ComponentOne

ComboBox for ASP.NET AJAX

Copyright © 1987-2010 ComponentOne LLC. All rights reserved.

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 ComboBox for ASP.NET AJAX

Display a resizable and editable text box/drop-down list on your Web form with this easy-to-use combo box control. The C1ComboBox control emulates the Microsoft ComboBox control for WinForms, but it offers additional features such as a multiple selection option, data loading on demand, and template support for embedding HTML controls within C1ComboBox.

Getting Started

To get started, review the following topics:

- Key Features (page 16)
- Quick Start (page 19)
- Samples (page 39)

What's New in ComboBox for ASP.NET AJAX

There were no new features added to **ComponentOne ComboBox for ASP.NET AJAX** for the 2010 v2 release of the ComponentOne Studios.

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 **ComboBox for ASP.NET AJAX**.

What's New in 2010 v1

The following enhancements were made to **ComboBox for ASP.NET AJAX** in the 2010 v1 release of the ComponentOne Studios.

Multi-column support has been added to **ComponentOne ComboBox for ASP.NET AJAX** for 2010 v1. See Adding Multiple Columns to C1ComboBox (page 41) for more information.

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 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 | Samples | Palomino Samples**.

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 Windows 7/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 (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, frequently asked licensing questions, and the ComponentOne licensing model, is available online at http://www.componentone.com/SuperPages/Licensing/.

Licensing FAQs

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

What is Licensing?

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

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

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

How does Licensing Work?

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

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

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

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

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

These files are automatically added to the project.

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

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

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

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

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

Common Scenarios

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

Creating components at design time

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

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

Creating components at run time

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

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

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

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

Inheriting from licensed components

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

- Add a LicenseProvider attribute to the component.

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

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

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

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

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

Using licensed components in console applications

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

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

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

Using licensed components in Visual C++ applications

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

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

1. Build the C++ project as usual. This should create an .exe file and also a licenses.licx file with licensing information in it.
2. Copy the licenses.licx file from the 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 **C1ComboBox** using the fully qualified name for this class:

- Visual Basic
```
Dim ComboBox As C1.Web.UI.Controls.C1ComboBox
```
- C#
```
C1.Web.UI.Controls.C1ComboBox ComboBox;
```

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 C1ComboBox = C1.Web.UI.Controls.C1ComboBox
Imports MyComboBox = MyProject.Objects.C1ComboBox

Dim wm1 As C1ComboBox
Dim wm2 As MyComboBox
```

- C#

```
using C1ComboBox = C1.Web.UI.Controls.C1ComboBox;
using MyComboBox = MyProject.Objects.C1ComboBox;

 C1ComboBox wm1;
 MyComboBox 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 ComboBox 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 C1ComboBox 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 C1ComboBox 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 2005, you will notice a **ComponentOne Studio for ASP.NET Projects** tab containing the ComponentOne controls that have automatically been added to the Toolbox.

If you decide to uncheck the **Create a ComponentOne Visual Studio 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 **C1ComboBox** 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.C1ComboBox. 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.Controls.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 (page 12) for more information.

Key Features

The following are some of the main features of C1ComboBox that you may find useful:

- **Users can select one or multiple items from the C1ComboBox at run time**

  C1ComboBox provide two different selection modes, **Single** mode (default) and **Multiple** mode, so you can allow users to select one or several items in the drop-down list. Use the SelectionMode property to select a mode. See Changing the C1ComboBox Selection Mode (page 41) for more information.

- **Load data on demand**

  Keep page sizes small and manageable using **C1ComboBox's** load on demand feature. C1ComboBox fires a server-side event, **C1ComboBox.OnItemsRequested**, that allows you to populate items based on the requested information. See the Load on Demand sample (page 39) installed with the product for a detailed example on using the load on demand feature.

- **Make the C1ComboBox drop-down list resizable at run time**

  You can allow users to resize the C1ComboBox drop-down list by setting the DropDownResizable property to **True**. At run time, users can resize the list by dragging the lower right corner. See Making the C1ComboBox Drop-Down List Resizable (page 43) for more information.

- **Use templates to embed HTML controls within C1ComboBox**

  Templates can be used to embed other HTML controls in the C1ComboBox control, making your application more appealing and informative. An item, header, or footer template can be used. See Using Templates to Embed HTML Controls in C1ComboBox (page 34) for more information.

- **C1ComboBox provides autocomplete support**

  The C1ComboBox control will show any items in the drop-down list that match the characters currently entered in the text box area of the control.

- **Built-in visual styles**

  Choose from five built-in visual styles to quickly change the C1ComboBox control's appearance. Visual styles include: **ArcticFox**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, and **Vista**. See Using Built-in Visual Styles (page 45) for additional information.

- **Customize the C1ComboBox control with visual styles and CSS styling**

  Easily customize the C1ComboBox control by pointing to your own visual styles using the **VisualStyle** property.

  ComboBox for ASP.NET AJAX also includes CSS-supported styling so that you can use cascading style sheets to easily style the C1ComboBox control to match the design of your current Web site.

  See Adding Custom Visual Styles (page 44) for more information.

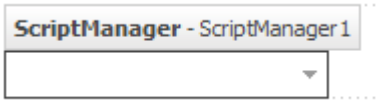
ComboBox for ASP.NET AJAX Quick Start

The C1ComboBox Quick Start describes how to get started with the ASP.NET control, C1ComboBox. In this quick start, you will create an ASP.NET AJAX-Enabled Web Site containing a C1ComboBox that is bound to a data source, **C1ListDemo.mdb**. You will also customize the look and behavior of C1ComboBox.

Step 1 of 4: Adding a C1ComboBox to the Page

In this topic you will add a C1ComboBox control to the page.

1. Begin by creating an ASP.NET AJAX-Enabled Web Site (page 13).
2. Click the **Design** tab to make sure you are in Design view, navigate to the Visual Studio Toolbox, and double-click the **C1ComboBox** control to add it to your form.



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 bind C1ComboBox to a data source, **C1ListDemo.mdb**.

Step 2 of 4: Binding C1ComboBox to a Data Source

In this topic, you will bind C1ComboBox to a data source, **C1ListDemo.mdb**. The **C1ListDemo.mdb** file is installed by default in the **C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Common** (Windows XP) or **C:\Users\<username>\Documents\ComponentOne Samples\Common** (Windows 7/Vista) folder. First, we will add the data source to the project, and then we can bind it to C1ComboBox.

1. In the **Solution Explorer**, right-click the **App_Data** folder, and then click **Add Existing Item**.
2. Locate the **C1ListDemo.mdb** file.
3. In the **Add Existing Item** dialog box, click the **C1ListDemo.mdb** file, and then click **Add**.
4. Go back to the Default.aspx page.
5. Select the C1ComboBox control, and click the Smart Tag (page 25) to open the **C1ComboBox Tasks** menu.
6. Click the **Choose Data Source** drop-down arrow and select **<New data source>**.
7. In the **Data Source Configuration Wizard**, select **Access Database** and click **OK**.
8. Click **Browse** and select **App_Data** under **Project folders** in the **Select Microsoft Access Database** dialog box.
9. Choose **C1ListDemo.mdb** in the **Contents of folder** pane on the right-hand side and click **OK**.

10. Click **Next**. The **Configure the Select Statement** window appears.
    a. Confirm that the **Specify columns from a table or view** option is selected.
    b. In the **Name** drop-down list, select **Customer**.
    c. In the **Columns** box, select **UserCode**, **LastName**, and **FirstName** or any other desired check boxes.
11. Click **Next**. The **Test Query** page appears. Optionally, click **Test Query** to test your query.
12. Click **Finish** to close the wizard and add the data source.
13. Click the smart tag again to open the **C1ComboBox Tasks** menu.
14. Click the drop-down arrow next to the DataTextField property and select **LastName**.

