTip** tags from step 1.
```
<cc1:C1ToolTip ID="C1ToolTip1" runat="server" HideDelay="0"
Position="Auto"
        ShowAnimation="FadeIn" ShowDelay="0"
        Text="This is the text for my C1ToolTip." VisualStyle="Vista">
        <TargetControls>
            <cc1:C1ToolTipTargetControl IsAjaxUpdate="False"
TargetControlID="Button1" />
        </TargetControls>
    </cc1:C1ToolTip>
```
4. Press **F5** to run the project and notice the C1ToolTip appears when you mouse over the button.

Creating and Assigning a C1ToolTip to Multiple Controls

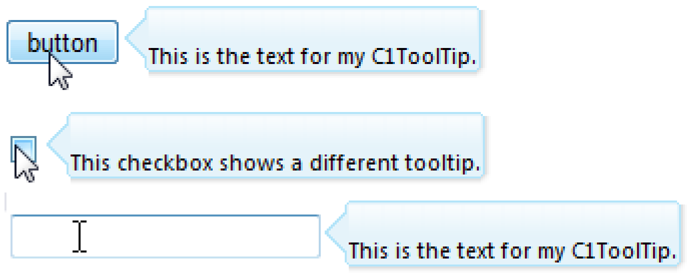
You can assign one C1ToolTip to multiple control. You can also show different content for each control's ToolTip. In this example, we will add three controls, a button, a check box, and a text box, to the C1ToolTipTargetControlCollection. The button and text box will show the same ToolTip, and the check box will have a different ToolTip.

In Design View

Follow these steps to add the controls and C1ToolTip:

1. Add a **Button**, **CheckBox**, **TexBox**, and C1ToolTip control to your page.
2. Click the **C1ToolTip** smart tag (<) and select **Edit TargetControls** in the **C1ToolTip Tasks** menu. The **C1ToolTip Designer Form** opens.
3. Select **C1ToolTip1** on the **Edit** tab.
4. In the properties pane, enter the text for your ToolTip next to the Text property.
5. Click the **Add Child Item** button on the toolbar and select the **C1ToolTipTargetControl** item in the tree.
6. In the properties pane, enter **Button1** next to the **C1ToolTip.TargetControlID** property.

7. Add two more child items in the same way, making sure **C1ToolTip1** is selected when you click the **Add Child Item** button.
8. Enter **CheckBox1** next to the **C1ToolTip.TargetControlID** property for the check box.
9. With **CheckBox1** still selected, enter some text next to the Value property. This is the ToolTip content that will appear for this control.
10. Enter **Text1** next to the **C1ToolTip.TargetControlID** property for the textbox.
11. Click **OK** to close the designer.
12. Press **F5** to run the project and then mouse over each control. Notice the **C1ToolTip** for the button and textbox is the same. The text box for the check box is different.


button
This is the text for my C1ToolTip.
This checkbox shows a different tooltip.
This is the text for my C1ToolTip.

In Source View

Follow these steps to add the controls and C1ToolTip:

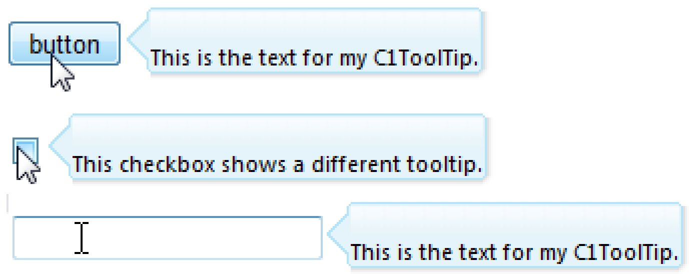
1. Create a C1ToolTip by entering the following markup within the **<form>** tags of the page, after the **ScriptManager** tags. Here is an example:
```
<cc1:C1ToolTip ID="C1ToolTip1" runat="server" HideDelay="0"
Position="Auto" ShowAnimation="FadeIn" ShowDelay="0" Text="This is the
text for my C1ToolTip." VisualStyle="Vista">
</cc1:C1ToolTip>
```
2. Add a **Button**, **CheckBox**, and **TextBox** control to your page by adding the following markup within the **<body>** tags after the **C1ToolTip** tags:
```
<p>
         <input id="Button1" type="button" value="button" /></p>
    <p>
    </form>
    <p>
        <input id="CheckBox1" type="checkbox" /></p>
    <p>
        </p>

    <p>
        <input id="Text1" type="text" /></p>
```
3. Add the **TargetControls** using markup like the following, which should appear within the **C1ToolTip** tags from step 1.
```
<cc1:C1ToolTip ID="C1ToolTip1" runat="server" HideDelay="0"
Position="Auto"
        ShowAnimation="FadeIn" ShowDelay="0"
        Text="This is the text for my C1ToolTip." VisualStyle="Vista">
        <TargetControls>
            <cc1:C1ToolTipTargetControl IsAjaxUpdate="False"
TargetControlID="Button1" />
```

```
            <cc1:C1ToolTipTargetControl IsAjaxUpdate="False"
TargetControlID="CheckBox1"
                Value="This checkbox shows a different tooltip." />
            <cc1:C1ToolTipTargetControl IsAjaxUpdate="False"
TargetControlID="Text1" />
        </TargetControls>
    </cc1:C1ToolTip>
```

Notice the **Value** is set for the **CheckBox** so that it will show a different ToolTip.

