ComponentOne

Slider for ASP.NET AJAX

Copyright © 2011 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 Slider for ASP.NET AJAX Overview

ComponentOne Slider for ASP.NET AJAX simplifies user input by providing a rich visual representation of numeric changes. The **C1Slider** control allows you to provide simple, familiar way for users to choose a value in a predefined range. You can customize the appearance of the control using one of several built-in Vista or Office2007 themes, or you can completely design the appearance of the control by using your own custom styles. **Slider for ASP.NET AJAX** includes several accessibility features including keyboard and mouse wheel support, to enhance the experience of users. Take advantage of **ComponentOne Slider for ASP.NET AJAX**'s features today!

Getting Started

Get started with the following topics:

- Key Features (page 16)
- Quick Start (page 19)
- Control Elements (page 25)
- Samples (page 37)

Installing ComponentOne Slider for ASP.NET AJAX

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

Slider 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.
C1WebUi	Contains files (at least a readme.txt) related to the product.
C1WebUi\VisualStyles	Contains all external file themes.

The **ComponentOne Studio for ASP.NET AJAX Help Setup** program installs integrated Microsoft Help 2.0 and Microsoft Help Viewer help to the C:\Program Files\ComponentOne\Studio for ASP.NET directory in the following folders:

H2Help	Contains Microsoft Help 2.0 integrated documentation for all Studio components.
HelpViewer	Contains Microsoft Help Viewer Visual Studio 2010 integrated documentation for all Studio components.

Samples

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

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

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

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

Common	Contains support and data files that are used by many of the demo programs.
Studio for ASP.NET\C1WebUi	Contains samples and tutorials for **ComponentOne Studio for ASP.NET AJAX**.

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 Slider for ASP.NET AJAX** 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

Note: Slider for ASP.NET AJAX requires Microsoft ASP.NET AJAX Extensions installed and a **ScriptManager** on the page before the C1Slider control is placed on the page. You must create an ASP.NET AJAX-Enabled Project so that the **ScriptManager** and Microsoft AJAX Extensions are included on the page. For more information, see Creating an AJAX-Enabled ASP.NET Project (page 13). For more information about Microsoft ASP.NET AJAX Extensions, see http://ajax.asp.net/. For information about the **ScriptManager**, see MSDN.

Installing Demonstration Versions

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

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

Uninstalling Slider for ASP.NET AJAX

To uninstall **ComponentOne Slider for ASP.NET AJAX**:

1. Open the **Control Panel** and select the **Add or Remove Programs** or **Programs and Features** (in Windows 7/Vista).
2. Select **ComponentOne Studio for ASP.NET AJAX** and click the **Remove** button.
3. Click **Yes** to remove the program.

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

1. Open the **Control Panel** and select **Add or Remove Programs** (**Programs and Features** in Windows 7/Vista).
2. Select **ComponentOne Studio for ASP.NET AJAX Help** and click the **Remove** button.
3. Click **Yes** to remove the integrated help.

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 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 application directory to the target folder (Debug or Release).
3. Copy the C1Lc.exe utility and the licensed DLLs to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use C1Lc.exe to compile the licenses.licx file. The command line should look like this:

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

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

Using licensed components with automated testing products

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

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

For example:

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

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

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

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

Troubleshooting

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

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

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

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

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

If that fails follow these steps:

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

6. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

Alternatively, follow these steps:

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

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

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

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

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

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

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

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

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

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

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

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

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

Option 2 – Continue to use the components you have.

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

Technical Support

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

Some methods for obtaining technical support include:

- **Online Resources**
  ComponentOne provides customers with a comprehensive set of technical resources in the form of FAQs, samples and videos, Version Release History, 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.

- **Product Forums**
  ComponentOne's product forums are available for users to share information, tips, and techniques regarding ComponentOne products. ComponentOne developers will be available on the forums to share insider tips and technique and answer users' questions. 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**
  Microsoft integrated ComponentOne documentation can be installed with each of our products, and documentation is also available online. 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 Slider 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, and Visual Studio, 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 trademark of Apple Inc., registered in the U.S. and other countries.

ComponentOne

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

Corporate Headquarters

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

http://www.componentone.com

ComponentOne Doc-To-Help

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

Namespaces

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

The general namespace for ComponentOne Web products is **C1.Web**. The following code fragment shows how to declare a **C1Slider** using the fully qualified name for this class:

- Visual Basic
```
Dim Slider As C1.Web.UI.Controls.C1Slider
```

- C#
```
C1.Web.UI.Controls.C1Slider Slider;
```

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 C1Slider = C1.Web.UI.Controls.C1Slider
Imports MySlider = MyProject.Objects.C1Slider

Dim wm1 As C1Slider
Dim wm2 As MySlider
```

- C#

```
using C1Slider = C1.Web.UI.Controls.C1Slider;
using MySlider = MyProject.Objects.C1Slider;

 C1Slider wm1;
 MySlider 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 Slider 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 C1Slider control is placed on the page. This allows you to take advantage of ASP.NET AJAX and certain features such as partial-page rendering and client-script functionality of the Microsoft AJAX Library.

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

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

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

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:

  a. From the File menu, select **New** | Web Site. The New Web Site dialog box opens.

b. 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.

c. In the list of templates, select **AJAX 1.0-Enabled ASP.NET 2.0 Web Site**.

d. 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.

a. From the **File** menu, select **New** | Project. The New Project dialog box opens.

b. 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.

c. 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**.

d. Select **AJAX 1.0-Enabled ASP.NET 2.0 Web Application** from the list of **Templates** in the right pane.

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

> **Note:** The Web server must have IIS version 6 or later and the .NET Framework installed on it. If you have IIS on your computer, you can specify http://localhost for the server.

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

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

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

a. From the **File** menu in Microsoft Visual Studio .NET, select **New Web Site**. The **New Web Site** dialog box opens.

b. Select **ASP.NET AJAX-Enabled Web Site** from the list of Templates.

c. 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.

  a. From the **File** menu in Microsoft Visual Studio 2005, select **New Project**. The **New Project** dialog box opens.

  b. Under **Project Types**, choose either **Visual Basic Projects** or **Visual C# Projects**. Note that one of these options may be located under **Other Languages**.

  c. Select **ASP.NET AJAX-Enabled Web Application** from the list of **Templates** in the right pane.

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

  Note: The Web server must have IIS version 6 or later and the .NET Framework installed on it. If you have IIS on your computer, you can specify http://localhost for the server.

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

Adding the Slider for ASP.NET AJAX Component to a Project

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

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

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

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

- C1Slider

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.2.dll**. Note that there may be more than one component for each namespace.
5. Click **OK** to close the dialog box.

   The controls are added to the Visual Studio Toolbox.

Adding Studio for ASP.NET 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.

> **Note: Slider for ASP.NET AJAX** requires Microsoft ASP.NET AJAX Extensions installed and a **ScriptManager** on the page before the C1Slider control is placed on the page. You must create an ASP.NET AJAX-Enabled Project so that the **ScriptManager** and Microsoft AJAX Extensions are included on the page. For more information, see Creating an AJAX-Enabled ASP.NET Project (page 13). For more information about Microsoft ASP.NET AJAX Extensions, see http://ajax.asp.net/. For information about the **ScriptManager**, see MSDN.

Adding a Reference to the Assembly

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

1. Select the **Add Reference** option from the **Website** menu of your Web Site project or from the Project menu of your Web Application project.
2. Select the most recent version of the **ComponentOne Studio for ASP.NET AJAX** assembly from the list on the **NET** tab or browse to find the C1.Web.UI.Controls.2.dll file and click **OK**.
3. Select the **Form1.vb** tab or go to **View|Code** to open the Code Editor. At the top of the file, add the following **Imports** directive (**using** in C#):

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

> **Note:** This makes the objects defined in the **C1.Web.UI.Controls.2** assembly visible to the project. See Namespaces (page 12) for more information.