The C1ComboBox control is now bound to the **C1ListDemo.mdb** data source. Customers' last names will appear in the drop-down list of the C1ComboBox control. In the next step, you will customize the C1ComboBox control by enabling auto completion, making it resizable, setting the visual style, and setting the selection mode to **Multiple** so users can choose multiple items from the C1ComboBox drop-down list.

Step 3 of 4: Customizing the C1ComboBox Control

In the previous step, you bound the C1ComboBox control to a data source. In this topic, you will customize the C1ComboBox control by enabling auto completion, making it resizable, setting the visual style, and setting the selection mode to **Multiple** so users can choose multiple items from the C1ComboBox drop-down list.

1. Click the C1ComboBox smart tag () to open the **C1ComboBox Tasks** menu.
2. Click the drop-down arrow next to the **VisualStyle** property and select **Vista** from the list. Notice the C1ComboBox turns dark blue.
3. Right-click C1ComboBox and select **Properties** from the context menu.
4. In the Visual Studio Properties window, click the drop-down arrow next to the AutoComplete property and select **True**.
5. Click the drop-down arrow next to the DropDownResizable property and select **True**.
6. Click the drop-down arrow next to the SelectionMode property and select **Multiple**.

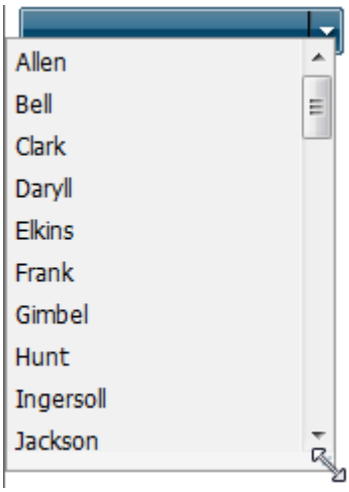
Now that you've customized the C1ComboBox control, let's run the application.

Step 4 of 4: Running the Application

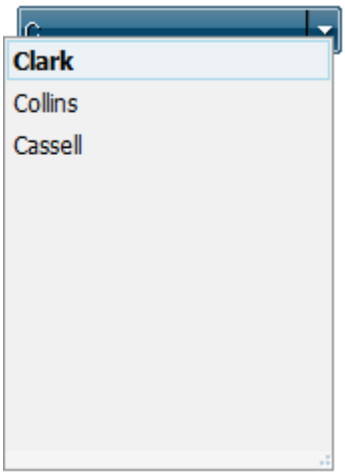
In the previous steps of the quick start you created a simple application with a C1ComboBox control bound to a data source, and you customized the control's appearance and behavior. In this step, you'll run the application and explore some of the run-time interactions possible with the C1ComboBox control.

Complete the following steps:

1. In Visual Studio, select **Debug | Start Debugging** or press **F5** to run the application.
2. Click the drop-down arrow in the C1ComboBox control and scroll the list of last names. Notice that you can resize the list by dragging the bottom right corner.



3. Select the items needed and the names appear with a semicolon between them in the text box of the C1ComboBox.
4. Select the items in the text box and click **Delete**.
5. Now enter a letter in the text box. Notice that all the names beginning with the letter appear in the list.



Congratulations! You've completed the C1ComboBox quick start. If you'd like to continue exploring **ComboBox for ASP.NET AJAX**, see the ComboBox for ASP.NET AJAX Samples (page 39) and take a look at the ComboBox for ASP.NET AJAX Task-Based Help (page 40) topics.

ComboBox for ASP.NET AJAX Top Tips

The following tips were compiled from frequently asked user questions posted in the C1ComboBox forum.

Tip 1: Use Callback Mode to Load C1ComboItem on Demand

C1ComboBox supports callback mode to save internet traffic. To enable callback mode, first set the EnableCallBackMode property to **True**. Then add an ItemsRequested event handler to add items to C1ComboBox. The following code enables call back mode for C1ComboBox. For example:

```
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            c1ComboBoxtest.EnableCallBackMode = true;
            c1ComboBoxtest.ItemsRequested += Requested;
        }
    }

    protected void Requested(object sender,
C1ComboBoxItemsRequestedEventArgs args)
    {
        // get a data table from database
        DataTable dt = GetData(args.FilterText);
        int itemOffset = args.ClientItemsNumber;
        int endOffset = Math.Min(itemOffset + ItemsPerRequest,
dt.Rows.Count);
        args.EndOfItems = endOffset == dt.Rows.Count;

        for (int i = itemOffset; i < endOffset; i++)
        {
            C1ComboBoxItem item = new
C1ComboBoxItem(dt.Rows[i]["Company"].ToString(),
 dt.Rows[i]["Company"].ToString());
            c1ComboBoxtest.Items.Add(item);
        }
        int count = dt.Rows.Count;
        if (count == 0)
        {
            args.Message = "No Match Found";
        }
        else
        {
   args.Message = String.Format("Items <b>1</b>-<b>{0}</b> out of
                          <b>{1}</b>",
                          endOffset, dt.Rows.Count);
        }
}

    private static DataTable GetData(string text)
    {
        string sql;
        if (text.Trim().Length>0)
        {
            sql = "SELECT * FROM Customers WHERE Company LIKE @text +
'%'";
        }
        else
        {
            sql = "SELECT * FROM Customers";
        }
        OleDbDataAdapter adapter = new OleDbDataAdapter(sql,
            "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=|DataDirectory|\\C1ListDemo.mdb;Persist Security Info=True");
```

```
        adapter.SelectCommand.Parameters.AddWithValue("@text", text);
        DataTable data = new DataTable();

        adapter.Fill(data);

        return data;
    }
```

Tip 2: Set ComboBoxItem Image

An image icon can be added to each C1ComboBoxItem. The following sample demonstrates how to add icons to C1ComboBoxItems. For example:

```
<cc1:C1ComboBox ID="c1ComboBox" runat="server">
<Items>
<cc1:C1ComboBoxItem Value="mastercard"
Text="Mastercard"
ImageUrl="card-types/mastercard.png"
DisabledImageUrl="card-types/mastercard_d.png"
HighLightedImageUrl="card-types/mastercard_h.png"> </cc1:C1ComboBoxItem>
<cc1:C1ComboBoxItem Value="express"
Text="American-express"
ImageUrl="card-types/american-express.png"
DisabledImageUrl="card-types/american-express_d.png"
HighLightedImageUrl="card-types/american-
express_h.png"></cc1:C1ComboBoxItem>
<cc1:C1ComboBoxItem Value="electron"
Text="Visa-electron"
ImageUrl="card-types/visa-electron.png"
DisabledImageUrl="card-types/visa-electron_d.png"
HighLightedImageUrl="card-types/visa-
electron_h.png"></cc1:C1ComboBoxItem>
<cc1:C1ComboBoxItem Value="visa"
Text="Visa"
ImageUrl="card-types/visa.png"
DisabledImageUrl="card-types/visa_d.png"
HighLightedImageUrl="card-types/visa_h.png"></cc1:C1ComboBoxItem>
</Items>
</cc1:C1ComboBox>
```

Tip 3: Disable C1ComboBox on the Client Side

C1ComboBox can be enabled or disabled on the client side using JavaScript. For example:

```
function applayEnabled() {
    var combo = $find('<%= c1ComboBoxtest.ClientID%>');
    var v = $get('inputEnabled').checked;
    combo.set_enabled(v);
}
```