4. Press **F5** to run the project and then mouse over each control. Notice the C1ToolTip for the button and textbox is the same. The text box for the check box is different.


button
This is the text for my C1ToolTip.
This checkbox shows a different tooltip.
This is the text for my C1ToolTip.

Setting the Animation Effects for C1ToolTip

You can add animation effects to the C1ToolTip control by setting any of the animation properties: ShowAnimation, HideAnimation, ShowEasing, HideEasing, ShowDuration, and HideDuration. In this example, we'll set the effects for animation when the ToolTip appears. This topic assumes you have a ToolTip assigned to at least one element.

1. Click the C1ToolTip smart tag to open the **C1ToolTip Tasks** menu.
2. Select **Edit TargetControls** and choose **C1ToolTip1**.
3. Set the ShowAnimation property to **UnFold**.
4. Set the ShowEasing property to **EaseInBounce**.
5. Set the ShowDuration property to "1500". This will make the duration effect a little longer so you can see it.
6. Click **OK** to close the **C1ToolTip Designer Form**. When you run your project and hover over the element with a ToolTip, the C1ToolTip bounces and unfolds as it appears.

Using Built-in Visual Styles

You can format C1ToolTip with one of five built-in visual styles. Note that you can also use your own style; see Adding Custom Visual Styles (page 29) for more information.

Changing the Visual Style Using the Smart Tag

You can change the style of C1ToolTip at design time using the **C1ToolTip Tasks** menu.

1. Click the C1ToolTip smart tag to open the **C1ToolTip Tasks** menu.
2. Select **Edit TargetControls**. The **C1ToolTip Designer Form** appears.
3. Click drop-down arrow next to **VisualStyle**.

4. Select one of the built-in styles listed. The style is applied to the C1ToolTip control.

Changing the Visual Style Programmatically

To change the style of C1ToolTip programmatically, use the following code. In this example, "Vista" is used, but you can replace it with any of the built-in styles (**ArcticFox**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, or **Vista**) or use your own style.

- Visual Basic
  ```
  C1ToolTip1.VisualStyle = "Vista"
  ```
- C#
  ```
  C1ToolTip1.VisualStyle = "Vista";
  ```

Changing the Location of the C1ToolTip

You can specify the position and animation of the C1ToolTip callout box using the following properties: CalloutSide, CalloutOffSet, CalloutAnimationDuration, and CalloutEasing. In this example, a ToolTip will appear when a user mouses over a button.

In Design View

To set the **Callout** properties in Design view, click **View | Designer** in Visual Studio and follow these steps:

1. Add an HTML **Input (Button)** control and C1ToolTip control to your page.
2. Right-click the C1ToolTip control and select Properties.
3. In the Visual Studio Properties window, click the drop-down arrow next to the CloseBehavior property and select none.
4. Enter the text for your tooltip next to the Text property.
5. Click the drop-down arrow next to the CalloutSide property and select **Top**, **Right**, **Bottom**, or **Left**, depending on where you want the callout box for the ToolTip to appear. **Auto** is the default option.
6. Set the CalloutOffset property to the number, in pixels, you would like to offset the ToolTip relative to the top or left edge of the ToolTip.
7. Click the drop-down arrow next to the CalloutEasing property and select one of the options to provide easing to the animation for the callout box when you change the callout position by code.
8. Enter a number, in milliseconds, next to the CalloutAnimationDuration property to specify how long the animation will last for the callout box moving from the last position to the current position.
9. In Design view, click the **Source** tab and enter markup for the button to set the callout's position so it looks like the following:
   ```
   <input type="button" onclick="$find('C1ToolTip1').set_calloutOffset(50)"
   value="Move Callout" />
   ```

In Markup View

To set the **Callout** properties in Markup view, click **View | Markup** in Visual Studio and follow these steps:

1. Create a C1ToolTip by entering the following markup within the **<form>** tags of the page, after the **ScriptManager** tags. Here is an example:
   ```
   <cc1:C1ToolTip ID="C1ToolTip1" runat="server" CloseBehavior="None"
   CssClass="leftsideTooltip" CalloutSide="Left" ShowCallOut="true"
   Width="60px" Height="320" CalloutOffset="20" Text="asp"
   VisualStyle="Vista">
   </cc1:C1ToolTip>
   ```
2. Add a Button control to your page by adding the following markup within the **<body>** tags after the **<form>** tags:
   ```
   <p>
   ```

```
         <input id="Button1" type="button"
onclick="$find('C1ToolTip1').set_calloutOffset(50)" value="Move Callout"
/></p>
```

3. Add the following css style to the <head> tag of the page:

```
<style type = "text/css">
    .leftsideTooltip
    {
       position:absolute;
       left:100px;
       top:100px;
    }
    </style>
```