Slider for ASP.NET AJAX Key Features

ComponentOne Slider for ASP.NET AJAX includes several unique features, including the following:

- **Range Slider Support**

  Slider for ASP.NET AJAX includes an option to create a simple slider with one thumb button or a range slider with two thumb buttons. Ranges can be used to filter values within limits. For example, to search an online store for items in a certain price range or search through a list of events for those that happened within a certain date range.

- **CSS Styling**

  Slider for ASP.NET AJAX includes CSS supported styling so that you can use cascading style sheets to easily style the **C1Slider** control to match the design of your current Web site.

- **AJAX Support**

  Built-in AJAX support lets users interact with the **C1Slider** control without the control performing a postback operation back to the server.

- **Set Postbacks**

  Use the AutoPostBack property to determine whether **Slider for ASP.NET AJAX** should perform a postback to the server each time a user interacts with the control. See Setting Post Back (page 38) for more information.

- **Client-Side Object Model**

The **C1Slider** control's client-side object model is exposed so that you can easily customize the control with client-side script.

- **Browser Support**

  Slider for ASP.NET AJAX includes support for the Internet Explorer (6.0 or later), Firefox (2 or later), and Safari Web browsers.

- **XHTML Compliant**

  Slider for ASP.NET AJAX provides complete XHTML compliance. The output that is generated is fully XHTML 1.1 compliant.

- **Visual Styles**

  Use the built-in Visual Styles to quickly change the **C1Slider** control's appearance. Built-in styles include Office 2007 and Vista styles. See Visual Styles (page 31) for more information.

- **Accessibility**

  Slider for ASP.NET AJAX includes options for adding keyboard support, and includes mouse wheel support. Users have several options for navigation, and features such as ToolTips help guide users when using **Slider for ASP.NET AJAX**.

- **Vertical and Horizontal Layout Options**

  Using the Orientation property you can set if the slider control should appear horizontally (default) or vertically on the page. This allows you to further customize the layout of the control in relation to other elements on the page.

- **Slider Direction**

  You can now reverse the slider's direction using the Reverse property. The Reverse property can be used to swap the meaning of the minimum and maximum value endpoints of the slider so you can set the slider to increase value when moving in either direction.

Slider for ASP.NET AJAX Quick Start

The following quick start is designed to quickly familiarize you with the features for the C1Slider control. In this quick start you'll create a simple Web site that uses the C1Slider control to change the number that appears in a text box.

Step 1 of 3: Adding C1Slider to the Page

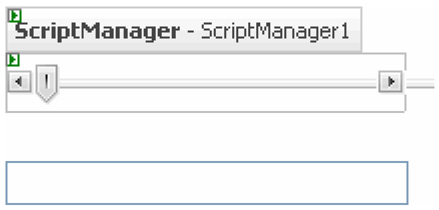
In this step you'll add a **C1Slider** control and a **TextBox** control to the page.

Complete the following steps:

1. Create a new ASP.NET AJAX-Enabled Website project.

   Note that as you've created an AJAX-Enabled Web site, a **ScriptManager** control initially appears on the page.
2. Click the **Design** tab located below the Document window to switch to Design view.
3. Navigate to the Visual Studio Toolbox and double-click the **C1Slider** control to add it to the page.
4. With the page selected, press the ENTER button twice to move to the next line.
5. In the Visual Studio Toolbox double-click the **Input (text)** icon to add an input box to the page below the slider.
6. Resize the text box so that it's the same width as the slider control.

   The page will now look similar to the following image:



Now that you've created your Web site and added controls to the page, the next step is to customize the controls with code and JavaScript.

Step 2 of 3: Customizing the Control

In this step you'll customize the **C1Slider** control by setting its starting values and adding script to the page. The following steps assume you've completed the Step 1 of 3: Adding C1Slider to the Page (page 19) topic and added the **TextBox** and **C1Slider** controls to the page.

Complete the following steps:

1. In Design view click the **C1Slider** control to select it.
2. Navigate to the Properties window and expand the **Client-side events** node, if needed.
3. Select the drop-down arrow next to **OnClientValueChanged**, and select **Add new client side handler** to switch to Source view and add the **C1Slider1_OnClientValueChanged** event handler.
4. Add JavaScript to the **C1Slider1_OnClientValueChanged** event handler so that it looks like the following :

```
function C1Slider1_OnClientValueChanged(sender, eventargs)
{
    document.getElementById("Text1").value =
$find("C1Slider1").get_value();
};
```

Now the text box's value will change when the **C1Slider**'s value changes.

5. Add the following JavaScript just below the **C1Slider1_OnClientValueChanged** event handler:

```
function Text_changed()
{
    var val=parseInt(document.getElementById("textBox").value);
    $find("C1Slider1").set_value(val);
}
```

The **Slider**'s value will now change when the text box's value changes.

6. Add `onchange="Text_changed()"` to the `<input>` Tag so it appears similar to the following:

```
<input id="Text1" type="text" style="width: 195px"
onchange="Text_changed()" />
```

This links to the **onchange** event.

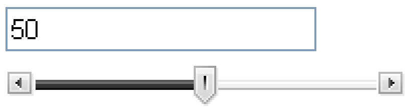
In this step you've customized the controls with JavaScript. In the next (and last) step you'll run your application and see the **C1Slider** control in action.

Step 3 of 3: Running the Application

In the previous steps you created and customized the **C1Slider** control. In this last step you'll run your application and view the **C1Slider** control at run time. Note that the following steps assume you've completed the Step 2 of 3: Customizing the Control (page 19) topic.

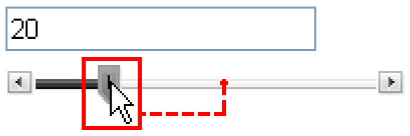
Complete the following steps:

1. Run your application. The page will look similar to the following:



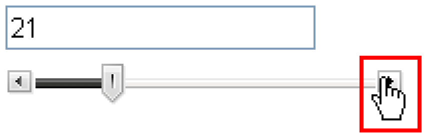
Note that the initial value of the **C1Slider** control reflects the initial value of the **TextBox** control.

2. Click the slider's thumb and perform a drag-and-drop operation to change the control's value:



Note that as you change the slider's value, the value within the text box also changes.

3. Click the **Increment** button:



Note that the value in the text box increases by 1. The amount the C1Slider control's value changes when the **Increment** and **Decrement** buttons are pressed is controlled by the Step property.

In this step you've observed the run-time interaction of the **C1Slider** control. After completing this quick start you've successfully created Web site with a simple **C1Slider** application.

Slider for ASP.NET AJAX Top Tips

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

Tip 1: Display two Thumb buttons

A second thumb button can be added to the C1Slider control by setting both ShowThumbButton2 and ShowThumbButton properties to **True.** Even if the ShowThumbButton2 property is set to **True**, the two thumb buttons will not be shown if the ShowThumbButton property is set to **False**.

The following code sets the ShowThumbButton2 property to show the second thumb button:

- Visual Basic

```
Me.C1Slider1.ShowThumbButton = True
Me.C1Slider1.ShowThumbButton2 = True
```

- C#

```
this.C1Slider1.ShowThumbButton = true;
this.C1Slider1.ShowThumbButton2 = true;
```

Note that the two thumb buttons will not be shown though ShowThumbButton2 property is set to **True** if the ShowThumbButton property is set to **False**.