Tip 4: Use Multiple Selection Mode

C1ComboBox has two selection modes: single and multiple. The mode can be changed by setting the SelectionMode.

```
   protected void Page_Load(object sender, EventArgs e)
    {
        if (IsPostBack)
        {
            c1ComboBox.SelectionMode = C1ComboBoxSelectionMode.Multiple;
        }
}
```

Tip 5 Multiple Column Support

C1ComboBox supports multiple-column items. You can easity create a grid-like drop-down list. For example:

```
<cc1:C1ComboBox DropDownResizable="true" ID="C1ComboBox6"
runat="server">
    <Columns>
        <cc1:C1ComboBoxColumn
HeaderText="header1"></cc1:C1ComboBoxColumn>
        <cc1:C1ComboBoxColumn
HeaderText="header2"></cc1:C1ComboBoxColumn>
        <cc1:C1ComboBoxColumn
HeaderText="header3"></cc1:C1ComboBoxColumn>
    </Columns>
    <Items>
        <cc1:C1ComboBoxItem Text="item1">
            <Cells>
                <cc1:C1ComboCell Text="item1"></cc1:C1ComboCell>
                <cc1:C1ComboCell Text="test2"></cc1:C1ComboCell>
                <cc1:C1ComboCell Text="test3"></cc1:C1ComboCell>
            </Cells>
        </cc1:C1ComboBoxItem>
        <cc1:C1ComboBoxItem Text="item2">
            <Cells>
                <cc1:C1ComboCell Text="item2"></cc1:C1ComboCell>
                <cc1:C1ComboCell Text="test2"></cc1:C1ComboCell>
                <cc1:C1ComboCell Text="test3"></cc1:C1ComboCell>
            </Cells>
        </cc1:C1ComboBoxItem>
        <cc1:C1ComboBoxItem Text="item3">
            <Cells>
                <cc1:C1ComboCell Text="item3"></cc1:C1ComboCell>
                <cc1:C1ComboCell Text="test2"></cc1:C1ComboCell>
                <cc1:C1ComboCell Text="test3"></cc1:C1ComboCell>
            </Cells>
        </cc1:C1ComboBoxItem>
    </Items>
</cc1:C1ComboBox>
```

Tip 6 Make Filtering Case-Sensitive

If any text is typed into the textbox of the C1ComboBox control, the C1ComboBoxItems in the drop-down list will be filtered according to the text. The filtering is case-insensitive by default, but you can easily make it case-sensitive by setting the IsCaseSensitive property to **True**.

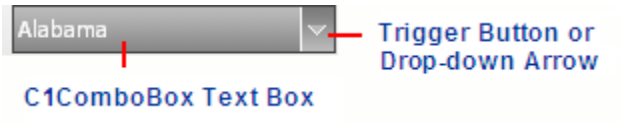
C1ComboBox Elements

This section provides a visual and descriptive overview of the elements that comprise the C1ComboBox control.

The C1ComboBox control is made up of an editable text box and drop-down list.

C1ComboBox Text Box

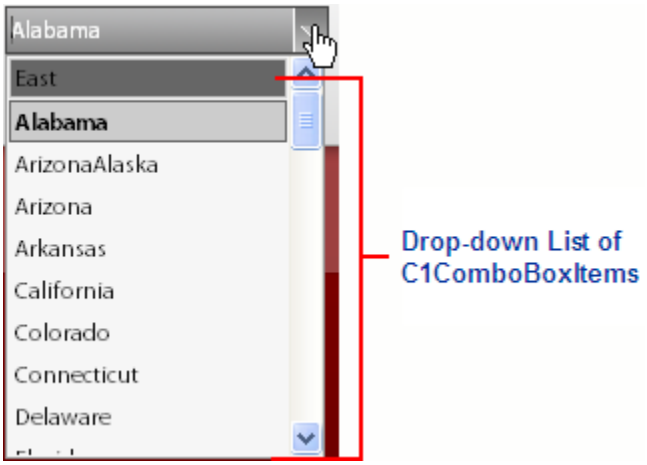
Users can enter text in the C1ComboBox text box at run time.



As text is entered, C1ComboBoxItems matching the characters will appear in the drop-down list. For example, if you enter "Ro" in the text box for a C1ComboBox listing customer names, any customer names starting with "Ro" appear in the drop-down list.

Drop-down List of C1ComboBoxItems

The drop-down list is made up of C1ComboBoxItems and is only visible at run time. You can access the drop-down list by clicking the trigger button, or drop-down arrow, next to the C1ComboBox text box.



You can add items to the list of C1ComboBoxItems using the Add method, declaring items on the ASPX page, or you can bind the C1ComboBox to a data source.

C1ComboBox Design-Time Support

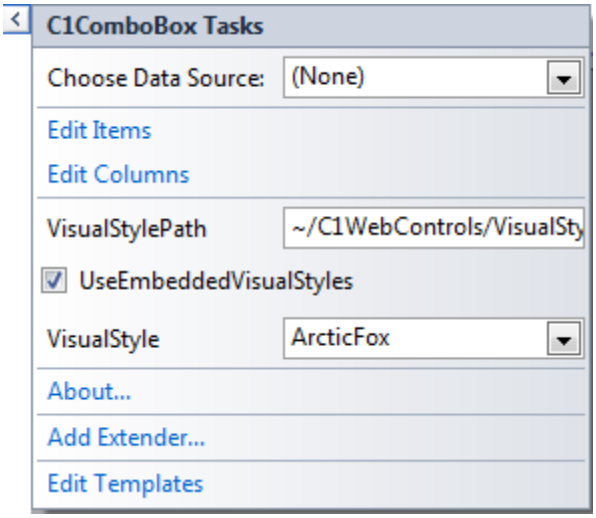
The following sections describe how to use **C1ComboBox**'s design-time environment to configure the C1ComboBox control.

C1ComboBox Smart Tag

The C1ComboBox control includes a smart tag () in Visual Studio. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in C1ComboBox.

The C1ComboBox control provides quick and easy access to common properties through its smart tag.

To access the **C1ComboBox Tasks** menu, click the smart tag in the upper-right corner of the C1ComboBox control.



The **C1ComboBox Tasks** menu operates as follows:

- **Choose Data Source**

  Clicking the drop-down arrow next to the **Choose Data Source** item opens a drop-down list where you can choose an existing data source or select a new data source to bind to and populate your C1ComboBox.

- **Configure Data Source (only visible if you specify a data source)**

  Clicking the **Configure Data Source** link allows you to determine which items from your data source will be used to populate C1ComboBox.

- **DataTextField (only visible if you specify a data source)**

  Clicking the drop-down arrow next to **DataTextField** opens a drop-down list where you can choose the field in the data source used to load the text values, or items that appear in the C1ComboBox drop-down list.

- **DataValueField (only visible if you specify a data source)**

  Clicking the drop-down arrow next to **DataValueField** opens a drop-down list where you can choose the field in the data source used to load the values for the items that appear in the C1ComboBox drop-down list.

- **Edit Items (only visible if you do not specify a data source)**

  Clicking the **Edit Items** link opens the **C1ComboBox Editor** where you can add and remove **C1ComboBoxItems** and change **C1ComboItem** properties, such as appearance, behaviors and so on.

- **Edit Columns**

  Clicking the **Edit Columns** link opens the **Columns Designer** dialog box where you can add and remove columns and set their properties.

- **VisualStylePath**

  The VisualStylePath property specifies the location of the visual styles used for the control. The default value for VisualStylePath property is ~/C1WebControls/VisualStyles. If you create a custom style, add it to this location **~/VisualStyles/StyleName/C1ComboBox/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 checkbox is checked by default so that the visual styles embedded in C1.Web.UI.Controls.2.dll, such as **ArcticFox** and **Vista**, can be used. If you want to use your own custom styles, uncheck this checkbox and specify the location of your visual styles using the VisualStylePath property.

- **VisualStyle**

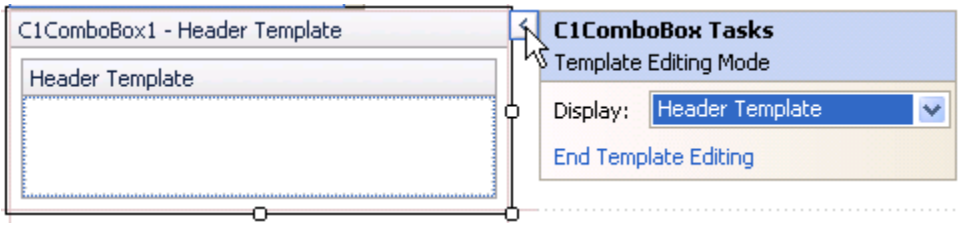
  Clicking the **VisualStyle** drop-down arrow allows you to select from various built-in visual styles. See Using Built-in Visual Styles (page 45) for more information.

- **About C1ComboBox**

  Displays the **About ComponentOne ComboBox** 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.

- **Edit Templates**

  Clicking the **Edit Templates** item switches the C1ComboBox control to Template Editing Mode:



In Template Editing Mode the C1ComboBox menu appears with different options:

- *Display*

  Selecting the **Display** drop-down arrow opens a list of customizable templates, including the **Header** and **Footer**. Select a template from this list to edit it.

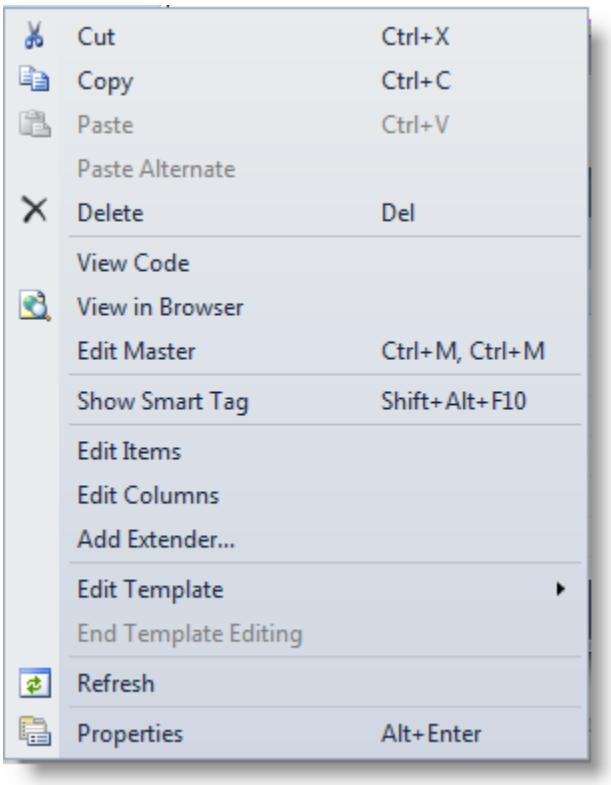
- *End Template Editing*

  Clicking the **End Template Editing** item will end Template Editing Mode and return you to the main **C1ComboBox Tasks** menu.

C1ComboBox Context Menu

C1ComboBox has additional commands available on the context menu that Visual Studio provides for all .NET and ASP.NET controls.

Right-click anywhere on the C1ComboBox control to display the context menu:



The context menu commands operate as follows:

- **Edit Items**

  Clicking **Edit Items** opens the **C1ComboBox Editor** where you can add and remove **C1ComboBoxItems** and change their appearance and behaviors.

- **Edit Template**

  Click **Edit Template** and select **Header** or **Footer** to begin Template Editing Mode where you can enter text for the header or footer of the C1ComboBox. If you are in Template Editing Mode and you right-click C1ComboBox to view the context menu, there is now an E**nd Template Editing** option you can click to exit this mode.

C1ComboBox Editor

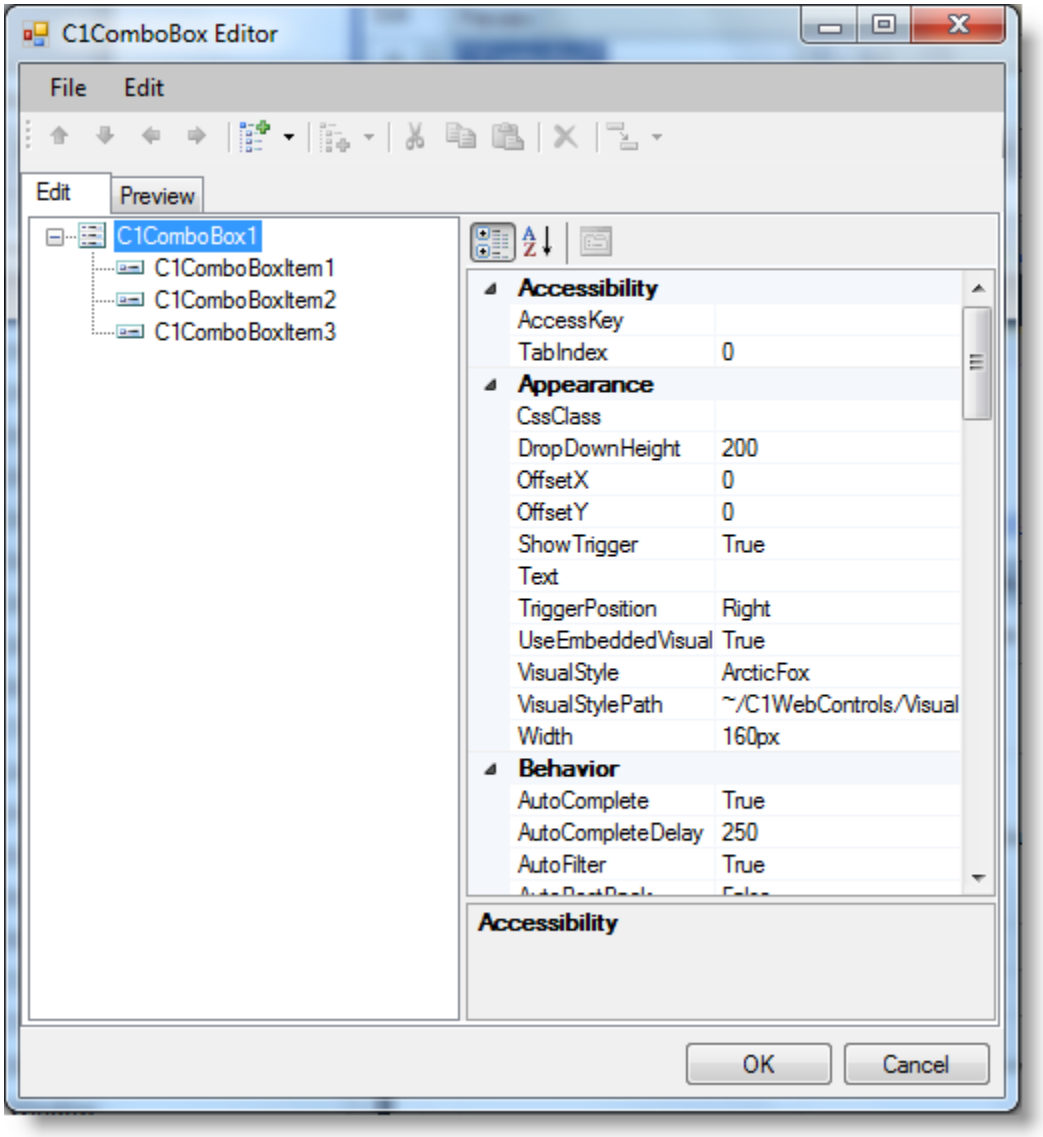
The **C1ComboBox Editor** is **C1ComboBox's** designer for editing its properties, as well as the **C1ComboBoxItem** properties. The **C1ComboBox Editor** is similar to the Properties window as it allows programmers to modify the control visually. However, it allows you to: import C1ComboBoxItem**s** and their formatting from an .xml file; export C1ComboBoxItem**s** and their formatting to an .xml file; set C1ComboBox and C1ComboBoxItem properties; rearrange the order of C1ComboBoxItem**s**; and add or remove C1ComboBoxItem**s**.