Tip 2: Customize Thumb button interaction

The ThumbsInteractionMode property can be used to customize the way that the two C1Slider thumb buttons interact . You can set the ThumbsInteractionMode property to **Free**, **Locked**, or **Push**.

The following table describes each of the ThumbsInteractionMode options:

Member Name	Description
Free	Thumbs are free to slide past each other.
Locked	Thumbs cannot slide past or push each other.
Push	Thumbs will push each other when they slide next to one another.

The following code sets the ThumbsInteractionMode property to show that thumbs cannot slide past or push each other:

- Visual Basic

```
Me.C1Slider1.ThumbsInteractionMode =
C1.Web.UI.Controls.C1Slider.SliderThumbsInteractionMode.Push
```

- C#

```
this.C1Slider1.ThumbsInteractionMode =
C1.Web.UI.Controls.C1Slider.SliderThumbsInteractionMode.Push;
```

Tip 3: Reverse the Increment and Decrement buttons

The Reverse property can be used to reverse **Increment/Decrement** buttons. When the **Increment** and **Decrement** buttons are reversed, the **Increment** button will decrease the Value and the **Decrement** button will increase the Value. For example:

- Visual Basic

```
Me.C1Slider1.Reverse = True
```

- C#

```
this.C1Slider1.Reverse = true;
```

Tip 4: Customize the appearance of the selected region on the track

You can set the TrackFill property to determine how the selected region on the slider track will be displayed. You can set the TrackFill property to **None**, **MainValue**, **Range**, or **All**. The following table describes each of the TrackFillMode options:

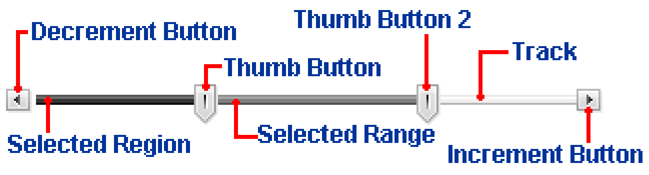
Member Name	Description
None	Values on the slider track bar are not displayed.
MainValue	Shows extra style with ValueFill in track bar for area between edge and main Value only.
Range	Shows extra style with ValueFill in track bar for area between Value and SecondaryValue.
All	Shows extra styles in track bar for both Value and SecondaryValue.

The following code sets the TrackFill property to show the selection region with **ValueFill** in the track bar for the area between Value and Value2:

- Visual Basic
  ```
  Me.C1Slider1.TrackFill = C1.Web.UI.Controls.C1Slider.TrackFillMode.Range
  ```
- C#
  ```
  this.C1Slider1.TrackFill =
  C1.Web.UI.Controls.C1Slider.TrackFillMode.Range;
  ```

C1Slider Elements

This section provides a visual and descriptive overview of the elements that comprise the C1Slider control. The **C1Slider** control consists of several distinct elements. The **C1Slider** control includes a track with a selected region, a thumb button that can be moved on the track, and decrement and increment buttons that can also cause the thumb to move on the track.

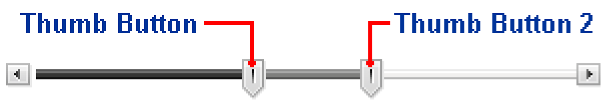


- **Selected region**: The selected region visualizes the selected range to the first thumb button in a contrasting color.
- **Selected range**: The selected region visualizes the selected range from the first thumb button to the second thumb button in a contrasting color.
- **Thumb button**: Changes the Value of the slider by dragging the handle with the mouse.
- **Thumb button2**: Changes the Value2 of the slider by dragging the handle with the mouse.
- **Track**: Visualizes the slider and changes the Value and Value2 when clicked.
- **Decrement button**: Decreases the Value of the slider by a single Step.
- **Increment button**: Increases the Value of the slider by a single Step.

The following topics are categorized into the control's distinct elements that represent different aspects of the control.

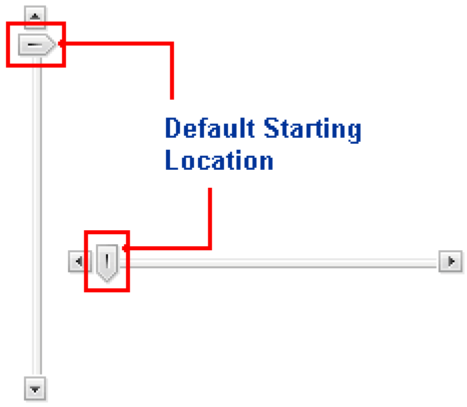
Thumb Buttons

The C1Slider control includes two thumb buttons. The first main thumb button and a second thumb button used for selecting a value range. By default the first thumb button is visible and the second is not visible. The value of the first thumb button is defined by the Value property. The value of the second thumb button is defined by the Value2 property.



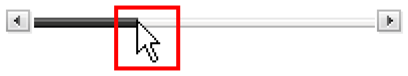
The main thumb button of the C1Slider control can be moved on the slider track through a drag-and-drop operation or by pressing the **Increment** and **Decrement** buttons. The second thumb button can also be moved by a drag-and-drop operation, but cannot be moved by clicking the **Increment** and **Decrement** buttons as these buttons only effect the Value property.

By default, if no Value has been set, the main slider thumb will appear on the slider track next to the decrement button at run time – the left side of the control if Orientation is set to **Horizontal** and the top of the control if Orientation is set to **Vertical**:



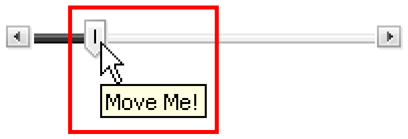
The second slider thumb appears at the opposite end of the slider track by default – the right side of the control if Orientation is set to **Horizontal** and the bottom of the control if Orientation is set to **Vertical**.

By default the first thumb is visible on the track and the ShowThumbButton property is set to **True**. If you do not wish the thumb to be visible, you can set the ShowThumbButton property to **False**. The **C1Slider** will appear similar to the following:



Note that value of the **C1Slider** control can still be changed by clicking on the slider's track or by clicking the **Increment** and **Decrement** buttons, but the thumb will not be seen. By default the second thumb button is not visible. To set it to be visible, set the ShowThumbButton2 property to **True**.

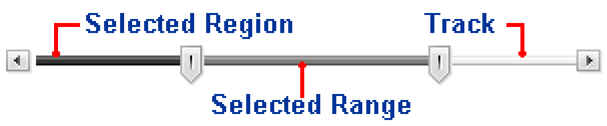
You can set the ThumbToolTip and Thumb2ToolTip properties if you want text to appear when the mouse hovers over the slider's thumb buttons. This can be helpful to state the slider's current value or prompt the user for an action. For example, in the following image the ThumbToolTip property was set to "Move Me!":



By default the ThumbToolTip and Thumb2ToolTip properties are not set and no ToolTips will appear on the slider's thumb buttons.

Slider Track, Selected Region, and Range

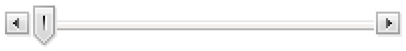
The track area of the C1Slider control forms the basis of the slider. The slider track is the area on which the thumb button moves to set the Value property. The **Decrement** and **Increment** buttons appear on either end of the slider track. The selected region area of the slider track visually indicates the difference between the MinimumValue and the current selected Value.



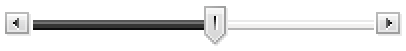
The selected range of the track visually indicates the difference between the selected Value and the selected Value2. Ranges can be used to filter values within limits. For example, to search an online store for items in a certain price range or search through a list of events for those that happened within a certain date range.

The slider track visually indicates the current Value though the use of shading. When the thumb button is moved or the Value is changed the slider track indicates the current value through a darker selected range of the slider track.

By default the Value is set to the MinimumValue and the slider track appears a uniform color:



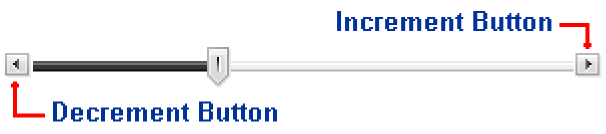
As the Value changes, so does the appearance of the track to indicate the current value and selected region:



You can customize the appearance of the slider's track by setting the TrackFill property. See Slider Track Fill (page 32) for more information.

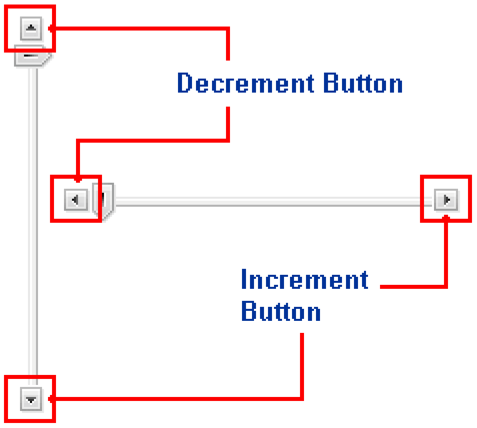
Increment and Decrement Buttons

The C1Slider control's **Increment** and **Decrement** buttons appear on either side of the slider track. Clicking the **Increment** and **Decrement** buttons changes the Value property and the position of the thumb button on the slider track.



Note: Clicking the **Increment** and **Decrement** buttons only changes the Value property and the position of the default thumb button on the slider track. It does not change the Value2 property and the position of the second thumb button, if used.

By default the **Decrement** button will appear on the left side of the slider if Orientation is set to **Horizontal** and the top of the control if Orientation is set to **Vertical**. By default the **Increment** button will appear on the right side of the slider if Orientation is set to **Horizontal** and the bottom of the control if Orientation is set to **Vertical**.



By default the **Increment** and **Decrement** buttons increase or decrease the slider's Value by 1 unit. To change the number by which the slider's Value is increased or decreased, you can use the Step property. Set the Step property to the number with which you want the Value to step up or down in value when the thumb is moved or the **Increment** or **Decrement** buttons are clicked.

By default the **Increment** and **Decrement** buttons are visible on each end of the slider track and the ShowIncrementButton and ShowDecrementButton properties are set to **True**. If you do not wish the **Increment** and **Decrement** buttons to be visible, you can set the ShowIncrementButton and ShowDecrementButton properties to **False**. The **C1Slider** will appear similar to the following:

You can set the IncrementToolTip and the DecrementToolTip properties if you want text to appear when the mouse hovers over the slider's **Increment** and **Decrement** buttons. This can be helpful to state the slider's minimum and maximum values or prompt the user for an action.

Slider for ASP.NET AJAX Design-Time Support

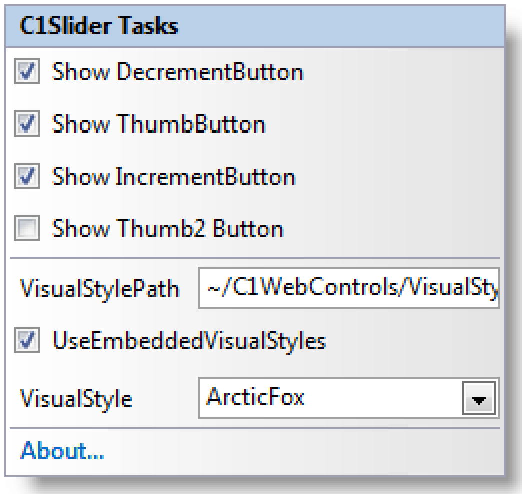
The following sections describe how to use **C1Slider** 's design-time environment to configure the C1Slider control.

C1Slider Smart Tag

In Visual Studio, the **C1Slider** control includes a smart tag. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in **C1Slider**.

The **C1Slider** control provides quick and easy access to common properties through its smart tag.

To access the **C1Slider Tasks** menu, click on the smart tag in the upper-right corner of the **C1Slider** control. This will open the **C1Slider Tasks** menu.



VisualStylePath: ~/C1WebControls/VisualSty

UseEmbeddedVisualStyles: ✓

VisualStyle: ArcticFox

The **C1Slider Tasks** menu operates as follows:

- **Show DecrementButton**

  When the **Show DecrementButton** check box is checked (default) the ShowDecrementButton property is set to **True** and the **Decrement** button is visible on the C1Slider control. Uncheck the **Show DecrementButton** check box to set the ShowDecrementButton property to **False**.

- **Show ThumbButton**

  When the **Show ThumbButton** check box is checked (default) the ShowThumbButton property is set to **True** and the slider's thumb is visible on the C1Slider control. Uncheck the **Show ThumbButton** check box to set the ShowThumbButton property to **False**.

- **Show IncrementButton**

  When the **Show IncrementButton** check box is checked (default) the ShowIncrementButton property is set to **True** and the **Increment** button is visible on the C1Slider control. Uncheck the **Show IncrementButton** check box to set the ShowIncrementButton property to **False**.

- **Show Thumb2 Button**

  Check the **Show Thumb2 Button** check box to set the ShowThumbButton2 property to **True** and show a second thumb button on the C1Slider control. When the **Show Thumb2 Button** check box is cleared the ShowThumbButton2 property is set to **False** (default) and the second thumb button is not shown.

- **VisualStylePath**

  The **VisualStylePath** property specifies the location of the visual styles used for the control. By default, embedded visual styles are located in **~/C1WebControls/VisualStyles**. If you create a custom style, add it to this location **~/VisualStyles/StyleName/C1Slider/styles.css**, set the **VisualStylePath** property to **~/VisualStyles**, and set the **VisualStyle** property to **StyleName** (assuming that **StyleName** is the name used to define the style in the style.css file). Uncheck the **UseEmbeddedVisualStyles** property.

- **UseEmbeddedVisualStyles**

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

- **Visual Style**

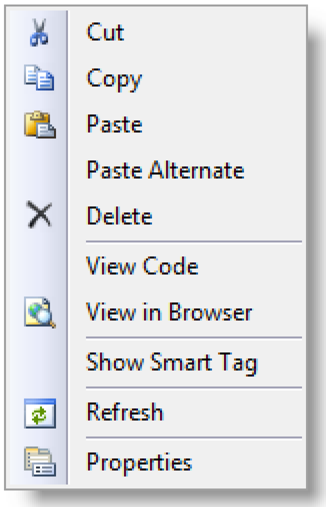
  Clicking the **Visual Style** drop-down box allows you to select from various built-in visual schemes. For more information about available visual styles, see Visual Styles (page 31).

- **About**

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

C1Slider Context Menu

Right-click anywhere on the list to display the C1Slider context menu, which is a context menu that Visual Studio provides for all .NET controls:



The context menu commands operate as follows:

- **Show Smart Tag**

  Clicking this item shows the **C1Slider Tasks** menu. For more information on how to use the smart tag and available features in the Tasks menu, see C1Slider Smart Tag (page 28).

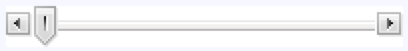
Slider for ASP.NET AJAX Appearance

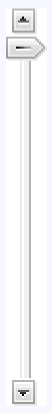
There are several options for customizing the appearance and layout of the C1Slider control. The following sections describe how to change the appearance of the control through built-in Visual Styles as well as how to customize other elements of the C1Slider control.