In this topic you will become familiar with the **C1ComboBox Editor's** design interface so you can use the commands within it to edit C1ComboBox with minimal effort and time.

To open the **C1ComboBox Editor**, click the C1ComboBox smart tag and select the **Edit Items** link from the **C1ComboBox Tasks** menu:

Exploring the C1ComboBox Editor

The **C1ComboBox Editor** contains a menu, toolbar, **Edit** tab, **Preview** tab, and properties pane.



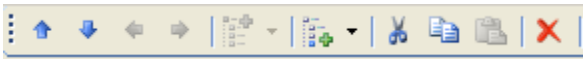
- **C1ComboBox Editor Menu**

  The **C1ComboBox Editor** menu contains the following menu items and subitems:

Menu Item	Submenu Item	Description
File	Load from XML	Load the formatting for a C1ComboBox control from an .xml file.
	Save as XML	Save the current formatting of the C1ComboBox control to an .xml file.
	Exit	Closes the **C1ComboBox Editor**.

Edit	Insert Item	Inserts a new **C1ComboBoxItem** at the specified place in the list of items.
	Add Child	Adds a new **C1ComboBoxItem** as a child of the C1ComboBox.
	Cut	Cuts the selected **C1ComboBoxItem** to be moved in the list of items.
	Copy	Copies the selected **C1ComboBoxItem**.
	Paste	Pastes a **C1ComboBoxItem** at the specified location in the list of items.
	Delete	Removes the selected **C1ComboBoxItem**.
	Rename	Allows you to change the name of the **C1ComboBoxItem**.

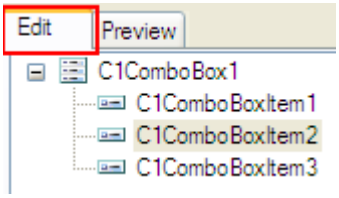
- **C1ComboBox Editor Toolbar**



The table below describes each button in the toolbar:

Button	Name	Description
	Move Item Up	Moves the selected **C1ComboBoxItem** up the list of items.
	Move Item Down	Moves the selected **C1ComboBoxItem** down the list items.
	Move Item Left	Moves the selected **C1ComboBoxItem** to the left in the hierarchy.
	Move Item Right	Moves the selected **C1ComboBoxItem** to the right in the hierarchy.
	Add Child Item	Inserts a **C1ComboBoxItem** as a child of the C1ComboBox control.
	Insert Item	Inserts a **C1ComboBoxItem** at the specified location in the list of items.
	Cut	Cuts the selected **C1ComboBoxItem** to be moved in the list of items.
	Copy	Copies the selected **C1ComboBoxItem**.
	Paste	Pastes a **C1ComboBoxItem** at the specified location in the list of items.
	Delete	Removes the selected **C1ComboBoxItem**.

- **Edit Tab**

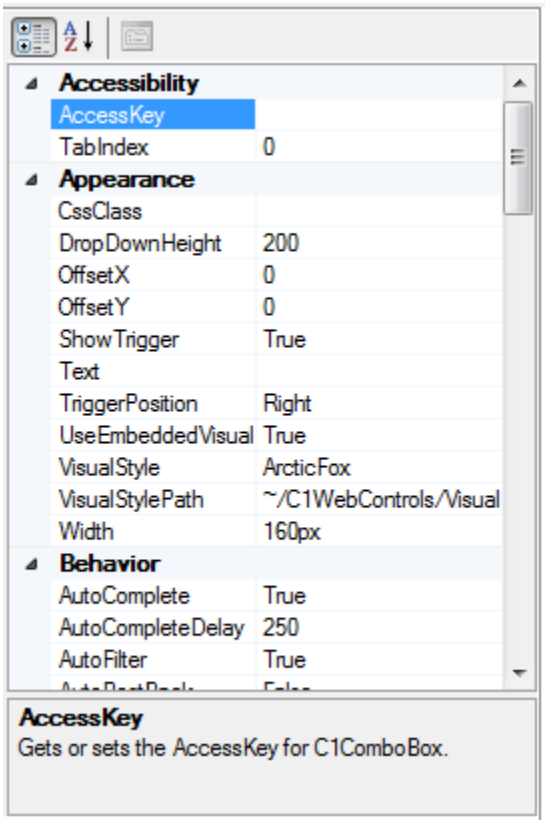


Click the **Edit** tab and select the C1ComboBox control or the desired **C1ComboBoxItem** for which you would like to manipulate or adjust the properties.

- **Preview Tab**

  Click the **Preview** tab for a preview of what the C1ComboBox control will look like.

- **Properties Pane**



  The **C1ComboBox Editor** Properties pane is almost identical to the Visual Studio Properties window. Simply select a C1ComboBoxItem or the C1ComboBox control 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 C1ComboBox control.
Cancel	Clicking **Cancel** closes the **C1ComboBox Editor**, cancelling the new settings and applying the default settings to the C1ComboBox control.

CombBox for ASP.NET Appearance

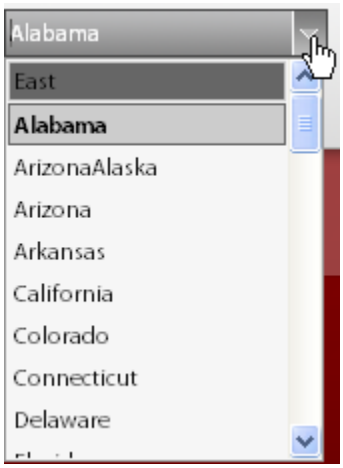
The following sections describe how to change the appearance of C1ComboBox using built-in visual styles as well as how to use your own custom styles.

C1ComboBox Visual Styles

ComponentOne ComboBox for ASP.NET AJAX provides five built-in visual styles, allowing you to automatically format the C1ComboBox control. The styles include the following: **ArcticFox**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, and **Vista**. For more information, see Using Built-in Visual Styles (page 45).