Orientation

C1Slider includes the ability to orient the control either horizontally or vertically using the Orientation property. By default the control initially appears with a horizontal page orientation when placed on the page. You can easily change the orientation from the Properties window, in Source view, and in code using the Orientation property.

C1Slider includes the following orientations:

Orientation	**Preview**
Horizontal (default)	

Vertical	

Slider Direction

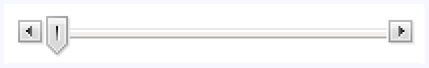
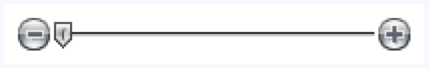
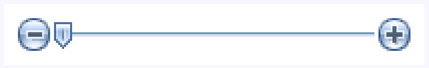
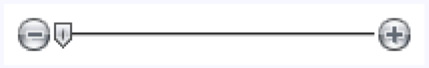
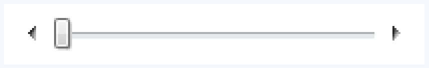
You can now reverse the slider's direction using the Reverse property. The Reverse property can be used to swap the meaning of the minimum and maximum value endpoints of the slider.

For example, in a vertical slider the highest value would appear at the bottom of the slider by default. If you were trying to set up a slider control with the appearance of a typical thermometer and wanted a 0 value at the bottom and 100 value at the top, you could set the Reverse property to **True**. Setting Reverse to **True** would increase the value as the slider thumb was moved from the bottom of the control to the top.

Visual Styles

C1Slider includes Visual Styles allowing you to easily change the control's appearance. The control includes several built-in visual styles, including Vista and Office 2007 styles, to quickly style your application. You can easily change Visual Styles from the **C1Slider Tasks** menu, the Properties window, in Source view, and in code using the VisualStyle property. For more information on changing Visual Styles see the Setting the Visual Style (page 39) topic.

C1Slider includes the following built-in styles:

Visual Style	**Preview**
ArcticFox (default)	
Office2007Black	
Office2007Blue	
Office2007Silver	
Vista	

Custom Visual Styles

If you choose to completely customize the appearance of the C1Slider you may not wish to use any of the available built-in Visual Styles. In that case, you can create your own visual style using CSS. To create your own visual style, you will need to do three things: add a custom style sheet to your project, set the **UseEmbeddedVisualStyles** property to **False**, and set the VisualStylePath and VisualStyle properties. For an example of creating a custom visual style, see Customizing the Slider's Appearance (page 44).

Step 1: Add a Custom Style Sheet

In order to create your own custom style, you must add a style sheet to your project. The VisualStylePath property determines the location of your visual style. The default directory and file name that the VisualStylePath property looks in for custom visual styles is: **~/VisualStyles/StyleName/C1Slider/styles.css** (replacing "StyleName" with the name of each style).

Step 2: Set the UseEmbeddedVisualStyles Property

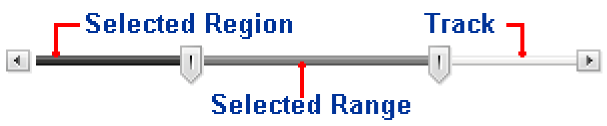
The **UseEmbeddedVisualStyles** property allows you to override built-in visual styles with your own custom appearance. By default C1Slider.**UseEmbeddedVisualStyles** property is **True** and Visual Styles are used. Any customizations you make while using Visual Styles will simply set specific elements in the control's appearance on top of the current Visual Style. To start customizing the control's appearance from scratch set C1Slider.**UseEmbeddedVisualStyles** to **False** and set your own styles.

Step 3: Set the VisualStylePath and VisualStyle Properties

Once you have created a custom visual style and set the **UseEmbeddedVisualStyles** to **False** you must assign the C1Slider control's the VisualStylePath and VisualStyle properties to the visual style directory and the visual style name that you are using. For example, if you add your style sheet to the following location **~/VisualStyles/StyleName/C1Slider/styles.css** the VisualStylePath should be set to **~/VisualStyles** and the VisualStyle property should be set to **StyleName** (assuming that **StyleName** is the name used to define the style in the **styles.css** file.

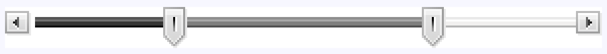
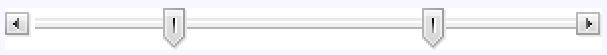
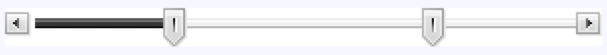
Slider Track Fill

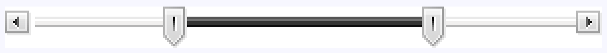
You can now customize the appearance of the slider's track by setting the TrackFill property. The TrackFill property can be used to indicate values chosen on the slider track. By default the TrackFill property is set to **All** and both the selected region and range are visible on the slider track:



The selected region area of the slider track visually indicates the difference between the MinimumValue and the current selected Value. The selected range of the track visually indicates the difference between the selected Value and the selected Value2.

You can set the TrackFill property to various options. These the listed in the table below:

Setting	Image	Description
All		The track fill indicates both the difference between the MinimumValue and the selected Value properties and the range difference between the Value and the selected Value2.
None		The track fill does not indicate a selected range or region and appears uniform in appearance.
MainValue		The track fill indicates only the difference between the MinimumValue and the selected Value.

Range		The track fill indicates only the range difference between the selected Value and the selected Value2.

Slider for ASP.NET AJAX Animations

There are several options for customizing the way that the thumb of the C1Slider control appears to move on the slider track. The following sections describe these animation transitions and animation duration.

C1Slider Transitions

C1Slider includes thirty-one built-in animation transition options that allow you to customize how animation effects are transitioned in the C1Slider control. You can change how the thumb moves when the slider track is clicked by setting the Easing property. By default the Easing property is set to **EaseLinear** and the control moves with a smooth linear transition effect.

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.

Transition Duration

You can set how long each C1Slider animation effect takes by using the AnimationDuration property. The unit of time used for specifying animation effect duration is in milliseconds, and the default setting for the AnimationDuration property is **100** milliseconds (one-tenth of a second). Increase this value for longer animation effect, and decrease this number for a shorter animation effect.

The AnimationDuration does not change if the distance for movement changes. If the AnimationDuration property is set to **100**, it will take 100 milliseconds for the slider thumb to move on the track no matter if it's moving 1 pixel or 500 pixels. If you've increased the control's Length, you may want to increase the transition animation's duration, if you've decreased the Length you may want to decrease the transition animation's duration. You may also wish to increase the transition animation's duration is you have increased the Step property.

Note that if the AnimationDuration property is set to **0**, no animation will be used. To disable the slider's animation, simply set AnimationDuration to **0** in code, Source view, or in the Properties window.

Slider for ASP.NET AJAX Client-Side Functionality

C1Slider includes a robust client-side object model, where a majority of server-side properties can be set on the client-side and client-side events are described in the properties window.

When a C1Slider control is rendered, an instance of the client side slider will be created automatically. This means that you can enjoy the convenience of accessing properties and methods of the C1Slider control without having to postback to the server.

For example, suppose a C1Slider control with name **C1Slider1** is hosted on Web page; when the page is rendered, a corresponding client slider object will be created. Use the following syntax to get the client object:

```
$get("C1Slider1").control
```

OR

```
$find("C1Slider1")
```

By using **C1Slider**'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).

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

- Visual Basic
```
Dim s As String = C1Slider1.ToolTip
C1Slider1.ToolTip = s
```
- C#
```
string s = C1Slider1.ToolTip;
C1Slider1.ToolTip = s;
```
- JavaScript
```
var dialog = $get("C1Slider1").control;
var s = slider.get_toolTip();
slider.set_toolTip(s);
```

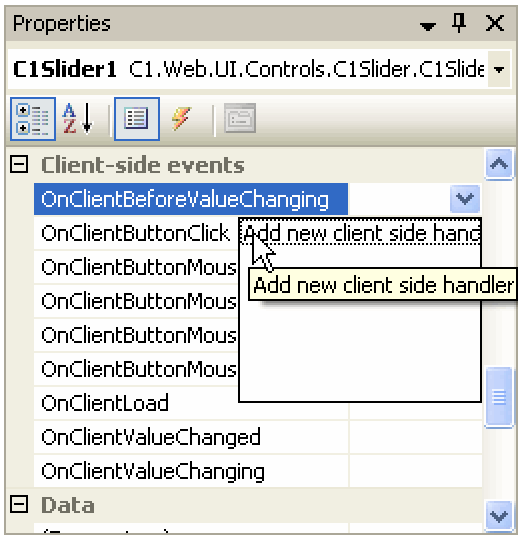
For an example of setting the Length property at run time, see Setting the Slider's Length at Run Time (page 43).

Client-Side Methods

C1Slider includes a rich client-side object model in which several properties can be set on the client side. For information about these client-side methods and what properties can be set on the client side, see the C1Slider Reference.

Client-Side Events

C1Slider includes a rich client-side object model in which includes several client-side events. You can access these client-side events from the 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
OnClientBeforeValueChanging	Fires on the client side before the value of the slider is changing.
OnClientButtonClick	Fires on the client side when a slider button is clicked.
OnClientButtonMouseDown	Fires on client side when the mouse is down on a slider's button.
OnClientButtonMouseOut	Fires on the client side when the user moves the mouse pointer outside the boundaries of a button.
OnClientButtonMouseOver	Fires on the client side when the mouse is over a button.
OnClientButtonMouseUp	Fires on the client side when the mouse is released from a button.
OnClientLoad	Fires on the client side when the slider is initialized.
OnClientValueChanged	Fires on the client side when the value has been changed.
OnClientValueChanging	Fires on the client side when the value is changing.

Slider 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**.

C# Samples

The following pages within the **ControlExplorer** sample installed with **ComponentOne Studio for ASP.NET AJAX** detail the C1Slider control's functionality:

Sample	Description
Animation	Demonstrates various animation effects possible with the **C1Slider** control.
Equalizer	Displays **C1Slider** controls as a sound equalizer system.
Range	Demonstrates adding a second thumb button to the **C1Slider** control to create a range slider with two values.
VisualStyles	Displays the **C1Slider** control's built-in visual styles.
Volume	Displays a **C1Slider** control used as a volume control.
Zoomer	Demonstrates a **C1Slider** control being used to zoom in and out of an image.

Slider for ASP.NET AJAX Task-Based Help

The task-based help assumes that you are familiar with programming in ASP.NET and know how to use controls in general. By following the steps outlined in the help, you will be able to create projects demonstrating a variety of **Slider for ASP.NET AJAX**'s features, and get a good sense of what the C1Slider control can do.

Each topic provides a solution for specific tasks using the C1Slider control. Each task-based help topic also assumes that you have created a new ASP.NET AJAX-Enabled project. For additional information on this topic, see Creating an AJAX-Enabled ASP.NET Project (page 13).

Note: Slider for ASP.NET AJAX requires Microsoft ASP.NET AJAX Extensions installed and a **ScriptManager** on the page before the C1Slider control is placed on the page. You must create an ASP.NET AJAX-Enabled Project so that the **ScriptManager** and Microsoft AJAX Extensions are included on the page. For more information, see Creating an AJAX-Enabled ASP.NET Project (page 13). For more information about Microsoft ASP.NET AJAX Extensions, see http://ajax.asp.net/. For information about the **ScriptManager**, see MSDN.

Creating a C1Slider in Code

Creating a C1Slider control in code is fairly simple. In the following steps you'll add a **PlaceHolder** control to the page, add an import statement, add and customize the C1Slider, and add the control to the **PlaceHolder**.

Complete the following steps:

1. In Design view, navigate to the Visual Studio Toolbox and add a **PlaceHolder** control to your page.

2. Double-click the page to create the **Page_Load** event and switch to Code view.
3. Add the following statement to the top of the Code Editor to import the appropriate namespace:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1Slider
     ```
   - C#
     ```
     using C1.Web.UI.Controls.C1Slider;
     ```
4. Add the following code to the **Page_Load** event to create and customize the **C1Slider** control.
   - Visual Basic
     ```
     ' Create a new C1Slider.
     Dim slide As New C1Slider
     ' Add the C1Slider to the PlaceHolder control.
     PlaceHolder1.Controls.Add(slide)
     ' Set the control's size and appearance.
     slide.MinimumValue = 1
     slide.MaximumValue = 10
     slide.Value = 5
     slide.VisualStyle = "ArcticFox"
     slide.Width = 250
     ```
   - C#
     ```
     // Create a new C1Slider.
     C1Slider slide = new C1Slider();
     // Add the C1Slider to the PlaceHolder control.
     PlaceHolder1.Controls.Add(slide);
     // Set the control's size and appearance.
     slide.MinimumValue = 1;
     slide.MaximumValue = 10;
     slide.Value = 5;
     slide.VisualStyle = "ArcticFox";
     slide.Width = 250;
     ```

Setting Post Back

You can easily set whether or not C1Slider automatically posts back to the server by using the AutoPostBack property. By default the AutoPostBack property is set to **False** and the **C1Slider** control does not automatically post back to the server; to allow **C1Slider** to post back to the server, set AutoPostBack to **True**.

In Source View

In Source view add `AutoPostBack="True"` to the `<cc1:C1Slider>` tag so it appears similar to the following:

```
<cc1:C1Slider ID="C1Slider1" runat="server" AutoPostBack="True" />
```

In Design View

In Design view select the **C1Slider** control and in the Properties window set the AutoPostBack property to **True**.

In Code

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

- Visual Basic
  ```
  Me.C1Slider1.AutoPostBack = True
  ```
- C#
  ```
  this.C1Slider1.AutoPostBack = true;
  ```

Adding Keyboard Support

The C1Slider control lets you easily add keyboard accessibility to the control. You can use C1Slider.**AccessKey** property to set how the user navigates to the control and through your user interface. At run time users can use a key combination to bring the C1Slider control into focus and can then change the Value of the control using the left and right arrow keys (if Orientation is set to **Horizontal**) or up and down arrow keys (if Orientation is set to **Vertical**).

In the following examples you'll set the C1Slider.**AccessKey** property to **s** so that pressing the ALT+S key combination at run time brings the C1Slider control into focus. Once the control is in focus, the user can use the arrow keys to change the control's Value.

In Source View

In Source view add `AccessKey="s"` to the `<cc1:C1Slider>` tag so it appears similar to the following:

```
<cc1:C1Slider ID="C1Slider1" runat="server" AccessKey="s" />
```

In Design View

In Design view select the **C1Slider** control and in the Properties window set the C1Slider.**AccessKey** property to "s".

In Code

Add the following code to the **Page_Load** event to set the C1Slider.**AccessKey** property to **s**:

- Visual Basic
  ```
  Me.C1Slider1.AccessKey = "s"
  ```
- C#
  ```
  this.C1Slider1.AccessKey = "s";
  ```

Setting the Visual Style

The control includes several built-in visual styles, including Vista and Office 2007 styles, to style your application. For more information about available styles, see Visual Styles (page 31). You can easily change Visual Styles in Source view, in Design view, and in code using the VisualStyle property. The examples below set the VisualStyle to **Vista**.