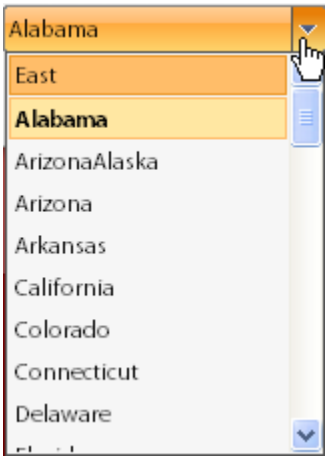
ArcticFox Style

The following image displays the **ArcticFox** style. This is the default format of the C1ComboBox control:



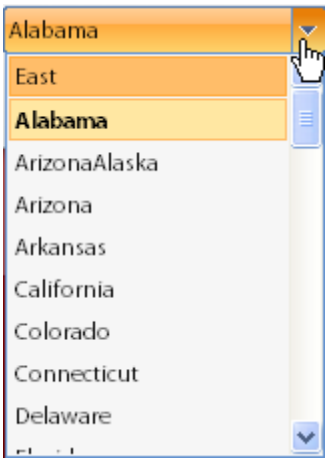
Office2007Black Style

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



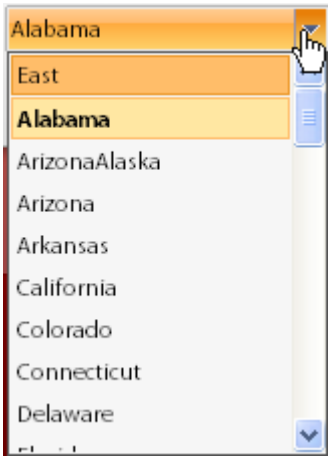
Office2007Blue Style

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



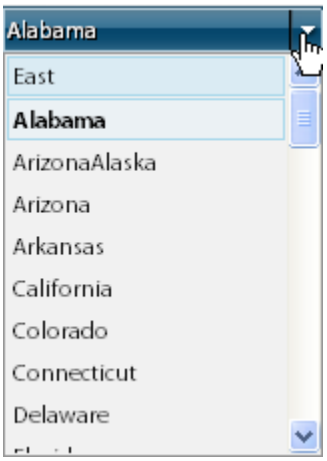
Office2007Silver Style

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



Vista Style

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



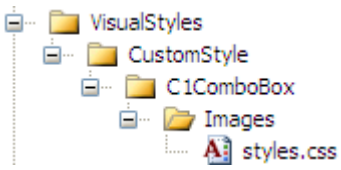
Note: An additional **Windows7** style is installed with the product, but it must be added as a custom visual style at this time. See Adding Custom Visual Styles (page 44) for more information.

Custom Visual Styles

While **ComboBox 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 **ComboBox for ASP.NET AJAX** control, you will create a custom CSS style sheet and add it to your project as a visual style. The custom CSS stylesheet must always be named "styles.css".

Tip: The easiest way to create a custom visual styles is by modifying one of the control's pre-existing visual styles. You can find the .css sheets and images for the **ComboBox 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 "C1ComboBox". The image folder and .css file should be placed underneath the **C1ComboBox** folder. The result will resemble the following:



This structure of these folders is very important; **ComboBox for ASP.NET AJAX** will always look for the **~/VisualStyles/StyleName/C1ComboBox/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 **ComboBox for ASP.NET AJAX** controls, see Adding Custom Visual Styles (page 44).

Using Templates to Embed HTML Controls in C1ComboBox

Templates can be used to embed other HTML controls in the C1ComboBox control. An item, header, or footer template can be used. C1ComboBox includes special template designers for customizing the C1ComboBox header and footer areas. The templates are useful for making the C1ComboBox more coherent to your application and for displaying additional information, such as images.

Item Template

To use an item template, complete the following steps. In this example, we'll add a **C1Calendar** control to the C1ComboBox.

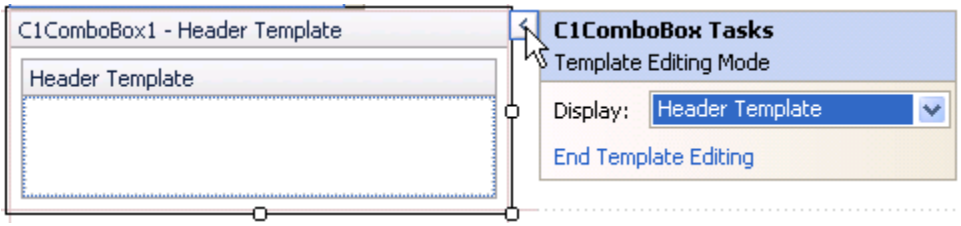
1. Click the **Source** tab to view the source markup for your project.
2. Add the <ItemsTemplate> tag within the C1ComboBox tags so it looks like the following:

```
<cc1:C1ComboBox ID="C1ComboBox1" runat="server"
        DataSourceID="AccessDataSource1" TemplateItemHighlighted="True">

        <ItemsTemplate>
            <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
        </ItemsTemplate>

    </cc1:C1ComboBox>
```

Header and Footer Templates

To use header or footer templates, complete the following steps:

1. Click the C1ComboBox smart tag to open the **C1ComboBox Tasks** menu.
2. Click **Edit Templates**. Template editing mode begins.



3. Click the drop-down arrow next to **Display** to switch between **Header Template** and **Footer Template**, depending on which you want to use.
4. Drag a control into the template area. You can drag and drop arbitrary controls and then modify the control's properties through the Properties window. For example, if you add a **Button** control, right-click it and select **Properties** to modify its properties.
5. Select **End Template Editing** to end the template editing mode.

Expanding and Collapsing C1ComboBox

The following topics describe the different animation effects when expanding and collapsing the C1ComboBox control and the duration of the animation effects.

Animation Effects

C1ComboBox includes thirty-one different built-in animation options that allow you to customize how animation effects are transitioned in the C1ComboBox 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 EaseLiner, and the control expands and collapses with a smooth linear transition effect.

The Setting the Animation Effects for C1ComboBox (page 41) topic shows you how to set the ExpandEasing property.

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.

Animation Duration

You can set how long the C1ComboBox expand or collapse animation effects take by using the CollapseDuration and ExpandDuration property. The unit of time used for specifying animation effect duration is in milliseconds, and the default setting is 1000 milliseconds, or one second. Increase this value for a longer animation effect, and decrease this number for a shorter animation effect.

The Setting the Animation Effects for C1ComboBox (page 41) topic shows you how to set the CollapseDuration property.

Client-Side Functionality

C1ComboBox includes a robust client-side object model, where a majority of server-side properties can be set on the client side.

When a C1ComboBox control is added to a Web project, an instance of the client-side control will be created automatically. You can then access properties and methods of the C1ComboBox control without having to postback to the server, increasing the efficiency of your Web site and providing a better user experience.

Suppose a C1ComboBox control with the name **C1ComboBox1** is added to a Web page. When the page is rendered, a corresponding **C1ComboBox** object will be created. Use the following syntax to get the client object:

```
$get("C1ComboBox1").control
OR
$find("C1ComboBox1")
```

By using C1ComboBox'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.

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

Client-Side Properties

The following conventions are used when accessing the client object properties:

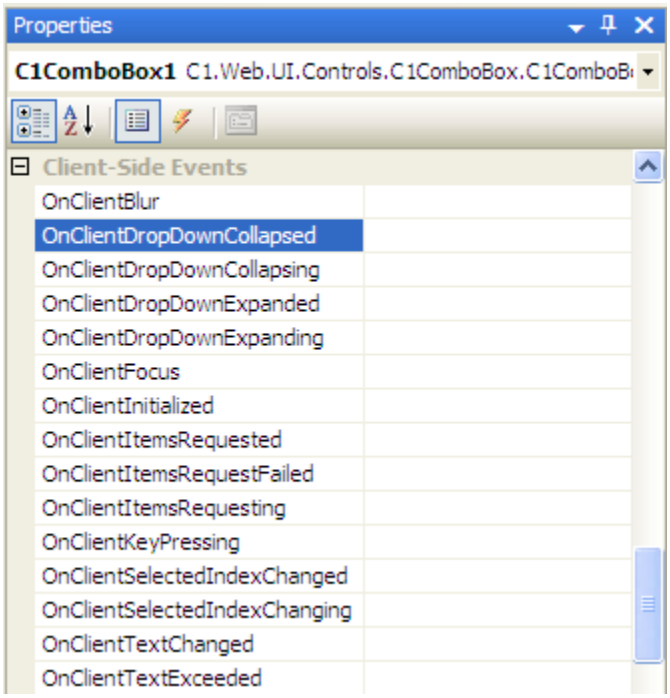
- Server properties on the client are implemented as a pair of get_ and set_ methods.
- Method names must start with "get_" (Get-method) and "set_" (Set-method) followed with the server property name. The first letter of the server property name must be lowercase (camel case).