Note that when you change the VisualStyle property the VisualStylePath and **C1Slider.CssClass** properties also change.

In Source View

In Source view add `VisualStyle="Vista"` to the `<cc1:C1Slider>` tag so it appears similar to the following:

```
<cc1:C1Slider ID="C1Slider1" runat="server" VisualStyle="Vista" />
```

From the Tasks menu

You can select a Visual Style to apply from the **C1Slider Tasks** menu:

1. Click on the **C1Slider**'s smart tag to open the **C1Slider Tasks** menu.
2. Click the **Visual Style** drop-down arrow.
3. In the **Visual Style** drop-down box select a Visual Style, for example **Vista**.

   The Visual Style you chose will be applied to the C1Slider control.

From the Properties window

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

1. Click on the **C1Slider** control to select it.

2. Navigate to the Properties window and to the VisualStyle property.
3. Type in a style to apply, for example "Vista".

   The Visual Style you choose will be applied to the C1Slider control.

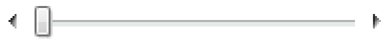
In Code

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

- Visual Basic
```
Me.C1Slider1.VisualStyle = "Vista"
```
- C#
```
this.C1Slider1.VisualStyle = "Vista";
```

Run your project and observe:

The **C1Slider** will appear set to the **Vista** visual style:



Adding a Border to C1Slider

You may want to add a border around the C1Slider control to increase the control's accessibility. A border allows the slider's track to be more easily distinguished when viewed in high contrast mode. The border's appearance can be set in Source view, Design view, and code using the control's **BorderColor**, **BorderStyle**, and **BorderWidth** properties. In the following example you'll add a thin, black line around the C1Slider control.

In Source View

In Source view set the control's **BorderColor**, **BorderStyle**, and **BorderWidth** properties by adding text to the `<cc1:C1Slider>` tag so it appears similar to the following:
```
<cc1:C1Slider ID="C1Slider1" runat="server" BorderColor="Black"
BorderStyle="Solid" BorderWidth="1px" />
```

In Design View

Complete the following to set the control's **BorderColor**, **BorderStyle**, and **BorderWidth** properties in the Properties window:

1. Click on the **C1Slider** control to select it.
2. Navigate to the Properties window and, if needed, expand the **Appearance** node.
3. Set the following properties:
   - Set the **BorderColor** property to **Black**.
   - Set the **BorderStyle** property to **Solid**.
   - Set the **BorderWidth** property to **1**.

In Code

Complete the following to set the control's **BorderColor**, **BorderStyle**, and **BorderWidth** properties in code:

1. Add the following imports/using statements to your project:
   - Visual Basic
```
Imports System.Web.UI.WebControls
Imports System.Drawing
Imports C1.Web.UI.Controls.C1Slider
```

- C#
```
using System.Web.UI.WebControls;
using System.Drawing;
using C1.Web.UI.Controls.C1Slider;
```

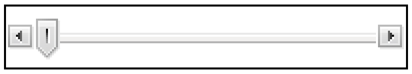
2. Add the following code to the **Page_Load** event to set the control's **BorderColor**, **BorderStyle**, and **BorderWidth** properties:

   - Visual Basic
   ```
   Me.C1Slider1.BorderColor = Color.Black
   Me.C1Slider1.BorderStyle = BorderStyle.Solid
   Me.C1Slider1.BorderWidth = 1
   ```

   - C#
   ```
   this.C1Slider1.BorderColor = Color.Black;
   this.C1Slider1.BorderStyle = BorderStyle.Solid;
   this.C1Slider1.BorderWidth = 1;
   ```

Run your project and observe:

The C1Slider control will appear with a solid, black, 1 pixel border:



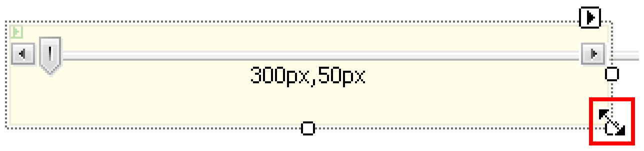
Resizing the Control

You can easily change the height and width on the **C1Slider** by setting the Height and Width properties. By default the height of the control is set to **30px** and the width of the control is set to **200px**. You can easily change the control's height and width in Design view, Source view, from the Properties window, or in code. Note that setting the Width property also sets the Length property to the same value.

In Design View

To resize the **C1Slider** control in Design view, click the bottom-right corner of the control and perform a drag-and-drop operation to set the control's size.

For example, in the following image the control has been resized to 300 pixels wide and 50 pixels tall:



In Source View

In Source view add `Height="50px" Width="300px"` (or substitute values you choose) in the `<cc1:C1Slider>` tag so that it looks similar to the following:

```
<cc1:C1Slider ID="C1Slider1" runat="server" AutoPostBack="True" Height="50px"
Width="300px" />
```

The above will resize the control to 50 pixels tall and 300 pixels wide.

In the Properties Window

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

1. Click on the **C1Slider** to select it.

2. Navigate to the Properties window and if needed expand the **Layout** node to locate the Height and Width properties.
3. Next to **Height**, enter the size you want the control's height to be, for example "50px".
4. Next to **Width**, replace "200px" with the size you want the control's width to be, for example "300px".
5. Press the ENTER key or click outside of the Properties window for the height and width you set to be applied to the **C1Slider** control.

In Code

Add the following code to the **Page_Load** event to set the Height to 50 pixels and the Width to 300 pixels:

- Visual Basic
```
Me.C1Slider1.Height = 50
Me.C1Slider1.Width = 300
```
- C#
```
this.C1Slider1.Height = 50;
this.C1Slider1.Width = 300;
```

Adding ToolTips to C1Slider

You can use ToolTips to add information or instructions to **C1Slider**. The **C1Slider** control includes properties to add ToolTips to the control, the **Increment** and **Decrement** buttons, and the slider's thumb. You can set these ToolTips using the ToolTip, IncrementToolTip, DecrementToolTip, and ThumbToolTip properties in Source view, Design view, or in code.

In Source View

In Source view, add text to the `<cc1:C1Slider>` tag so it appears similar to the following:
```
<cc1:C1Slider ID="C1Slider1" runat="server" ToolTip="I am a slider control."
IncrementToolTip="Push this button to increase my value."
DecrementToolTip="Push this button to decrease my value." ThumbToolTip="Move
this button to change my value." />
```
The ToolTip, IncrementToolTip, DecrementToolTip, and ThumbToolTip properties will be set.

In Design View

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

1. Click on the **C1Slider** control to select it.
2. Navigate to the Properties window and, if needed, expand the **Behavior** node.
3. Set the following properties in the Properties window to set the ToolTip, IncrementToolTip, DecrementToolTip, and ThumbToolTip properties:
   - Set the **ToolTip** property to "I am a slider control."
   - Set the **IncrementToolTip** property to "Push this button to increase my value."
   - Set the **DecrementToolTip** property to "Push this button to decrease my value."
   - Set the **ThumbToolTip** property to "Move this button to change my value."