Client-Side Methods

C1ComboBox includes a rich client-side object model in which several properties and methods can be used on the client side. For more information about the client-side methods and what properties can be set on the client side, see the C1ComboBox client-side reference section of this documentation.

Client-Side Events

C1ComboBox includes a rich client-side object model that includes several client-side events. You can access these client-side events from the Visual Studio Properties window. To create a new client-side event, select the drop-down arrow next to a client-side event and select **Add new client side handler**.



The following table names and describes the built-in client-side events available for your use:

Client-Side Event	Description

OnClientBlur	Fired when the C1ComboBox loses focus.
OnClientDropDownCollapsed	Fired when the drop-down list is closed.
OnClientDropDownCollapsing	Fired when the drop-down list is about to be closed.
OnClientDropDownExpanded	Fired when the drop-down list is expanded.
OnClientDropDownExpanding	Fired when the drop-down list is about to be expanded.
OnClientFocus	Fired when the C1ComboBox gains focus.
OnClientInitialized	The name of the javascript function called after the C1ComboBox is initialized.
OnClientItemsRequested	Fired after the request for items has been completed.
C1ComboBox.OnClientRequestFailed	Fired after the request for items has failed.
C1ComboBox.OnClientRequesting	Fired before the tiems are requested on the server side.
OnClientKeyPressing	Fired when a key is pressed.
OnClientSelectedIndexChanged	Fired after the selected index of the C1ComboBox has changed.
OnClientSelectedIndexChanging	Fired when the selected index of the C1ComboBox is about to change.
OnClientTextChanged	Fired when the Text property is changed.
OnClientTextExceeded	Fired when the text exceeds the maximum length.

ComboBox 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 | Samples | Palomino Samples**.

The following pages within the ControlExplorer sample installed with C**omponentOne Studio for ASP.NET AJAX** detail the C1ComboBox control's functionality:

C# Samples

Sample	Description
Animation	This sample shows how you can set the ExpandAnimation and CollapseAnimation, as well as the ExpandDuration and CollapseDuration.
CSOM	This sample demonstrates using properties in the client-side object model.
DataBind	This sample demonstrates how C1ComboBox can be bound to a data source.
ImageIcon	This sample shows how images can be added to the C1ComboBox items.
Load On Demand	This sample shows how C1ComboBox populates the drop-down list on demand, when a user enters text in the text box.
VisualStyles	This sample displays the built-in C1ComboBox control visual styles including: **ArcticFox**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, and **Vista**.

ComboBox 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 ComboBox for ASP.NET AJAX features and get a good sense of what ComboBox 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 ComboBox 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 C1ComboBox 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).

Binding C1ComboBox to a Data Source

This topic demonstrates how to bind C1ComboBox to a data source control. Once bound, you can use the DataTextField property to populate the **C1ComboBox** drop-down list. In this example, we'll use the **C1ListDemo.mdb** file installed by default in the **C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Common** (Windows XP) or **C:\Users\<username>\Documents\ComponentOne Samples\Common** (Windows 7/Vista) folder.

To bind C1ComboBox to a data source control, complete the following steps:

1. In the **Solution Explorer**, right-click the **App_Data** folder, and then click **Add Existing Item**.
2. Locate the **C1ListDemo.mdb** file.
3. In the **Add Existing Item** dialog box, click the **C1ListDemo.mdb** file, and then click **Add**.
4. Go back to the Default.aspx page.
5. Select the C1ComboBox control, and click the Smart Tag (page 25) to open the **C1ComboBox Tasks** menu.
6. Click the **Choose Data Source** drop-down arrow and select **<New data source>**.
7. In the **Data Source Configuration Wizard**, select **Access Database** and click **OK**.
8. Click **Browse** and select **App_Data** under **Project folders** in the **Select Microsoft Access Database** dialog box.
9. Choose **C1ListDemo.mdb** in the **Contents of folder** pane on the right-hand side and click **OK**.
10. Click **Next**. The **Configure the Select Statement** window appears.
    a. Confirm that the **Specify columns from a table or view** option is selected.
    b. In the **Name** drop-down list, select **Customer**.
    c. In the **Columns** box, select **UserCode**, **LastName**, and **FirstName** or any other desired check boxes.
11. Click **Next**. The **Test Query** page appears. Optionally, click **Test Query** to test your query.
12. Click **Finish** to close the wizard and add the data source.
13. Click the smart tag again to open the **C1ComboBox Tasks** menu.
14. Click the drop-down arrow next to the DataTextField property and select **LastName**. Customers' last names will appear in the drop-down list of the C1ComboBox control.

15. Click the drop-down arrow next to the DataValueField property and select **UserCode**.

Setting the Animation Effects for C1ComboBox

You can add animation effects to the C1ComboBox control by setting any of the animation properties: CollapseEasing, ExpandEasing, and AnimationDuration.

1. Click the C1ComboBox smart tag to open the **C1ComboBox Tasks** menu.
2. Select **Edit Items** and choose **C1ComboBox1**.
3. Set the CollapseDuration property to "1500". This will make the duration of the **C1ComboBox** collapse a little slower so you can see the animation effect.
4. Set the CollapseEasing property to **EaseInBounce**, for example, or one of the other options.
5. Click **OK** to close the **C1ComboBox Editor**. When you run your project and click the drop-down arrow in the C1ComboBox, notice how it bounces when expanded.

Changing the C1ComboBox Selection Mode

C1ComboBox provide two different selection modes, **Single** mode (default) and **Mulitple** mode, so you can allow users to select one or several items in the drop-down list. Use the SelectionMode property to set the selection mode.

Setting the selection mode in Design View

1. Right-click the C1ComboBox control and select **Properties** from the context menu.
2. In the Visual Studio Properties window, click the drop-down arrow next to the SelectionMode property and choose **Single** or **Multiple**.

Setting the selection mode programmatically

Add the following code to your form:

- Visual Basic

```
' add the Imports statement at the top of the form
Imports C1.Web.UI.Controls.C1ComboBox;

Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
        C1ComboBox1.SelectionMode = C1ComboBoxSelectionMode.Multiple
End Sub
```

- C#

```
// add the using statement at the top of the form
using C1.Web.UI.Controls.C1ComboBox;

protected void Page_Load(object sender, EventArgs e)
    {
        C1ComboBox1.SelectionMode = C1ComboBoxSelectionMode.Multiple;
    }
```

If the selection mode is set to **Multiple**, users can use the CTRL button and make multiple selections in the C1ComboBox drop-down list at run time.

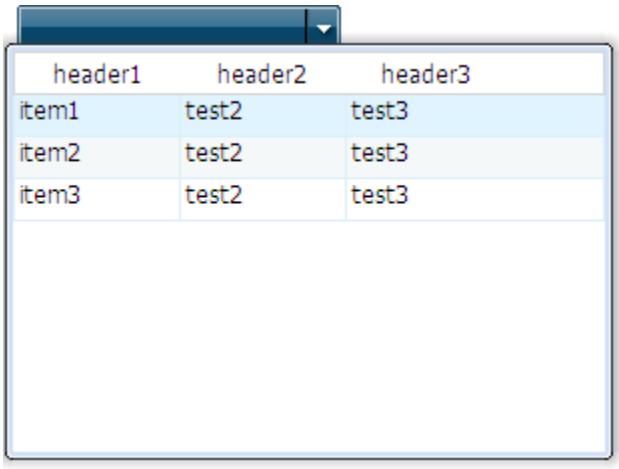
Adding Multiple Columns to C1ComboBox

You can easily add multiple columns to C1ComboBox. The markup used in this example creates a C1ComboBox with three columns.

1. Select **View | Markup** in your AJAX-enabled ASP.NET project.
2. Add the following markup within the tags.

```
<cc1:C1ComboBox ID="C1ComboBox1" DropDownWidth="300" runat="server"
VisualStyle="Vista">
        <Columns>
            <cc1:C1ComboBoxColumn
HeaderText="header1"></cc1:C1ComboBoxColumn>
            <cc1:C1ComboBoxColumn
HeaderText="header2"></cc1:C1ComboBoxColumn>
            <cc1:C1ComboBoxColumn
HeaderText="header3"></cc1:C1ComboBoxColumn>
        </Columns>
        <Items>
            <cc1:C1ComboBoxItem Text="item1">
                <Cells>
                    <cc1:C1ComboCell Text="item1"></cc1:C1ComboCell>
                    <cc1:C1ComboCell Text="test2"></cc1:C1ComboCell>
                    <cc1:C1ComboCell Text="test3"></cc1:C1ComboCell>
                </Cells>
            </cc1:C1ComboBoxItem>
            <cc1:C1ComboBoxItem Text="item2">
                <Cells>
                    <cc1:C1ComboCell Text="item2"></cc1:C1ComboCell>
                    <cc1:C1ComboCell Text="test2"></cc1:C1ComboCell>
                    <cc1:C1ComboCell Text="test3"></cc1:C1ComboCell>
                </Cells>
            </cc1:C1ComboBoxItem>
            <cc1:C1ComboBoxItem Text="item3">
                <Cells>
                    <cc1:C1ComboCell Text="item3"></cc1:C1ComboCell>
                    <cc1:C1ComboCell Text="test2"></cc1:C1ComboCell>
                    <cc1:C1ComboCell Text="test3"></cc1:C1ComboCell>
                </Cells>
            </cc1:C1ComboBoxItem>
        </Items>
    </cc1:C1ComboBox>
```

3. Press **F5** to run the project. The C1ComboBox looks like the following image:



Making the C1ComboBox Drop-Down List Resizable

You can allow users to resize the C1ComboBox drop-down list by setting the DropDownResizable property to **True**.

Setting the DropDownResizable property in Design View

1. Right-click the C1ComboBox control and select **Properties** from the context menu.
2. In the Visual Studio Properties window, click the drop-down arrow next to the DropDownResizable property and select **True**.

Setting the DropDownResizable property programmatically

Add the following code to your form:

- Visual Basic

```
' add the Imports statement at the top of the form
Imports C1.Web.UI.Controls.C1ComboBox;

Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
        C1ComboBox1.DropDownResizable = True
End Sub
```

- C#

```
// add the using statement at the top of the form
using C1.Web.UI.Controls.C1ComboBox;

Protected void Page_Load(object sender, EventArgs e)
    {
        C1ComboBox1.DropDownResizable = true;
    }
```

At run time, users can resize the list by dragging the lower right corner.

Adding Keyboard Accessibility Support to C1ComboBox

You can make C1ComboBox and each of its **C1ComboBoxItems** accessible through the keyboard by setting the AccessKey property.

Setting the access key using the C1ComboBox Editor

1. Click the C1ComboBox smart tag to open the **C1ComboBox Tasks** menu.
2. Select **Edit Items** and choose **C1ComboBox1** in the **ComboBox Editor**. Note that you can also add keyboard access to each of the **C1ComboBoxItems**.
3. Enter the desired letter next to the AccessKey property.
4. Click **OK** to close the **ComboBox Editor**. When you run the project and press the ALT key (ALT + SHIFT if using Firefox) plus the letter key entered in the previous step, **C1ComboBox1** gets the focus.

Setting the access key programmatically

Add the following code to your form:

- Visual Basic

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
        C1ComboBox1.AccessKey = "M"
End Sub
```

- C#

```
protected void Page_Load(object sender, EventArgs e)
    {
        C1ComboBox1.AccessKey = "M";
    }
```

Adding Custom Visual Styles

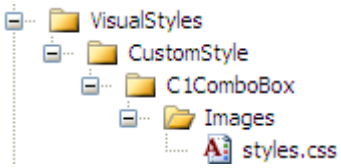
You can use the **VisualStyle**, **VisualStylePath**, and **UseEmbeddedVisualStyles** properties to create a custom visual style for your C1ComboBox. This topic assumes you have already added a C1ComboBox control to your page. For more information on custom visual styles, see Custom Visual Styles (page 34).

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 **C1ComboBox 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\C1ComboBox to a new folder in your Visual Studio project so the folder structure is: ~\VisualStyles\CustomStyle\C1ComboBox.



2. Open the styles.css file in the ~/VisualStyles/CustomStyle/C1ComboBox 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 **.C1ComboBox_CustomStyle** and change the border attribute so it looks like the following:

```
.C1ComboBox_CustomStyle
{
    background: #ff3;
    text-align: left;
    border: solid 1px #444;
}
```

4. Save and close the styles.css file.
5. Select the C1ComboBox control on your form and click the smart tag to open the **C1ComboBox Tasks** menu.
    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.
6. Press F5 to run your project and notice the C1ComboBox control has a yellow background.

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. Click the **Source** tab to open the Source view and enter `VisualStyle="CustomStyle"`, `VisualStylePath="~/VisualStyles"`, and `UseEmbeddedVisualStyles="False"` into the `<cc1:C1Combobox>` tag. Your XHTML will resemble the following:

```
<cc1:C1ComboBox ID=" C1ComboBox 1" runat="server" Height="22px"
VisualStyle="CustomStyle" VisualStylePath="~/VisualStyles"
UseEmbeddedVisualStyles="False" Width="155px" />
```

Adding Custom Visual Styles Programmatically

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
     ```
     C1ComboBox1.UseEmbeddedVisualStyles = False
     ```
   - C#
     ```
     C1ComboBox1.UseEmbeddedVisualStyles = false;
     ```
4. Change the **VisualStylePath** property:
   - Visual Basic
     ```
     C1ComboBox1.VisualStylePath = "~/VisualStyles"
     ```
   - C#
     ```
     C1ComboBox1.VisualStylePath = "~/VisualStyles";
     ```
5. Select the custom visual style:
   - Visual Basic
     ```
     C1ComboBox1.VisualStyle = "CustomStyle"
     ```
   - C#
     ```
     C1ComboBox1.VisualStyle = "CustomStyle";
     ```
6. Run the program and observe that the C1ComboBox control has adopted your custom visual style.

Using Built-in Visual Styles

You can format C1ComboBox with one of five built-in visual styles. Note that you can also use your own style; see Adding Custom Visual Styles (page 44) for more information.

Changing the Visual Style Using the Smart Tag

You can change the style of C1ComboBox at design time using the **C1ComboBox Tasks** menu.

1. Click the C1ComboBox smart tag to open the **C1ComboBox Tasks** menu.
2. Click drop-down arrow next to **VisualStyle**.
3. Select one of the built-in styles listed. The style is applied to the C1ComboBox control.

Changing the Visual Style Programmatically

To change the style of C1ComboBox 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
  ```
  C1ComboBox1.VisualStyle = "Vista"
  ```

- C#

```
C1ComboBox1.VisualStyle = "Vista";
```