In Code

Add the following code to the **Page_Load** event to ToolTip, IncrementToolTip, DecrementToolTip, and ThumbToolTip properties:

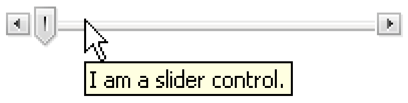
- Visual Basic
```
Me.C1Slider1.ToolTip = "I am a slider control."
Me.C1Slider1.IncrementToolTip = "Push this button to increase my value."
Me.C1Slider1.DecrementToolTip = "Push this button to decrease my value."
Me.C1Slider1.ThumbToolTip = "Move this button to change my value."
```

- C#

```
this.C1Slider1.ToolTip = "I am a slider control.";
this.C1Slider1.IncrementToolTip = "Push this button to increase my
value.";
this.C1Slider1.DecrementToolTip = "Push this button to decrease my
value.";
this.C1Slider1.ThumbToolTip = "Move this button to change my value.";
```

Run your project and observe:

Tooltips will now appear at run time when the mouse hovers over the control, the **Increment** button, the **Decrement** button, and the slider's thumb:



Setting the Slider's Length at Run Time

You can easily change the length of the **C1Slider** control at run time by setting the Length property. By default the length of the control is set to **200px**. In this topic you'll add a text box and button to change the control's length at run time. For information about changing the control's height and width in Design view, Source view, In Design view, and in code, instead, see Resizing the Control.

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

1. Navigate to the Visual Studio Toolbox and double-click the **C1Slider** icon to add the control to your page.
2. Click on the **Source** tab to switch to Source view, and note that the body of the page looks similar to the following:

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
    </div>
        <cc1:C1Slider ID="C1Slider1" runat="server" Width="200px" />
    </form>
</body>
```

3. Add the following tags just below the `<cc1:C1Slider>` tag to add a button and text box to set the Length property:

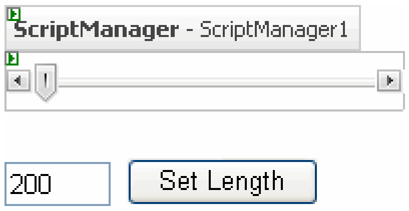
```
<br /><br />
<input type="text" value="200" id="textLength" size="3" />&nbsp;
<input type="button" id="btnLength" onclick="SetLength()" value="Set
Length" />
```

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

```
<script language="javascript" type="text/javascript">
function SetLength()
{
var slider = Sys.Application.findComponent("<%=C1Slider1.ClientID%>");
slider.set_length(document.getElementById('textLength').value);
}
</script>
```

This function will change the **C1Slider**'s Length property when the button on the page is clicked.

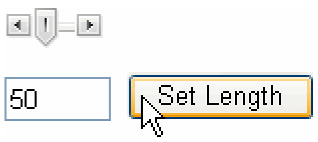
5. Switch to Design view and note that the page now looks similar to the following:



Run the project and observe:

1. Enter **50** in the text box and click the **Set Length** button.

   The control resizes and now appears similar to the following:



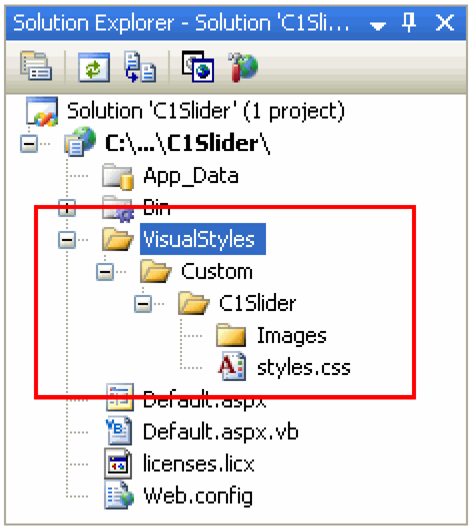
2. Enter another value, for example **500** in the text box and click the **Set Length** button again. The control will resize again.

Customizing the Slider's Appearance

You can customize the C1Slider control appearance with your own images and styles by creating a custom visual style. All of **C1Slider**'s style settings are controlled by style sheets. In the following example you'll create a custom style sheet to change the appearance of a horizontal slider control. For additional information, see Custom Visual Styles (page 31).

1. Create a new ASP.NET AJAX-Enabled Web Site. For more information, see Creating an AJAX-Enabled ASP.NET Project (page 13).
2. In Design view, navigate to the Visual Studio Toolbox and double-click the **C1Slider** icon to add the control to your page.
3. In the Solution Explorer, right-click the Web site and select **New Folder** to add a new folder to your project. Name the folder **VisualStyles**.
4. Right-click the **VisualStyles** folder and select **New Folder** to add a new folder. Name the folder **Custom**.
5. Right-click the **Custom** folder and select **New Folder** to add a new folder. Name the folder **C1Slider**.
6. Right-click the **C1Slider** folder and select **New Folder** to add a new folder. Name the folder **Images**.
7. Right-click the **C1Slider** folder and select **Add New Item**.
8. In the **Add New Item** dialog box, select **Style Sheet**. Name the file **styles.css** and select **Add** to close the dialog box and add the style sheet to your project.

   Note that your file structure should look similar to the following image:



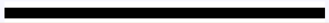
This structure, **~/VisualStyles/StyleName/C1Slider/styles.css**, is the default file structure and file name that the VisualStylePath property will look in for the VisualStyle.

9. Double-click the **styles.css** item in the Solution Explorer if is not open. Add the following text to the style sheet to create a custom horizontal visual style:

```
/*--Set the C1Slider control's positioning.--*/
.C1Slider_Custom-C1Horizontal
{
   -moz-user-select: none; font-size:1px; float:left; text-align:left;
   width:200px; height:30px; position:relative;overflow:hidden;
}
/*--Set the decrement button's appearance.--*/
.C1Slider_Custom-C1Horizontal .C1DecButton
{
   background-image:url('Images/dec_btn.png'); background-position:left
center;
   position:absolute;width:15px; height:12px; left:0px;
top:9px;display:block;
}
/*--Set the track's appearance.--*/
.C1Slider_Custom-C1Horizontal .C1Track
{
   background-image:url('Images/track.png');
   background-position:center; background-repeat:repeat-x;
   height:10px;display:block; position:absolute; left:15px; top:10px;
width:170px;
}
/*--Set the increment button's appearance.--*/
.C1Slider_Custom-C1Horizontal .C1IncButton
{
   background-image:url('Images/inc_btn.png'); background-position:left
center;
   position:absolute; width:15px; height:12px; right:0px; top:9px;
}
/*--Set the track selected region's appearance.--*/
.C1Slider_Custom-C1Horizontal .C1SelectedRegion
{
   background-image:url('Images/selected.png');
   background-repeat:repeat-x; background-position:center;
   position:absolute;  width:0px; height:10px;
}
/*--Set the thumb button's appearance.--*/
.C1Slider_Custom-C1Horizontal .C1Thumb
{
   background-image:url('Images/thumb.png'); background-position:left
center;
   background-repeat:no-repeat;
   position:absolute; top:-3px; width:12px; height:20px; left:0px;
}
```

10. Right-click the **Images** folder in the Solution Explorer and select **Add Existing Item** to locate and add images for the slider track and selected region and increment, decrement, and thumb buttons.

    In the style sheet above, the following images were used:

Image	Name	Size
	dec_btn.png	20 x 18 pixels
	track.png	160 x 10 pixels
	inc_btn.png	20 x 18 pixels
	selected.png	1 x 10 pixels
	thumb.png	9 x 13 pixels

    You can add your own images, or use the images above. To use the above images, right-click each image in the table above and select **Copy**. Paste the image into a paint program of your choice and save it with the name and dimensions indicated above.

11. Select **Default.aspx** to return to your page.
12. Click the **C1Slider** control to select it and, in the Properties window, set the following properties:
    - Set the **UseEmbeddedVisualStyles** property to **False**.
    - Set the VisualStylePath property to "~/VisualStyles".
    - Set the VisualStyle property to "Custom".

    These settings will set the C1Slider control to not use the built-in visual styles, to look in the directory you created for visual styles to use, and to use the visual style you created (named "Custom") as its current visual style.

Run your application and observe:

The slider will look similar to the following image: