ComponentOne

MultiPage for ASP.NET AJAX

Corporate Headquarters
ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com
Web site: http://www.componentone.com

Sales
E-mail: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

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 MultiPage for ASP.NET AJAX Overview 1
- What's New in MultiPage for ASP.NET AJAX 1
- Revision History 1
- What's New in 2010 v1 1
- What's New in 2009 v3 2
- Installing MultiPage for ASP.NET AJAX 2
- MultiPage for ASP.NET AJAX Setup Files 2
- System Requirements 2
- Uninstalling MultiPage for ASP.NET AJAX 3
- Deploying your Application in a Medium Trust Environment 3
- End-User License Agreement 6
- Licensing FAQs 6
- What is Licensing? 6
- How does Licensing Work? 7
- Common Scenarios 7
- Troubleshooting 9
- Technical Support 10
- Redistributable Files 11
- About This Documentation 11
- Namespaces 12
- Creating an AJAX-Enabled ASP.NET Project 13
- Adding the ASP.NET Components to a Project 15

Key Features 17

MultiPage for ASP.NET AJAX Quick Start 21
- Step 1 of 3: Adding C1MultiPage to the Page 21
- Step 2 of 3: Working with the C1MultiPage Designer Form 21
- Step 3 of 3: Adding Content to the C1MultiPage Control 24

Top Tips 25

C1MultiPage Design-Time Support 26
- C1MultiPage Smart Tag 26
- C1MultiPage Context Menu 27
- C1MultiPage Designer Form 28

C1MultiPage Elements 30

C1MultiPage Appearance 31
- Visual Styles 31
- Custom Visual Styles 32
- CSS Styling 33

C1MultiPage Behavior 33
- C1MultiPage AutoPlay Feature 34
- C1MultiPage Animation and Transition Effects 34
- C1MultiPage Animation Effects 34
- Animation Effect Duration 34
- C1MultiPage Transition Effects 35
- ToolTips 37
- Selected Index 37

Client-Side Functionality 37

Client-Side Properties 38
Client-Side Methods 38

MultiPage for ASP.NET AJAX Samples 38

MultiPage for ASP.NET AJAX Task-Based Help 39
Adding a C1MultiPage in Code 39
Adding and Manipulating Page Content 40
Adding Arbitrary Controls to a Page 40
Adding Text to a Page 43
Displaying External Content in a Page 45
Adding Navigation to a C1MultiPage Control 46
Using the Toolbar for Navigation 46
Using a C1TabStrip for Navigation 49
Applying CSS Styles to C1MultiPage 51
Adding a Border to the C1MultiPage Control 51
Adding Font Styles to a Page 53
Adding a Gradient to a Page 55
Client-Side Tasks for C1MultiPage 57
Using Arbitrary Buttons for Navigation 57
Selecting C1PageViews by ID 59
Creating Play and Stop Buttons 61
Setting the Duration of Animations at Run-Time 62
Customizing the Appearance of C1MultiPage 63
Adding Custom Visual Styles 63
Changing the Visual Style 68
Setting C1MultiPage Behaviors 69
Changing C1MultiPage's Selected Index 69
Autoplaying the Pages of a C1MultiPage 71
Displaying ToolTips for Pages 73
Loading Pages on Demand 74
Setting Automatic Postback 75
Using Animation Effects 76

ComponentOne MultiPage for ASP.NET AJAX Overview

ComponentOne MultiPage for ASP.NET AJAX enables you to create and group multiple pages of Web-based content, meaning that you can provide users with access to substantial amounts of information without wasting screen space. Use a C1MultiPage control to display text, arbitrary controls, or even external web pages. Change its style to suit your needs by selecting one of our built-in schemes or by creating a custom visual style. Want easy user navigation? Just turn on C1MultiPage"s built-in navigation toolbar or synchronize the C1MultiPage with a **C1TabStrip**. With **MultiPage for ASP.NET AJAX**, you will be creating sophisticated, professional Web applications in minutes.

Getting Started

- Quick Start (page 21)
- Design-Time Support (page 26)
- C1MultiPage Elements (page 30)

What's New in MultiPage for ASP.NET AJAX

No new features were added to **MultiPage for ASP.NET AJAX** in its 2010 v2 release.

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

Revision History

This section provides an overview of the changes that have occurred in **MultiPage for ASP.NET AJAX** over the last few releases.

What's New in 2010 v1

The following new members were added to MultiPage for ASP.NET in its 2010 v2 release.

New Server-Side Members

The following new members were added to the C1MultiPage control's server-side API:

Member	Description
OnClientBeforeSelectedIndexChange property	Handler name of the client **BeforeSelectedIndexChange** event.
OnClientSelectedIndexChanged property	Handler name of the client **SelectedIndexChanged** event.

New Client-Side Members

The following new members were added to the C1MultiPage control's client-side API:

Member	Description

BeforeSelectedIndexChange event	Fires before SelectedIndex changes.
SelectedIndexChanged event	Fires after the SelectedIndex changed.

What's New in 2009 v3

The following members have been added to **MultiPage for ASP.NET AJAX**:

- Added **BeforeSelectedIndexChange** and **SelectedIndexChanged** events to the client object.
- Added **OnClientBeforeSelectedIndexChange** and **OnClientSelectedIndexChanged** properties to the server control

Installing MultiPage for ASP.NET AJAX

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

MultiPage 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 AJAX. This directory contains the following subdirectories:

bin	Contains copies of all binaries (DLLs, Exes) in the ComponentOne Visual Studio ASP.NET package.
H2Help	Contains documentation for Studio for ASP.NET AJAX components.
C1WebUi	Contains files (at least a readme.txt) related to the product.
C1WebUi\VisualStyles	Contains all external file themes.

Samples

Samples for the product are installed in the **ComponentOne Samples** folder by default. The path of the **ComponentOne Samples** directory is slightly different on Windows XP and Windows 7/Vista machines:

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

Windows 7/Vista path: C:\Users\<username>\Documents\ComponentOne Samples

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

Common	Contains support and data files that are used by many of the demo programs.
Studio for ASP.NET\C1WebUi	Contains a readme.txt file and the folders that make up the Control Explorer and other samples.

You can access samples from the **ComponentOne Control Explorer**. To view samples, click the **Start** button and then click **ComponentOne | Studio for ASP.NET | Control Explorer**.

System Requirements

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

Operating Systems:	Windows® 2000

	Windows Server® 2003
	Windows Server 2008
	Windows XP SP2
	Windows Vista™
	Windows 7
Web Server:	Microsoft Internet Information Services (IIS) 5.0 or later
Environments:	.NET Framework 2.0 or later
	Visual Studio 2005 or later
	Internet Explorer 6.0 or later
	Firefox® 2.0 or later
	Safari® 2.0 or later
Disc Drive:	CD or DVD-ROM drive if installing from CD

Uninstalling MultiPage for ASP.NET AJAX

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

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

Deploying your Application in a Medium Trust Environment

Depending on your hosting choice, you may need to deploy your Web site or application in a medium trust environment. Often in a shared hosting environment, medium trust is required. In a medium trust environment several permissions are unavailable or limited, including OleDbPermission, ReflectionPermission, and FileIOPermission. You can configure your Web.config file to enable these permissions.

Note: ComponentOne controls will not work in an environment where reflection is not allowed.

ComponentOne ASP.NET controls include the AllowPartiallyTrustedCallers() assembly attribute and will work under the medium trust level with some changes to the Web.config file. Since this requires some control over the Web.config file, please check with your particular host to determine if they can provide the rights to override these security settings.

Modifying or Editing the Config File

In order to add permissions, you can edit the exiting web_mediumtrust.config file or create a custom policy file based on the medium trust policy. If you modify the existing web_mediumtrust.config file, all Web applications will have the same permissions with the permissions you have added. If you want applications to have different permissions, you can instead create a custom policy based on medium trust.

Edit the Config File

In order to add permissions, you can edit the exiting web_mediumtrust.config file. To edit the exiting web_mediumtrust.config file, complete the following steps:

1. Locate the medium trust policy file web_mediumtrust.config located by default in the %windir%\Microsoft.NET\Framework\{Version}\CONFIG directory.

2. Open the web_mediumtrust.config file.
3. Add the permissions that you want to grant. For examples, see Adding Permissions (page 4).

Create a Custom Policy Based on Medium Trust

In order to add permissions, you can create a custom policy file based on the medium trust policy. To create a custom policy file, complete the following steps:

1. Locate the medium trust policy file web_mediumtrust.config located by default in the %windir%\Microsoft.NET\Framework\{Version}\CONFIG directory.
2. Copy the web_mediumtrust.config file and create a new policy file in the same directory.

   Give the new 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.)

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 derive 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.

Using licensed components in console applications

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

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

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

Using licensed components in Visual C++ applications

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

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

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

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 AJAX 2.x applications, follow these steps:

8. Open the project and go to the Solution Explorer window.
9. Find the licenses.licx file and right-click it.
10. Select the Rebuild Licenses option (this will rebuild the App_Licenses.licx file).
11. 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 AJAX 2.x applications, be sure that the App_Licenses.dll assembly created during development of the application is deployed to the bin application bin directory on the Web server.

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

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

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

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

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

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

Option 2 – Continue to use the components you have.

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

Technical Support

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

Some methods for obtaining technical support include:

- **Online Support via HelpCentral**
  ComponentOne HelpCentral provides customers with a comprehensive set of technical resources in the form of FAQs, samples, Version Release History, Articles, searchable Knowledge Base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.

- **Online Support via our Incident Submission Form**
  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you will immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.
- **Peer-to-Peer Product Forums and Newsgroups**
  ComponentOne peer-to-peer product forums and newsgroups are available to exchange information, tips, and techniques regarding ComponentOne products. ComponentOne sponsors these areas as a forum for users to share information. While ComponentOne does not provide direct support in the forums and newsgroups, we periodically monitor them to ensure accuracy of information and provide comments when appropriate. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.
- **Installation Issues**
  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.
- **Documentation**
  ComponentOne documentation is installed with each of our products and is also available online at HelpCentral. If you have suggestions on how we can improve our documentation, please email the Documentation team. Please note that e-mail sent to the Documentation team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

Note: You must create a ComponentOne Account and register your product with a valid serial number to obtain support using some of the above methods.

Redistributable Files

ComponentOne Studio for ASP.NET AJAX (CTP) 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
- C1.Web.UI.Design.4.dll

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

About This Documentation

Acknowledgements

ComponentOne

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

Corporate Headquarters

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

http://www.componentone.com

ComponentOne Doc-To-Help

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

Namespaces

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

The general namespace for ComponentOne Web products is **C1.Web**. The following code fragment shows how to declare a **C1MultiPage** (which is one of the core **Studio for ASP.NET AJAX** classes) using the fully qualified name for this class:

- Visual Basic
```
Dim MultiPage As C1.Web.UI.Controls.C1MultiPage
```

- C#
```
C1.Web.UI.Controls.C1MultiPage MultiPage;
```

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 C1MultiPage = C1.Web.UI.Controls.C1MultiPage
Imports MyMultiPage = MyProject.Objects.C1MultiPage

Dim wm1 As C1MultiPage
```

```
Dim wm2 As MyMultiPageMenu
```

- C#

```
using C1MultiPage = C1.Web.UI.Controls.C1MultiPage;
using MyMultiPage = MyProject.Objects.C1MultiPage;

 C1MultiPage wm1;
 MyMultiPage 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 MultiPage 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 C1MultiPage 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 http://www.asp.net/AJAX/downloads/archive/
Visual Studio 2005	ASP.NET AJAX Extensions 1.0 Visual Studio update and add-in (2 installs for Web application project support)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  1. From the **File** menu in Microsoft Visual Studio 2005, select **New Project**. The **New Project** dialog box opens.
  2. Under **Project Types**, choose either **Visual Basic Projects** or **Visual C# Projects**. Note that one of these options may be located under **Other Languages**.
  3. Select **ASP.NET AJAX-Enabled Web Application** from the list of **Templates** in the right pane.
  4. Enter a URL for your application in the **Location** field and click **OK**.

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

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

Adding the ASP.NET Components to a Project

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

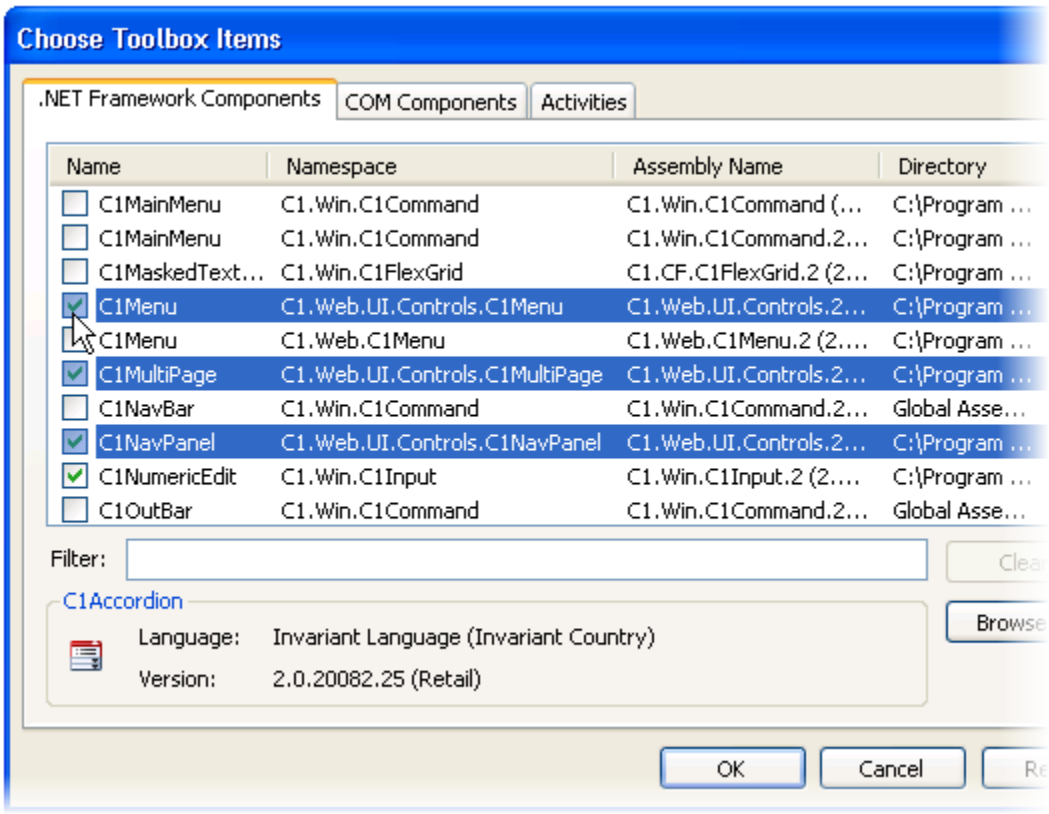
Manually Adding the Studio for ASP.NET AJAX controls to the Toolbox

When you install **ComponentOne Studio for ASP.NET AJAX**, the C1MultiPage component will appear in the Visual Studio Toolbox customization dialog box.

To manually add the Studio for ASP.NET AJAX controls to the Visual Studio Toolbox:

1. Open the Visual Studio IDE (Microsoft Development Environment). Make sure the Toolbox is visible (select Toolbox in the View menu if necessary) and right-click it to open the context menu.
2. To make the Studio for ASP.NET AJAX components appear on their own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, Studio for ASP.NET AJAX, for example.
3. Right-click the tab where the component is to appear and select **Choose Items** from the context menu.

   The **Choose Toolbox Items** dialog box opens.
4. In the dialog box, select the **.NET Framework Components** tab. Sort the list by Namespace (click the **Namespace** column header) and check the check boxes for all components belonging to namespace C1.Web.UI.Controls.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.

Adding a Reference to the Assembly

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

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

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

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

Key Features

C1Multipage includes several unique features, including the following:

- **CSS Styling**

  C1MultiPage includes CSS supported styling so that you can use cascading style sheets to easily style the **C1MultiPage** control to match the design of your current Web site.

- **Built-in Visual Styles**

  Use the built-in styles to quickly change the **C1MultiPage** control's appearance. **C1MultiPage** currently includes five built-in visual styles: Arctic Fox, Office 2007 Blue, Office 2007 Black, Office 2007 Silver, and Vista.

- **Animation Effects**

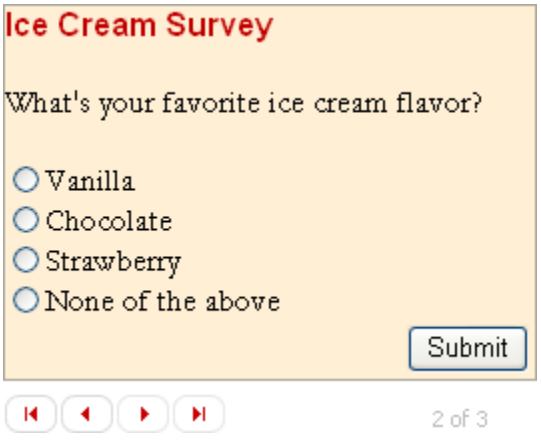
  C1MultiPage includes 10 built-in animated effects and twenty-six transition effects that can be used to transition from one page to another. Easily change transition effects by setting two properties – Animation and Easing. You can also create your own custom effects to apply to the **C1MultiPage** control.

- **C1TabStrip Support**

  You can create a customized tabbed navigation pane by using the **C1MultiPage** control with the **C1TabStripControl**. You can also use other menu and navigation controls with the **C1MultiPage** control.

- **Add Content**

  You can add images, text, and controls – standard and third-party controls – to pages in the **C1MultiPage** control by simply performing a drag-and-drop operation. Customizing a PageView's content is as easy as designing the content of a standard Panel control.

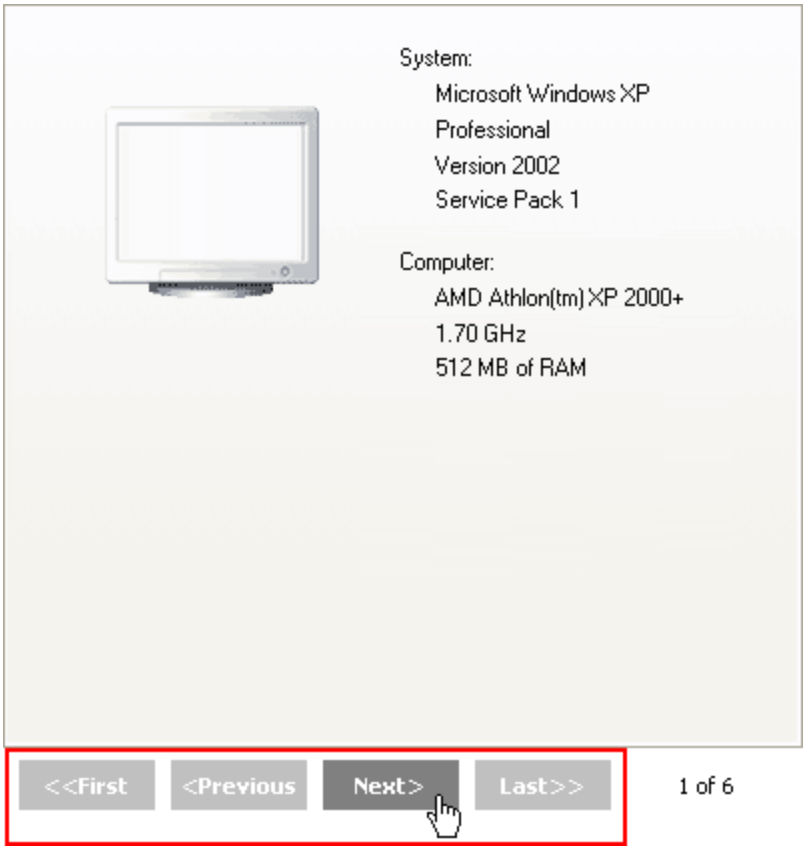


- **AutoPlay Pages**

  By setting a single property you can choose if the **C1MultiPage** control automatically navigates between pages. You can customize auto play as well, including setting the delay between page transitions.

- **Navigation Buttons**

  C1MultiPage can display **Next**, **Previous**, **First**, and **Last** page navigation buttons allowing users to quickly navigate between pages. You can also customize the text and style of these buttons or use the default text and one of the built-in styles.



- **Set Postbacks**

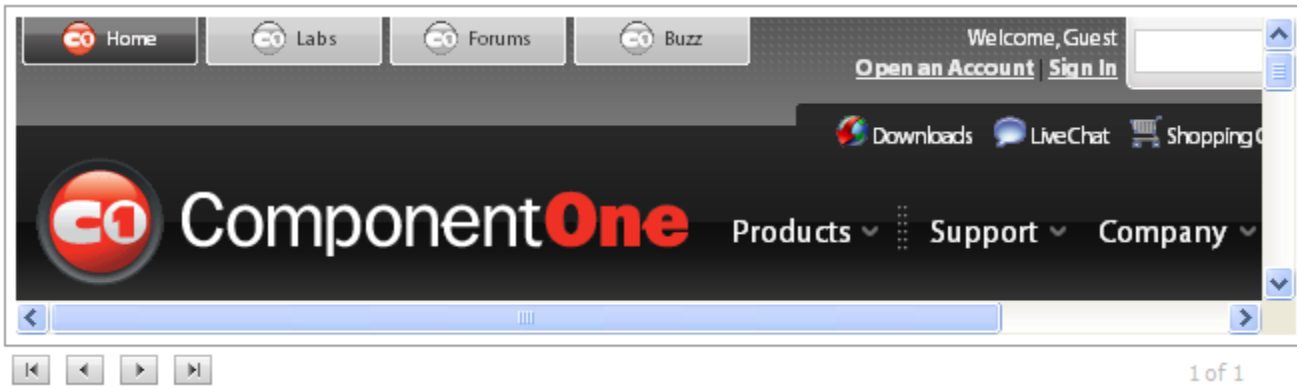
  Use the AutoPostBack property to determine whether **C1MultiPage** should perform a postback to the server when switching between multiple page views.

- **Load On Demand**

  Set **C1MultiPage** to only render the currently selected page on the client side, reducing initial page size and load time. Doing this is as simple as setting two properties – AutoPostBack and LoadOnDemand – to **True**. This feature is particularly useful when hosting the **C1MultiPage** control inside of an **UpdatePanel**.

- **External Content**

  You can show external content in a PageView by using the **ContentUrl** property. This means that you can have the content of another Web page in your project or even the content of a Web site outside of your project appear in a **C1MultiPage** control.



- **Browser Support**

  C1MultiPage includes support for the Internet Explorer (6.0 or later), Firefox (2 or later), and Safari Web browsers.

- **Search Engine Optimization**

  Optimized for search engines, **C1MultiPage** uses semantic lists and `<a>` tags which are recognized and indexed by web crawlers.

- **XHTML Compliant**

  C1MultiPage provides complete XHTML compliance. The output that is generated is fully XHTML 1.1 compliant.

- **Client-Side Object Model**

  The **C1MultiPage** control's client-side object model is exposed so that you can easily customize the control with client-side script.

MultiPage for ASP.NET AJAX Quick Start

In this quick start guide, you will explore the functionality of the C1Multipage control by creating a simple paged area with navigation controls within a Web site.

Step 1 of 3: Adding C1MultiPage to the Page

In this step you will begin by adding the **C1MultiPage** control to the page.

To begin, 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. In the **Website** menu select **Add Reference**. In the **Add Reference** dialog box select or browse to and select the **C1.Web.UI.2** and **C1.Web.Control.2** assemblies and click **OK** to add the references to your project.
3. Click the **Design** tab located below the Document window to switch to Design view.
4. Navigate to the Visual Studio Toolbox and double-click **C1MultiPage** to add the control to the page.

   The Web page will contain the empty **C1MultiPage** control and will appear similar to the following:

Step 2 of 3: Working with the C1MultiPage Designer Form

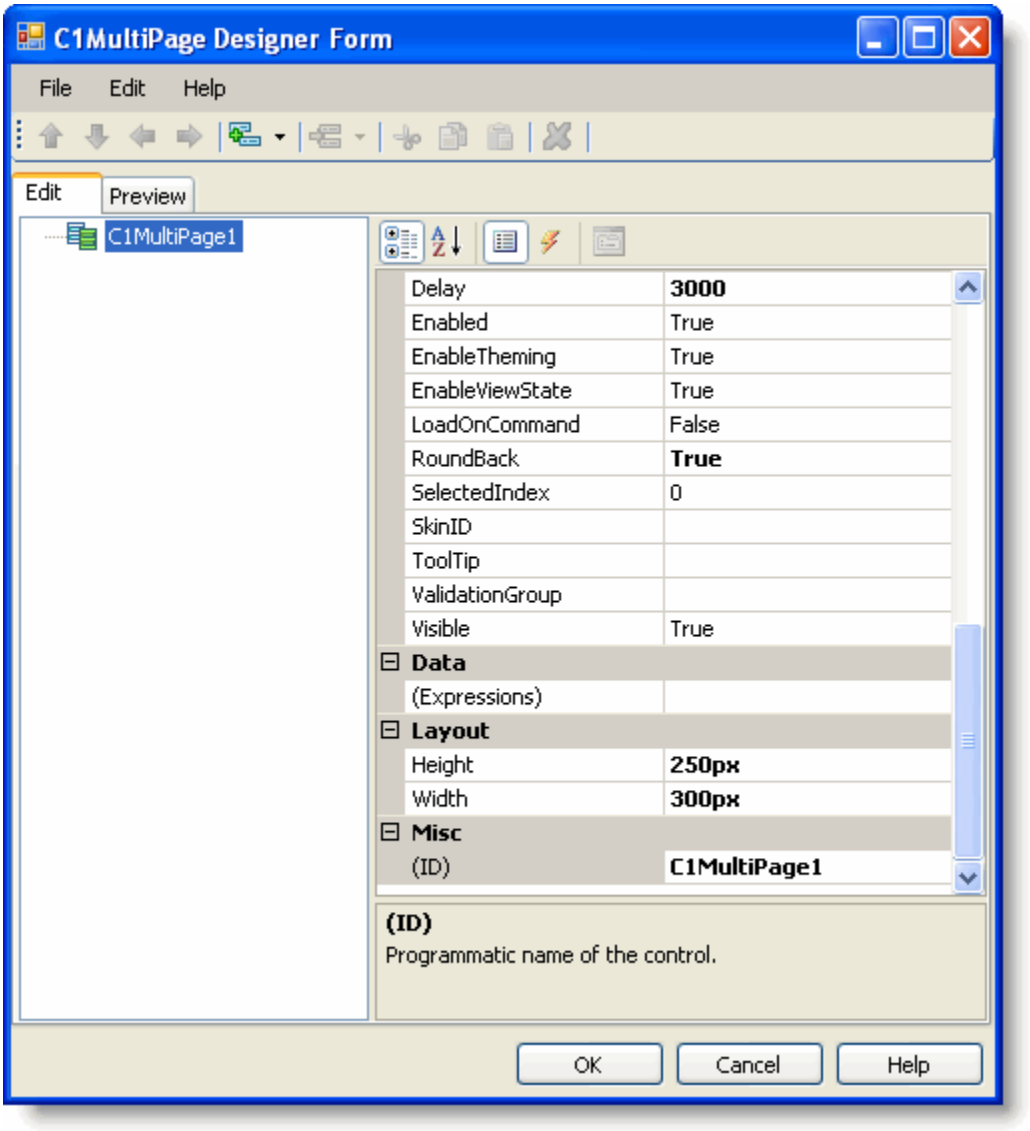
The **C1MultiPage Designer Form** allows you to easily customize the **C1MultiPage** control and each **C1PageView** you choose to include. In this step you will add three **C1PageView**s to the **C1MultiPage** control and change its behavior using the **C1MultiPage Designer Form**. Note that you should complete the steps in the

Adding C1MultiPage to the Page (page 21) topic or create a new Web site project and add a **C1MultiPage** control to the page before completing the steps outlined here.

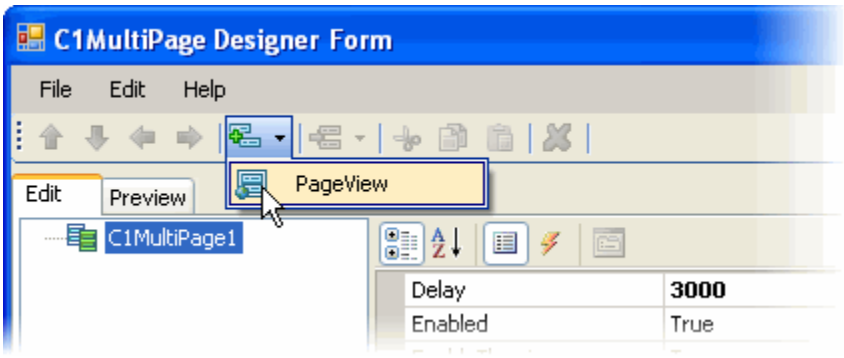
Complete the following steps:

1. Click **C1MultiPage1**'s smart tag to open the **C1Multipage Tasks** menu, and select the **MultiPage Designer**.

   The **C1MultiPage Designer Form** will open.



2. In the **C1MultiPage Designer Form** select **C1MultiPage1**, click the **Add Child Item** button, and select **PageView**. Repeat this step two more times to add a total of three **PageViews** to the **MultiPage** control.



3. Select **C1MultiPage1** and set the following properties in the Properties grid.
   - Set the **Height** property to **300px** and the **Width** property to **400px**.

     The **C1MultiPage** control is resized.
   - Set the ShowToolBar property to **True**.

     The **Next** and **Previous** navigation buttons are visible.
4. Click **OK** to save and close the **C1MultiPage Designer Form**.
5. Run the project and note that you can now navigate between the three PageViews that you added:

In the next step, we will add content to the **C1MultiPage** control and customize its appearance.

Step 3 of 3: Adding Content to the C1MultiPage Control

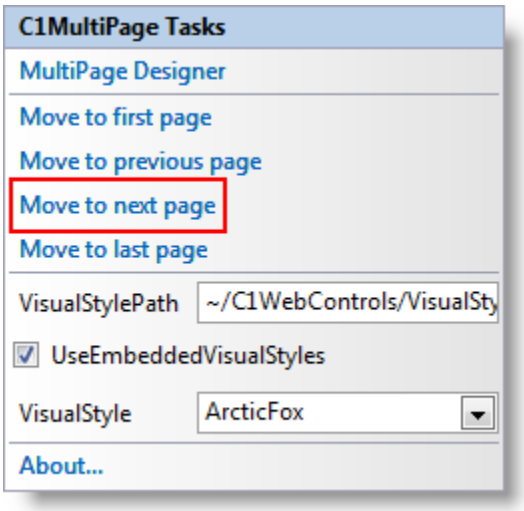
In this step you will add content to the **C1MultiPage** control and customize its appearance. Adding content to the control is as simple as clicking in the control's body and typing text, or adding controls from the Toolbox. Complete the steps outlined in the Working with the C1MultiPage Designer Form (page 21) to add three PageViews to the **C1MultiPage** control before completing the following.

Complete the following to add standard controls and text content to pages in the **C1MultiPage** control:

1. Click in the content area of the **C1MultiPage** control and type "This is Page 1."

   The text content is added to the first page of the **C1MultiPage** control.

2. Click the **C1MultiPage1**'s smart tag to open the **C1MultiPage Tasks** menu and select **Move to next page** to edit the second page.



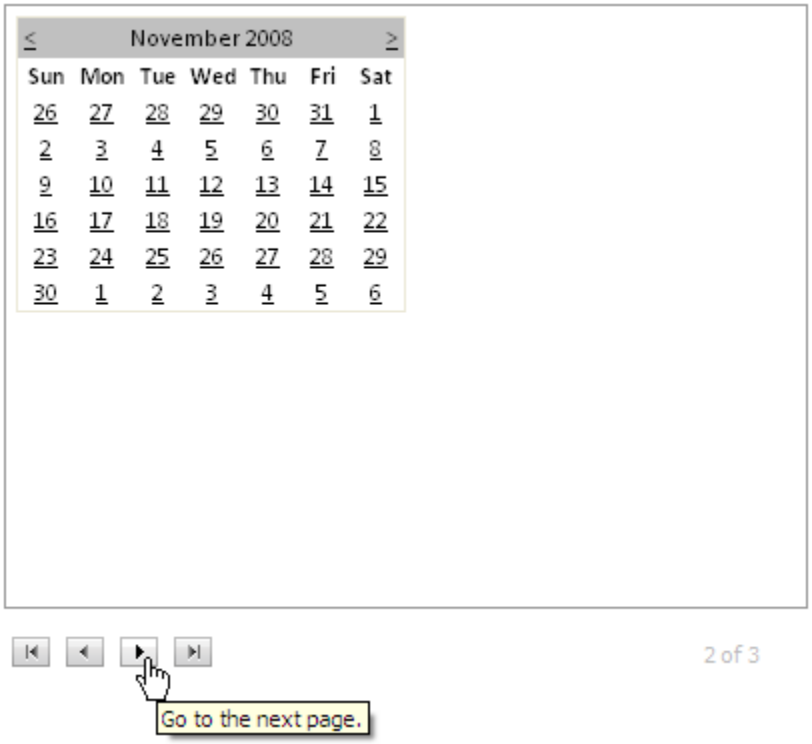
3. Navigate to the Toolbox and add a control to the second C1PageView. For example, add the standard **Calendar** control to the **C1MultiPage** control.

4. Switch to Source view to observe the structure of the C1MultiPage control, and add the following `<asp:Button>` tag within the `<cc1:C1PageView>` tag so it appears like the following:

```
<cc1:C1PageView ID="C1PageView03" runat="server">
    <asp:Button ID="Button1" runat="server" Text="Button" />
</cc1:C1PageView>
```

   This will add a Button control to the last page of the **C1MultiPage** control.

5. Run your application and note that you can navigate through the content on each PageView:



Top Tips

This section provides tips and tricks for using the C1MultiPage control.

- Remember always to use the **ScriptManager** control.
- Use image sprites in your custom visual styles to increase performance and decrease load times.
- To decrease bandwith, render only the currently selected page. This can be achieved by setting LoadOnDemand to **True** and AutoPostBack to **True**. For task-based help, see Loading Pages on Demand.
- Set visual styles on your Studio for ASP.NET AJAX control to add rich themes to your application. For task-based help, see Changing the Visual Style. For a list of visual styles, see Visual Styles (page 31).
- Use the built-in animations to add transitional effects to your Web site or Web application.
- Add easing in and easing out to make your animations smoother and more natural.
- Extend our animation library to *any* part of your web application, including portions that aren't ComponentOne controls. See C1MultiPage Animation and Transition Effects (page 34) for a complete list of animation and transition effects.
- Update the client-side model properties and events when you don't need to perform server-side processing. The C1MultiPage control can be coded on both the client side and server side.

C1MultiPage Design-Time Support

C1MultiPage provides smart tags and a designer that offers rich design-time support and simplifies working with the object model.

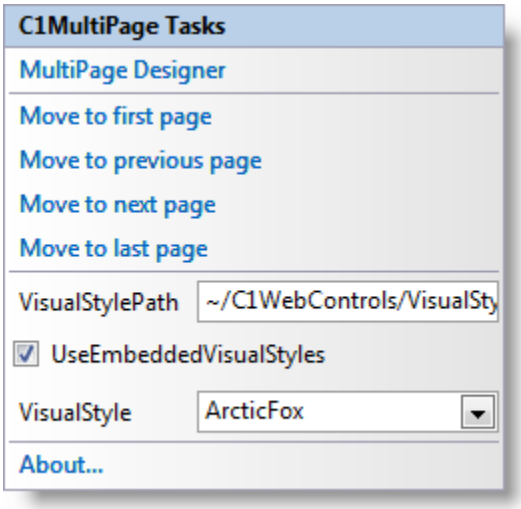
The following topics describe how to use **C1MultiPage**'s design-time environment to configure the **C1MultiPage** control.

C1MultiPage Smart Tag

The **C1MultiPage** control includes a smart tag in Visual Studio. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in **C1MultiPage**.

The **C1MultiPage** control provides quick and easy access to the **MultiPage Designer** and common properties through its smart tag.

To access the **C1MultiPage Tasks** menu, click on the smart tag in the upper-right corner of the **C1MultiPage** control. This will open the **C1MultiPage Tasks** menu.



The **C1MultiPage Tasks** menu operates as follows:

- **MultiPage Designer**

  Clicking on the **MultiPage Designer** item opens the **MultiPage Designer Form** where you can quickly configure **C1MultiPage**'s elements without having to scroll through its **Properties** window. You can load and save the control's content and can add additional C1PageViews. For more information on the **MultiPage Designer Form**, see C1MultiPage Designer Form (page 28).

- **Move to first page**

  Clicking on the **Move to the first page** item makes the first C1PageView in the C1MultiPage control visible in Design view, allowing you to view and change that C1PageView"s content.

- **Move to previous page**

  Clicking on the **Move to previous page** item makes the previous C1PageView in the C1MultiPage control visible in Design view, allowing you to view and change that C1PageView"s content.

- **Move to next page**

Clicking on the **Move to next page** item makes the next C1PageView in the C1MultiPage control visible in Design view, allowing you to view and change that C1PageView"s content.

- **Move to last page**

  Clicking on the **Move to last page** item makes the last C1PageView in the C1MultiPage control visible in Design view, allowing you to view and change that C1PageView"s content.

- **VisualStylePath**

  Sets the path to the built-in or custom visual style.

- **UseEmbeddedVisualStyles**

  When selected, this forces the control to use a built-in visual style. When unselected, the control will shed its built-in visual style and you will have to add a custom style.

- **VisualStyle**

  Add a visual style for the C1MultiPage control by selecting a theme from the **VisualStyle** drop-down list. You can choose from **ArcticFox**, **Office2007Black**, **Office2007Blue**, **Office2007Silver**, and **Vista**.

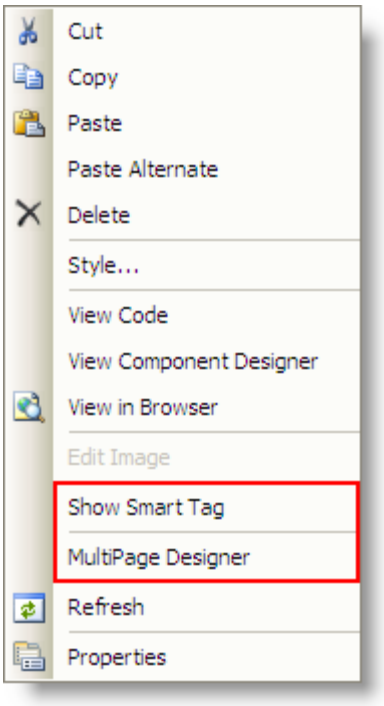
- **About**

  Clicking **About** reveals the **About ComponentOne** dialog box. This dialog box displays the version number and licensing information for the ComponentOne product.

C1MultiPage Context Menu

C1MultiPage has additional commands available on the context menu that Visual Studio provides for all .NET and ASP.NET controls.

Right-click anywhere on the C1MultiPage control to display the context menu:



The **C1MultiPage** context menu operates as follows:

- **Show Smart Tag**

  Clicking **Show Smart Tag** opens the **C1MultiPage Tasks** menu.

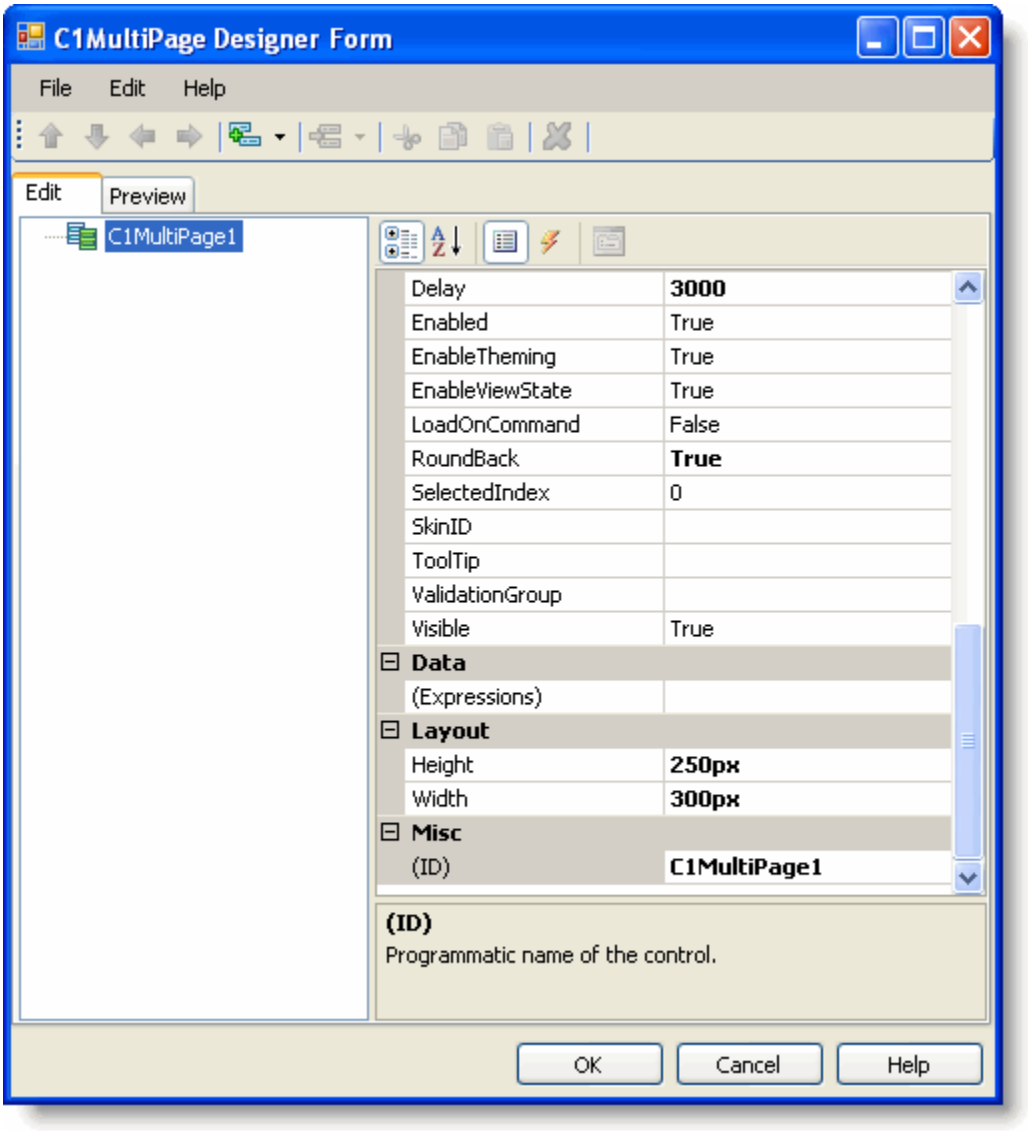
- **MultiPage Designer**

  Clicking on the **MultiPage Designer** item opens the **MultiPage Designer Form** where you can quickly configure **C1MultiPage**'s elements without having to scroll through its **Properties** window. You can load and save the control's content and can add additional C1PageViews. For more information on the **MultiPage Designer Form**, see C1MultiPage Designer Form (page 28).

C1MultiPage Designer Form

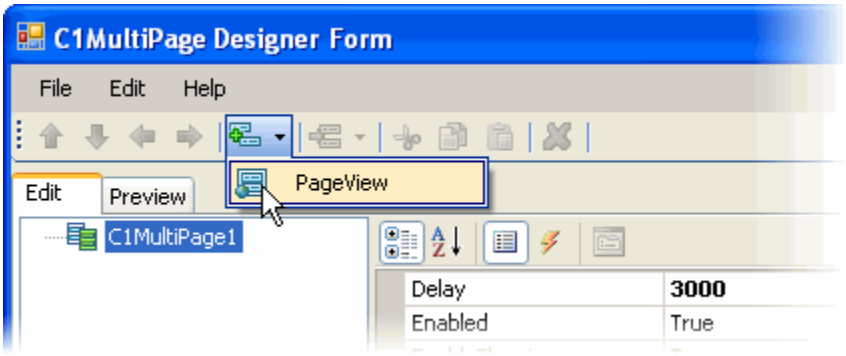
The **MultiPage Designer Form** lets you quickly configure the C1MultiPage control's elements without having to scroll through the control's **Properties** window. Using the **MultiPage Designer Form** you can load and save the control's content and can add additional pages.

To access the **MultiPage Designer Form** select the **MultiPage Designer** item from the **C1MultiPage Tasks** menu (see C1MultiPage Smart Tag (page 26) for details) or right-click on the C1MultiPage control at design time and select **MultiPage Designer**. The Designer looks similar to the following:



Add a C1PageView

To add a C1PageView to the C1MultiPage, select the C1MultiPage and in the toolbar click the **Add Child Item** button.



Delete a C1PageView

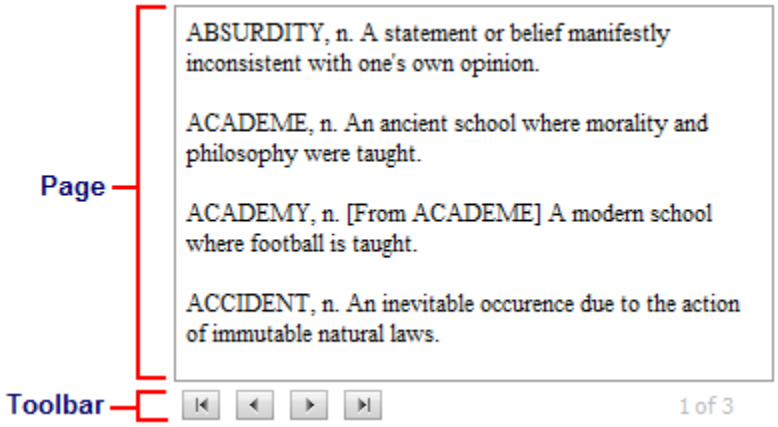
To delete a C1PageView, select the existing C1PageView and select the **Delete** button from the toolbar or **Edit** menu or right-click the item and select **Delete** from the context menu that appears.

Rename an Item

To rename a C1PageView, select the existing C1PageView and select **Rename** from the **Edit** menu or right-click the item and select **Rename** from the context menu that appears. Type in text to change its ID.

C1MultiPage Elements

The C1MultiPage control is used to display a collection of pages. The pages are viewed one at a time, allowing your page to hold a lot of content without wasting screen real estate. The pages are represented by the C1PageView class. Each page can be navigated using the optional navigational buttons.



Page

Pages of a C1MultiPage control can hold formatted text. For more information about adding text to a C1Multipage, see Adding Text to a Page.

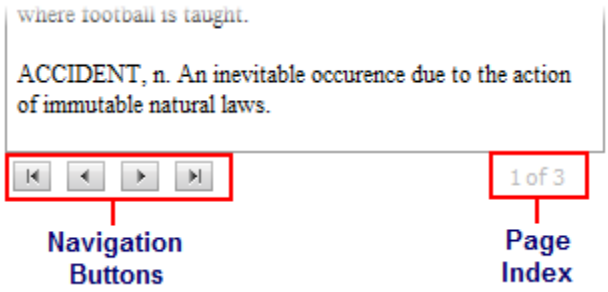
Using the **ContentUrl** property, you can set a page of a C1MultiPage to display external content. See Displaying External Content in a Page for more information.

Arbitrary controls can also be added to C1MultiPage simply by declaring the server control within the C1PageView tag. For example, the following markup adds the Button server control inside the first page of the C1MultiPage control:

```
<cc1:C1PageView ID="C1PageView01" runat="server">
    <asp:Button ID="Button1" runat="server" Text="Button" />
</cc1:C1PageView>
```

For help on adding arbitrary controls to the C1MultiPage, see Adding Arbitrary Controls to a Page.

Toolbar



The toolbar consists of two elements: a set of navigation buttons and a page index. When the ShowToolBar property is set to **True**, the **First**, **Previous**, **Next**, and **Last** buttons appear near the bottom-left of the C1MultiPage control, while the page index appears on the bottom-right. You can remove the page index by setting the ShowPageIndex property to **False** (see Removing the Toolbar's Page Index).

If you prefer, you can add tabbed navigation to the C1MultiPage control by using it in conjunction with the **C1TabStrip** control (see Using a C1TabStrip for Navigation). You can also use arbitrary controls, such as buttons, for C1MultiPage navigation. See Using Arbitrary Buttons for Navigation and Selecting C1PageViews by ID for more information.

C1MultiPage Appearance

The following topics provide information about the appearance of the C1MultiPage control.

Visual Styles

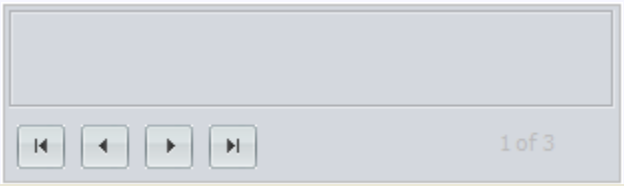
The C1MultiPage control provides five built-in visual styles that can be easily applied using the VisualStyle property. You can choose from one of the following schemes: **ArcticFox** (default), **Office2007Black**, **Office2007Blue**, **Office2007Silver**, and **Vista**.

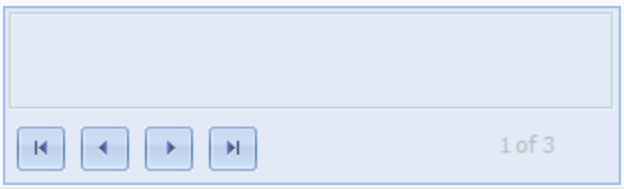
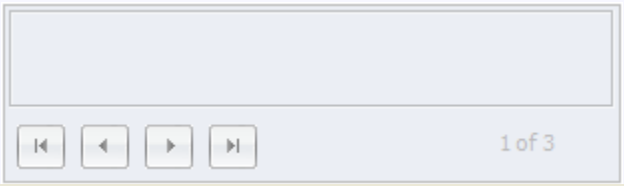
Using the Visual Styles

To learn how to add a preformatted scheme to C1MultiPage, see Changing the Visual Style.

Appearance of the Visual Styles

The C1MultiPage control contains the following visual styles:

Format	Appearance
ArcticFox	
Office200 Black	

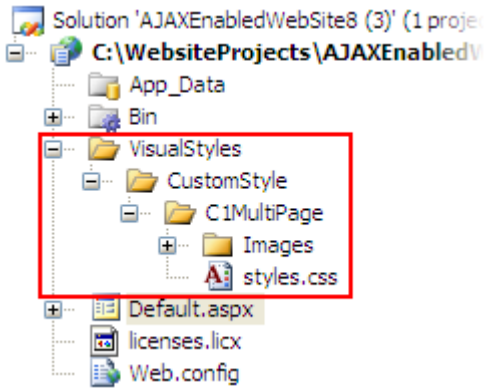
Office2007Blue	
Office2007Silver	
Vista	

Custom Visual Styles

While **MultiPage for ASP.NET AJAX** comes with five built-in styles, we recognize that there are instances where that you will want to customize your C1MultiPage control. To customize the C1MultiPage control, you will create a custom CSS style sheet and add it to your project as a visual style. The custom CSS style sheet must always be named "styles.css".

Tip: The easiest way to create a custom visual style is by modifying one of the control's pre-existing visual styles. You can find the .css style sheets and images for **C1MultiPage**'s visual styles within the installation directory at *C:\Program Files\ComponentOne\Studio for ASP.NET AJAX\C1WebUI\VisualStyles*.

Before you add your .css file and images, you will have to create a hierarchy of folders, the last of which will contain your files. On the top-level of your project, create a folder named "VisualStyles". Underneath the VisualStyles folder, create a sub-folder bearing the theme name (such as "CustomStyle"), and then, beneath that, create a sub-folder named "C1MultiPage". The image folder and .css file should be placed underneath the **C1MultiPage** folder. The result will resemble the following image:



This structure of these folders is very important; **C1MultiPage** will always look for the **~/VisualStyles/StyleName/C1MultiPage/styles.css** path, as it is the default visual style path for the control.

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

For task-based help on custom visual styles, see Adding Custom Visual Styles.

CSS Styling

You can apply CSS styling to modify the styling of a C1MultiPage"s elements. C1MultiPage offers two properties that allow you to alter different sections of the control. The properties are named and described in the table below.

Property	Description
C1MultiPage.CssClass	Sets the CSS class for the entire control.
C1PageView.CssClass	Sets the CSS class for an individual tab page.

You can add CSS styles to the control by adding a CSS style sheet to your project and setting an element's property to one of the CSS classes specified in your style sheet. To add font styling to one C1PageView, for example, you would add a class similar to the following to the style sheet:

```
.Font
{
       font-family:Comic Sans MS;
       font-size:x-large;
       color:Purple;
}
```

Once the class is added to the style sheet, you can assign the **Font** class to the **C1PageView.CssClass** property of the specific page you want to add that styling to. If you want to apply that styling to the entire control, set the **C1MultiPage.CssClass** property to the **Font** class name.

CSS style sheets are particularly useful if you have numerous controls that you want to add the same styles to. For example, you can create one style sheet that specifies the characteristics of a control's border and then apply it to both a C1MultiPage control and a **C1TabControl** control.

To add a style sheet to your project, select **Website | Add New** Item from the Visual Studio's toolbar to open the **Add New Item** dialog box. Select Style Sheet from the box and click **Add** to add it to your Web project.

Note: You will have to link your style sheet to your project by adding the following line of code between the `<head></head>` tags in your .aspx page: `<link href="StyleSheet.css" type="text/css" rel="stylesheet">`

For task-based help on CSS styling, see Applying CSS Styles to C1MultiPage.

C1MultiPage Behavior

The following topics provide information about C1MultiPage"s behavioral features. Some of these features affect how the control acts when loaded, whereas others affect the users' interactions with the control.

C1MultiPage AutoPlay Feature

You can set up the C1MultiPage control to automatically transition through page views by setting the AutoPlay property to **True**. By default, there will be a 3,000 millisecond (3 second) delay between automatic page transitions, but you can adjust this by setting the Delay property. The pages will stop autoplaying at on the last page of the C1MultiPage unless the Loop property is set to **True**; when Loop is set to **True**, the control will transition back to the first page and repeat the autoplay process.

C1MultiPage Animation and Transition Effects

The following topics provide information about **C1MultiPage**'s animation and transition effects.

C1MultiPage Animation Effects

C1MultiPage includes ten built-in animation options that enable you to customize interaction with the **C1MultiPage** control. By default, no animation occurs, but you can change the animation on a control by setting the Animation property.

For task-based help on using animation effects, see Using Animation Effects.

The table below describes each animation effect:

Name	Description
None (default)	No animation is used.
Auto	Page slides in from right if the next page in the succession is selected, but slides in from the left if the previous page is selected.
Fade	Previous page fades out as the next page fades in.
SlideLeft	New page slides in from the left side of the control.
SlideRight	New page slides in from the right side of the control.
SlideTop	New page slides in from the top of the control.
SlideBottom	New page slides in from the bottom of the control.
SlideLeftRight	Previous page slides out towards the left of the control while the next page slides in from the right of the control.
SlideRightLeft	Previous page slides out towards the right of the control while the next page slides in from the left of the control.
SlideTopBottom	Previous page slides out towards the top of the control while the next page slides in from the bottom of the control.
SlideBottomTop	Previous page slides out towards the bottom of the control while the next page slides in from the top of the control.

Animation Effect Duration

You can set the length of C1MultiPage"s animation effect using the **Duration** property. The unit of time used for specifying animation effect duration is in milliseconds, and the default setting for the **Duration** property is **500** milliseconds (or half a second). Increase this value for a longer animation effect, and decrease this number for a shorter animation effect.

C1MultiPage Transition Effects

The C1MultiPage control contains twenty-six built-in animation transition effects that allow you to customize how your animation effects are transitioned between page views. The default transition is **EaseLinear**, but you can set it to another transitional effect by setting the **Easing** property.

For task-based help on using transition effects, see Using Animation Effects.

The table below describes each animation effect:

Name	Description
FadeIn	Expands body of the control so that it appears to fade in.
FadeOut	Collapses the body of the control, so that it appears to fade out.
ScrollInFromTop	Expands the body of the control, scrolling into view from the top.
ScrollInFromRight	Expands the body of the control, scrolling into view from the right.
ScrollInFromBottom	Expands the body of the control, scrolling into view from the bottom.
ScrollInFromLeft	Expands the body of the control, scrolling into view from the left.
ScrollOutToTop	Collapses the body of the control, scrolling out of view to the top.
ScrollOutToRight	Collapses the body of the control, scrolling out of view to the right.
ScrollOutToBottom	Collapses the body of the control, scrolling out of view to the bottom.
ScrollOutToLeft	Collapses the body of the control, scrolling out of view to the left.
Fold	Collapses the body of control vertically and then horizontally so it appears to unfold.
UnFold	Expands the body of control horizontally and then vertically so it appears to unfold.
OpenVertically	Expands the body of control vertically from the center of the body area.
CloseVertically	Collapses the body of control vertically from the center of the body area.
OpenHorizontally	Expands the body of control horizontally from the center of the body area.
CloseHorizontally	Collapses the body of control horizontally from the center of the body area.
Shake	Expands or Collapses the body of control with a horizontal shaking motion.
Bounce	Expands or Collapses the body of control with a vertical bouncing motion.
DropInFromTop	Expands the body of the control from below the control to the top.
DropInFromRight	Expands the body of the control from the left of the control to the right.
DropInFromBottom	Expands the body of the control from above the control to the bottom.
DropInFromLeft	Expands the body of the control from the right of the control to the left.
DropOutToTop	Collapses the body of the control out to above the control.

DropOutToRight	Collapses the body of the control out to the right of the control.
DropOutToBottom	Collapses the body of the control out to below the control.
DropOutToLeft	Collapses the body of the control out to the left of the control.

ToolTips

You can use the **C1PageView.ToolTip** property to create a user-friendly interface. ToolTips are graphic user interface elements that are used to provide users with information or instructions regarding elements of a user interface. When users hover over the page with their cursor, a box will appear with the additional information.

ToolTips can be applied to each page of a C1MultiPage by setting the **C1PageView.ToolTip** property to a string. If you wanted to set a ToolTip for the first tab of the control, you would use the following code:

- Visual Basic
```
C1PageView.ToolTip = "Hello World!"
```
- C#
```
C1PageView.ToolTip = "Hello World!";
```

You can also set the **C1PageView.ToolTip** property in Design view or in Source view. For more information on setting this property, see Displaying ToolTips for Pages.

Selected Index

The C1MultiPage control's tabs follow a zero-based index, meaning that the index of the first tab is zero. By default, the SelectedIndex property will be set to zero ("0"), and the first page will have focus at run-time. To change which page is selected at run-time, set the SelectedIndex property to a different number in the index. For example, if you have four pages and want the last one to be selected at run-time, you would set the SelectedIndex property to **3**.

For task-based help on setting the SelectedIndex property, see Changing C1MultiPage's Selected Index.

Client-Side Functionality

The C1MultiPage control includes a rich and flexible client-side object model. Several server-side properties and methods can be used on the client-side.

When a C1MultiPage control is added to a Web project, an instance of the client-side **C1MultiPage** control will be created automatically. For example, if you have a **C1MultiPage** control with the server-side ID of "C1MultiPage1", you can use the following script to acquire a reference to its client object:

```
var newMultiPage = $find("<%= C1MultiPage1.ClientID %>");
```

Using C1MultiPage"s client-side functionality, you can implement many features in your Web page without having to post back to the server. Thus, using client-side methods , properties, and events will increase the efficiency of your Web site.

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

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 task-based help on setting a property on the client-side, see Setting the Duration of Animations at Run-Time.

See **C1MultiPage**'s client-side reference for more information.

Client-Side Methods

C1MultiPage 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 **C1MultiPage** reference.

For task-based help on using client-side methods, see Creating Play and Stop Buttons, Selecting C1PageViews by ID, and Using Arbitrary Buttons for Navigation.

See **C1MultiPage**'s client-side reference for more information.

MultiPage 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.

You can access samples from the **ComponentOne Control Explorer**. To view samples, click the **Start** button and then click **ComponentOne | Studio for ASP.NET | Control Explorer**. The following table provides a short overview of each sample.

Sample	Description
Visual Styles	Illustrates **C1MultiPage**'s five built-in visual styles.
Auto-Play	Demonstrates **C1MultiPage**'s auto play functionality.
Load on Demand	Demonstrates the 'load on demand' feature that can be in conjunction with the auto play feature. When **LoadOnDemand** and **AutoPostBack** are set to True, there is only one page is rendered to client every time.
Animation	Illustrates the animation and transition effects that can be used with the **C1MultiPage** control.

MultiPage for ASP.NET AJAX Task-Based Help

The task-based help section assumes that you are familiar with programming in the Visual Studio ASP.NET environment and have a general understanding of the **ComponentOne MultiPage** control.

Each topic provides a solution for specific tasks using the C1MultiPage control. By following the steps outlined in each topic, you will be able to create projects using a variety of C1MultiPage features.

Each task-based help topic also assumes that you have created a new AJAX-enabled ASP.NET project. **ComponentOne MultiPage 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 C1MultiPage 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.

Adding a C1MultiPage in Code

To add a C1Multipage control to your Web project in code, complete these steps:

1. Add a **PlaceHolder** control to your project.
2. Import the following namespace into your project:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1MultiPage
     ```
   - C#
     ```
     using C1.Web.UI.Controls.C1MultiPage;
     ```
3. Create a C1MultiPage object and add it to the page:
   - Visual Basic
     ```
     Dim NewMultiPage As New C1MultiPage()
     PlaceHolder1.Controls.Add(NewMultiPage)
     ```
   - C#
     ```
     C1MultiPage NewMultiPage = new C1MultiPage();
     PlaceHolder1.Controls.Add(NewMultiPage);
     ```
4. Set C1MultiPage"s height, width, and visual style:
   - Visual Basic
     ```
     NewMultiPage.Width = 100
     NewMultiPage.Height = 100
     NewMultiPage.VisualStyle = "Office2007Blue"
     ```
   - C#
     ```
     NewMultiPage.Width = 100;
     NewMultiPage.Height = 100;
     NewMultiPage.VisualStyle = "Office2007Blue";
     ```
5. Run the program.

This Topic Illustrates the Following:

When the program is run, your C1MultiPage will appear on the page as a blue box. It resembles the following image:

Adding and Manipulating Page Content

A C1MultiPage control can hold arbitrary controls, display text, and display external content. In the following topics, you will learn how to add and manipulate the content of a C1MultiPage control.

Adding Arbitrary Controls to a Page

You can add arbitrary controls to each of your **C1MultiPage**'s pages using a simple drag-and-drop operation, HTML, or code. In this topic, you will create a C1MultiPage control with two pages and then add an arbitrary control to each page in Design view, in Source view, and in code

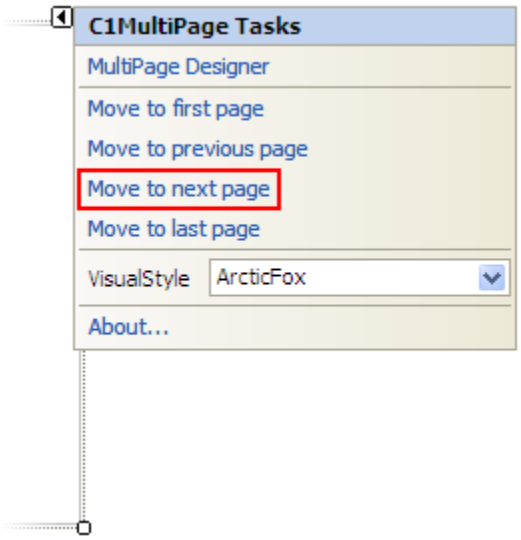
Adding Arbitrary Controls in Design View

To add arbitrary controls, complete these steps:

1. Click C1MultiPage"s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.

   The **C1MultiPage Designer Form** appears.

2. Use the **Add Child Item** button  to add two pages to your C1MultiPage.
3. Click **OK** to close the **C1MultiPage Designer Form**.
4. Select a **Button** control from the Visual Studio Toolbox and drag it onto the C1MultiPage control.
5. Click C1MultiPage"s smart tag (▶) and select **Move to next page** to bring the second page into focus.



6. Select a **TextBox** from the Toolbox window and drag it onto the C1MultiPage control.
7. Run the project.

Adding Arbitrary Controls in Source View

To add arbitrary controls, complete these steps:

1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and then select **MultiPage Designer**.

   The **C1MultiPage Designer Form** appears.
2. Use the **Add Child Item** button  to add two pages to your **C1MultiPage**.
3. Click **OK** to close the **C1MultiPage Designer Form**.
4. Click the **Source** tab to enter Source view.
5. Locate the `<cc1:C1PageView>` tags for `PageView1` and place the following tag between them:

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

6. Locate the `<cc1:C1PageView>` tags for `PageView2` and place the following tag between them:

```
<asp:TextBox ID="TextBox2" runat="server" />
```

7. Run the project.

Adding Arbitrary Controls in Code

To add arbitrary controls, complete these steps:

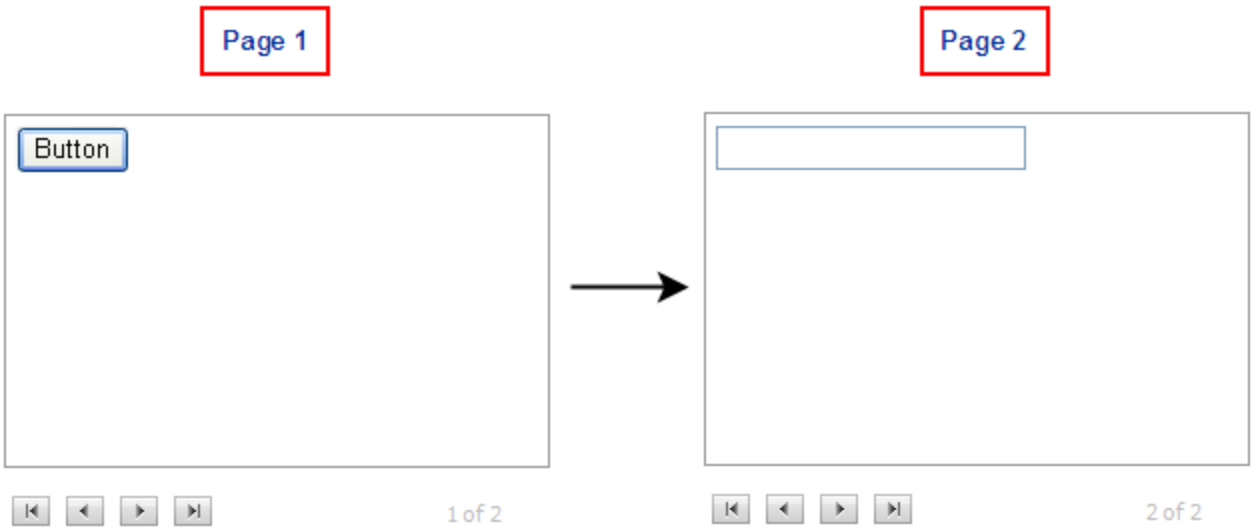
1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and then select **MultiPage Designer**.

   The **C1MultiPage Designer Form** appears.
2. Use the **Add Child Item** button  to add two pages to your **C1MultiPage**.
3. Click **OK** to close the **C1MultiPage Designer Form**.

4. In the Solution Explorer window, right-click on the project and select **View Code** to enter the code editor.
5. Create a **Button** control and add text to it by entering the following code to the **Page_Load** event:
   - Visual Basic
     ```
     Dim nuButton As Button =  New Button()
     nuButton.Text = "Hello World!"
     ```
   - C#
     ```
     Button nuButton = new Button();
     nuButton.Text = "Button";
     ```
6. Create a **TextBox** control:
   - Visual Basic
     ```
     Dim nuTextBox As TextBox = New TextBox()
     ```
   - C#
     ```
     TextBox nuTextBox = new TextBox();
     ```
7. Add the **Button** control to the first page:
   - Visual Basic
     ```
     C1PageView1.Controls.Add(nuButton)
     ```
   - C#
     ```
     C1PageView1.Controls.Add(nuButton);
     ```
8. Add the **TextBox** control to the second page:
   - Visual Basic
     ```
     C1PageView2.Controls.Add(nuTextBox)
     ```
   - C#
     ```
     C1PageView2.Controls.Add(nuTextBox);
     ```
9. Run the program.

This Topic Illustrates the Following:

After you run the project, use the navigation buttons (see Using the Toolbar for Navigation (page 46)) to switch between the first and second pages. Observe that a **Button** control appears on the first page, whereas a **TextBox** control appears on the second page.

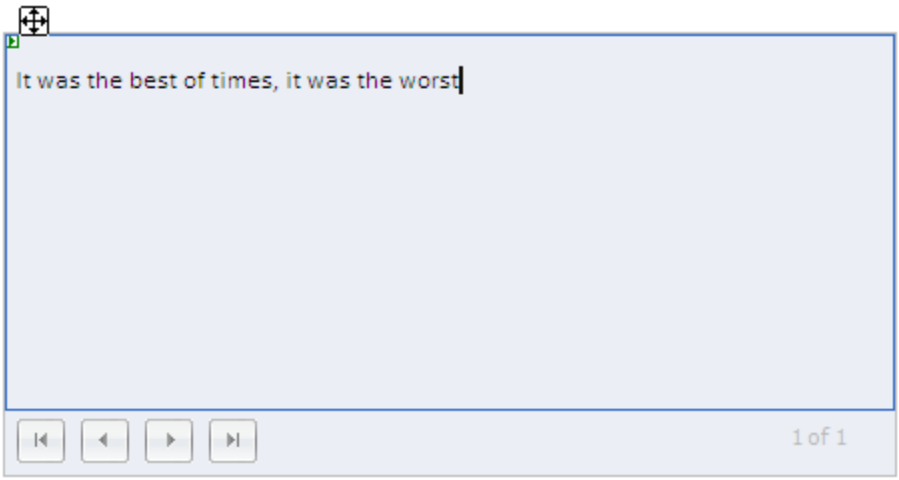


Adding Text to a Page

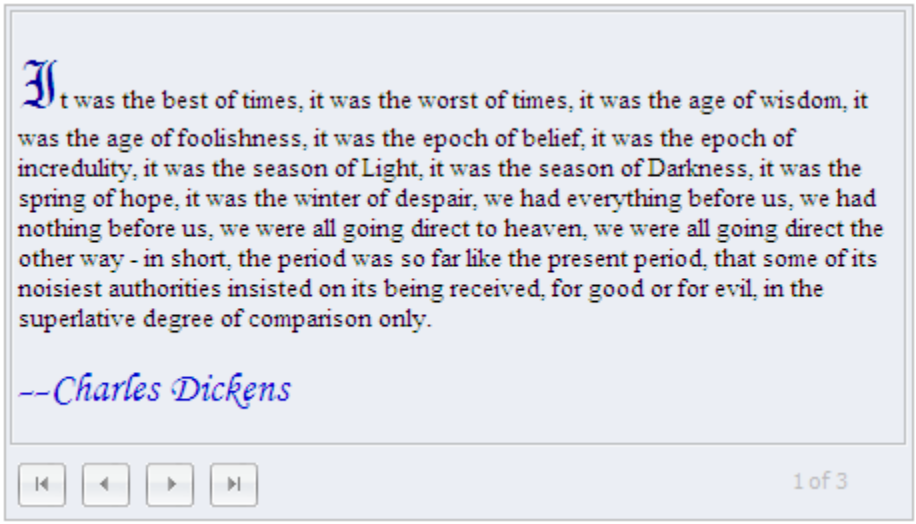
In this topic, you will learn how to add text to a C1MultiPage control using the designer and HTML markup.

Adding Text to a PageView in Design View

To add text to a page, simply place your cursor inside the page and type (or copy) the text into the page.



Once you've added text to the page, you can use Visual Studio's Formatting toolbar (to view this toolbar, use the following path: **View | Toolbars | Formatting**) to format the text. The image below features a **C1PageView** with formatted text:



Adding Text to a PageView in Source View

You can add text to a C1PageView in Source view by placing text between the `<cc1: C1PageView>` tags. To format the text, you would use HTML markup.

To add text to a **C1PageView** in Source view, follow these steps:

1. Add a C1MultiPage control to your project.
2. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.

   The **C1MultiPage Designer Form** appears.
3. Use the **Add Child Item** button  to add one page to your **C1MultiPage**.
4. Switch to Source view and paste the following text and HTML tags between the `<cc1: C1PageView>` tags:

```
You can also add text to a <b>C1MultiPage</b> control in Source view.
Simply place the text between the <span style="color: #0000ff; font-
family: Courier New">&lt;cc1: C1PageView&gt;</span> tags and use HTML
markup to format the text.
```

5. Click the **Design** tab to return toDesign view and observe that text has been added to the **C1PageView** of your **C1MultiPage**. Your result will resemble the following image:

Displaying External Content in a Page

The **ContentUrl** property can be used to display external content inside of the C1MultiPage control. You can either use it to display the content of a Web page that you have created or to exhibit a Web page outside of your domain. In this topic, you will set the **ContentUrl** property in Design view, in Source view, and in Code.

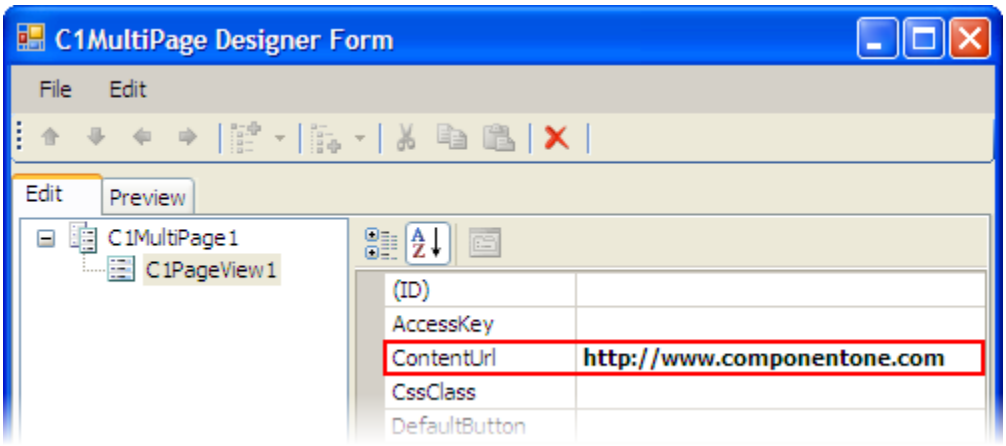
Setting the ContentUrl Property in Design View

Complete the following steps to set the **ContentUrl** property:

1. Click C1MultiPage"s smart tag to open the **C1Multipage Tasks** menu and then select **MultiPage Designer**.

   The **C1MultiPage Designer Form** opens.

2. In the **C1MultiPage Designer Form**, click the **Add Child Item** button  to add a new page to the control.
3. In treeview, select the **C1PageView1** and set the **ContentUrl** property in the Properties grid. You can set this to any URL; for this example, we're going to type in "http://www.componentone.com".



4. Click **OK** to exit the **C1MultiPage Designer Form**.
5. Run the project and note that the **C1MultiPage** now displays the ComponentOne Web site.

Setting the ContentUrl Property Demand in Source View

To set a C1MultiPage to load on demand in Source view , add `ContentUrl="http://www.componentone.com"` to the `<cc1:C1PageView>` tag. Your HTML will resemble the following:

```
<cc1:C1PageView ID="C1PageView1" runat="server"
ContentUrl="http://www.componentone.com">
```

Run the project and note that the **C1MultiPage** now displays the ComponentOne Web site.

Setting the ContentUrl Property in Code

To set the **ContentUrl** property, complete the following:

1. Import the following namespace to your project:

   - Visual Basic
     `Imports C1.Web.UI.Controls.C1MultiPage`

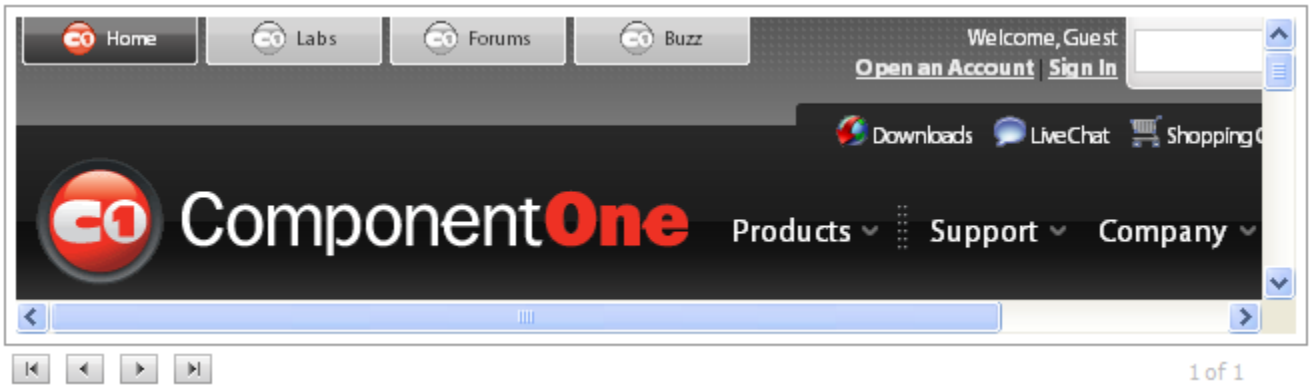
- C#
```
using C1.Web.UI.Controls.C1MultiPage;
```

2. Add the following code to the **Page_Load** event:
   - Visual Basic
```
Me.C1PageView1.ContentUrl = http://www.componentone.com
```
   - C#
```
this.C1PageView1.ContentUrl = "http://www.componentone.com";
```

3. Run the project and note that the **C1MultiPage** now displays the ComponentOne Web site.

This Topic Illustrates the Following:

A C1MultiPage can be used to exhibit outside content. The graphic below features a C1MultiPage control displaying external content:



Adding Navigation to a C1MultiPage Control

The following topics illustrate how to add navigation to a C1MultiPage control using its built-in toolbar or by binding it to a **C1TabStrip** control. To learn about navigation options that involve using client-side script, see Using Arbitrary Buttons for Navigation (page 57) and Selecting C1PageViews by ID (page 59).

Using the Toolbar for Navigation

In its default state, the C1MultiPage control doesn't show its set of navigation buttons. This topic explains how to add navigation buttons to your C1MultiPage control in Design view, in Source view, and in code.

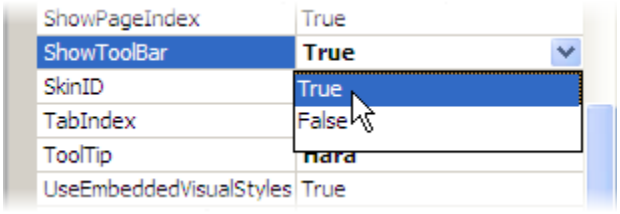
Adding Navigation Buttons in Design View

To add navigation buttons, follow these instructions:

1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.

   The **C1MultiPage Designer Form** opens.

2. In treeview, select **C1MultiPage1** to reveal its list of properties.
3. Locate the ShowToolbar property, click its drop-down arrow, and select **True** from the list.



4. Press **OK** to close the **C1MultiPage Designer Form**.

   Observe that the **C1MultiPage** control features four navigation buttons.

Adding Navigation Buttons in Source View

To add navigation buttons to the C1MultiPage control in Source view, add `ShowToolBar="True"` to the `<cc1:C1MultiPage>` tag. The final result should resemble the following:

```
<cc1:C1MultiPage ID="C1MultiPage1" runat="server" Height="250px" Loop="True"
ShowToolBar="True" VisualStylePath="~/C1WebControls/VisualStyles" Width="300px">
```

Observe that the **C1MultiPage** control features four navigation buttons.

Adding Navigation Buttons in Code

To add navigation buttons to a **C1MultiPage**, complete the following steps:

1. Import the following namespace into your project:

   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1MultiPage
     ```

   - C#
     ```
     using C1.Web.UI.Controls.C1MultiPage;
     ```

2. Add the following code to your **Page_Load** event:

   - Visual Basic
     ```
     C1MultiPage1.ShowToolBar = True
     ```

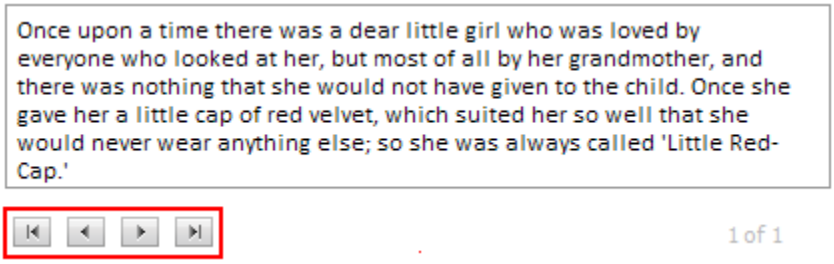
   - C#
     ```
     C1MultiPage1.ShowToolBar = True;
     ```

3. Run the program.

   Observe that the **C1MultiPage** control features four navigation buttons.

This Topic Illustrates the Following:

Adding navigation buttons is as simple as setting the ShowToolBar property to **True**. The image below features a C1MultiPage control with a series of navigation buttons:



Observe that a page index has also been added to the C1MultiPage. To remove the page index, see Removing the Toolbar's Page Index (page 48).

Removing the Toolbar's Page Index

When navigation buttons are added to the control, the C1MultiPage control adds a page index to the bottom left corner of the control. If you don't want this page index to appear on your control, you can remove it by setting the ShowPageIndex property to **False**. In this topic, you will set the PageIndex property in Design view, in Source view, and in code.

Note: The page index materializes by default when you add navigation buttons to the **C1MultiPage**. To add navigation buttons to your control, please see Using the Toolbar for Navigation (page 46).

Removing the Page Index in Design View

To remove the page index, complete the following steps:

1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.

   The **C1MultiPage Designer Form** opens.

2. In treeview, select **C1MultiPage1** to reveal a list of its properties.
3. Locate the ShowPageIndex property, click its drop-down arrow, and select **False**.
4. Press **OK** to close the **C1MultiPage Designer Form**.

   Observe that the **C1MultiPage** control no longer features a page index in the bottom left corner of the control.

Removing the Page Index in Source View

To add navigation buttons to the C1MultiPage control in Source view, add `ShowPageIndex="False"` to the `<cc1:C1MultiPage>` tag. The HTML should resemble the following:

```
<cc1:C1MultiPage ID="C1MultiPage1" runat="server" Height="250px" Loop="True"
ShowPageIndex="False" VisualStylePath="~/C1WebControls/VisualStyles"
Width="300px">
```

Observe that the **C1MultiPage** control no longer features a page index in the bottom left corner of the control.

Removing the Page Index in Code

To remove the page index of a **C1MultiPage** control in code, complete the following steps:

1. Import the following namespace into your project:

   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1MultiPage
     ```

- C#
```
using C1.Web.UI.Controls.C1MultiPage;
```

2. Add the following code to your **Page_Load** event:

   - Visual Basic
```
C1MultiPage1.ShowPageIndex = False
```

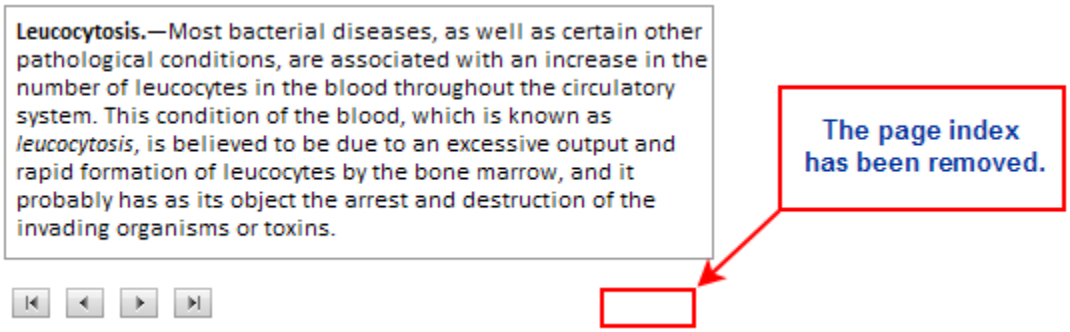
   - C#
```
C1MultiPage1.ShowPageIndex = false;
```

3. Run the program.

   Observe that the **C1MultiPage** control no longer features a page index in the bottom left corner of the control.

This Topic Illustrates the Following:

The image below features a C1MultiPage with its default page index removed:



Using a C1TabStrip for Navigation

You can add tabbed navigation by using the C1MultiPage control in coordination with the **C1TabStrip** control. This topic illustrates how to use a C1MultiPage control with a **C1TabStrip** control.

Step 1: Add a C1MultiPage Control and Create Pages

To add a **C1MultiPage** control and create pages, complete the following:

1. Add a **C1MultiPage** control to your project.
2. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.
   The **C1MultiPage Designer Form** opens.
3. Use the **Add Child Item** button  to add three pages to your **C1MultiPage**. Note that the new pages are named as follows:
   - C1PageView1
   - C1PageView2
   - C1PageView3
4. Click **OK** to leave the **C1MultiPage Designer Form**.

Step 2: Add a C1TabStrip Control and Create Tabs

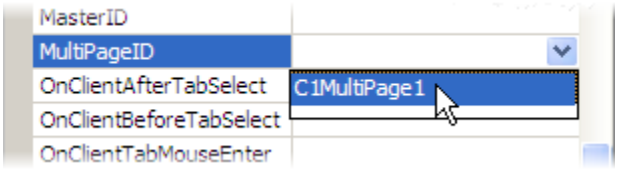
To add a **C1TabStrip** control and create tabs, complete the following:

1. Add a **C1TabStrip** control to your project.
2. Click **C1TabStrip's** smart tag (▶) and select **TabStrip Designer** from the menu.
   The **C1TabStrip Designer Form** opens.
3. Use the **Add Child Item** button  to add three tabs to your **C1TabStrip**. Note that the new tabs are named as follows:
   - Tab1
   - Tab2
   - Tab3

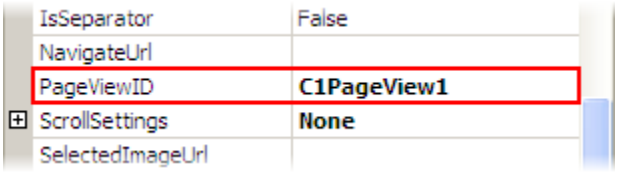
Step 3: Bind the C1TabStrip Control to the C1MultiPage Control

To bind the **C1TabStrip** control to the **C1MultiPage** control, complete the following:

1. While in the **C1TabStrip Designer Form**, select **C1TabStrip1** in treeview to reveal its list of properties.
2. Locate the **MultiPageID** property and select **C1MultiPage1** from its drop-down list.



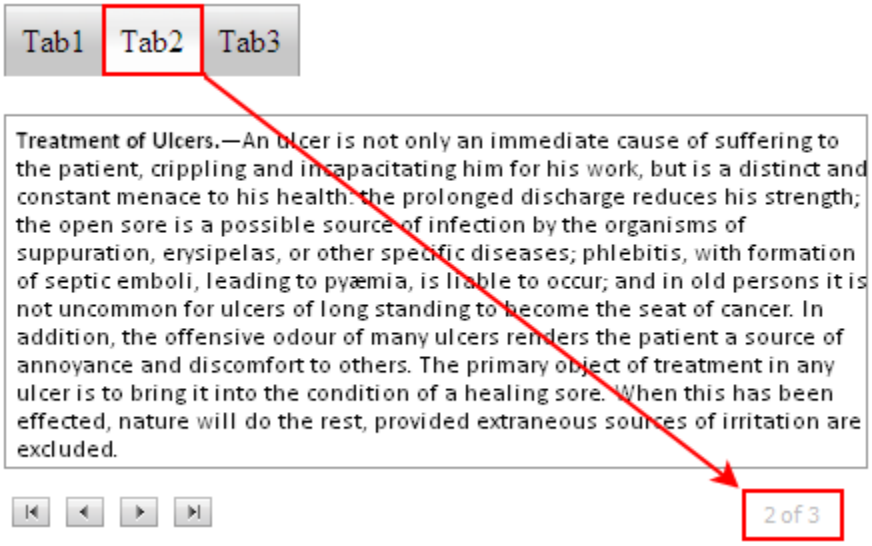
3. Select **Tab01** from the treeview, locate the **PageViewID** property, and enter "C1PageView1" into its text box.



4. Repeat the previous step for **Tab02** and **Tab03**, only this time you will enter the following information into their text boxes.
   - **Tab2** – "C1PageView2"
   - **Tab3** – "C1PageView3"
5. Press **OK** to leave the **C1TabStrip Designer Form** and then run the project.

This Topic Illustrates the Following:

A **C1MultiPage** control can be used with a **C1TabStrip** just by setting a few properties. Once you've run your project, click the second tab of the **C1TabStrip** and observe that the second page of the **C1MultiPage** comes into view. The following image illustrates a **C1MultiPage** that has been bound to a **C1TabStrip**:



Applying CSS Styles to C1MultiPage

In situations where you need to apply the same style settings to several controls, you might want to use CSS style sheets to keep the intricacy and repetition of your code to a minimum. With a CSS style sheet, you can create classes specifying layout, border types, color, and typographic characteristics; once those classes are created, you can apply them to numerous controls, thus eliminating the need for code duplication.

You can add CSS styles to the C1MultiPage control by adding a CSS style sheet to your project and setting an element's property to one of the CSS classes specified in your style sheet. In the following topics, you will learn how to apply CSS styles to the different areas (C1MultiPage or C1PageView) of the C1Multipage control.

These topics only cover CSS styling of a control. For information on how to style the C1MultiPage control using the designer, HTML, or code, see Customizing the Appearance of C1MultiPage (page 63).

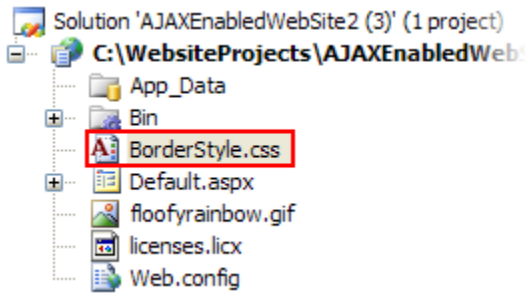
Adding a Border to the C1MultiPage Control

The following steps illustrate how to apply a border style to the C1MultiPage control using CSS styles. Upon completion of this topic, the C1MultiPage control will have a thick, inset border that is lime green in color. To accomplish this task, you will create a CSS style sheet, add a class specifying border styles to that style sheet, and then call that class by setting the **CSSClass** property.

To add a border to **Panel1** of the C1MultiPage control, complete these steps:

1. Add a C1MultiPage control to your Web project.
2. In Solution Explorer, right-click on the project name and select **Add New Item**.

   The **Add New Item** dialog box appears.
3. In the **Add New Item** dialog box, complete the following tasks:
   a. From the Templates pane, select **Style Sheet**.
   b. Enter "BorderStyles.css" into the **Name** field.
   c. Press **Add** to close the **Add New Item** dialog box.
   BorderStyles.css is added to your project.



4. In Solution Explorer, double-click **BorderStyles.css** to open the file in Visual Studio. The CSS body style appears by default:

```
body {
}
```

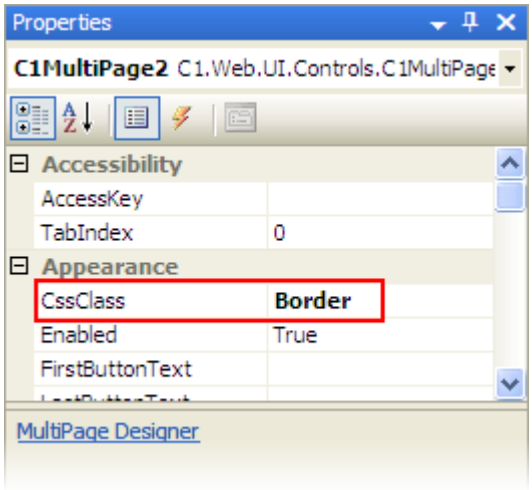
5. Place the following CSS code after the right bracket of the `body` style:

```
.Border

{
	border-style:inset;
	border-width:thick;
	border-color:Lime;
}
```

6. Return to the Design view of your .aspx page and complete the following tasks:
   a. Right-click the C1MultiPage control to open its context menu and select **Properties**.

      C1MultiPage"s properties take focus in the **Properties** window.
   b. Set the **C1MultiPage.CSSClass** property to "Border".



7. Now you will have to link the Web project to the **BorderStyles.css** style sheet by calling it in the project's Source code. To link the style sheet to the project, click the **Source** tab to switch to Source view and place the following line of code between the <head> tags:

```
<link href="BorderStyles.css" type="text/css" rel="stylesheet">
```

8. Press F5 to build your project. Observe that the control adopted the border styles specified in the **BorderStyles.css** style sheet.

This Topic Illustrates the Following:

By completing the steps in this topic, you have added a thick, lime green inset border to the C1MultiPage control. The final product will resemble the following image:

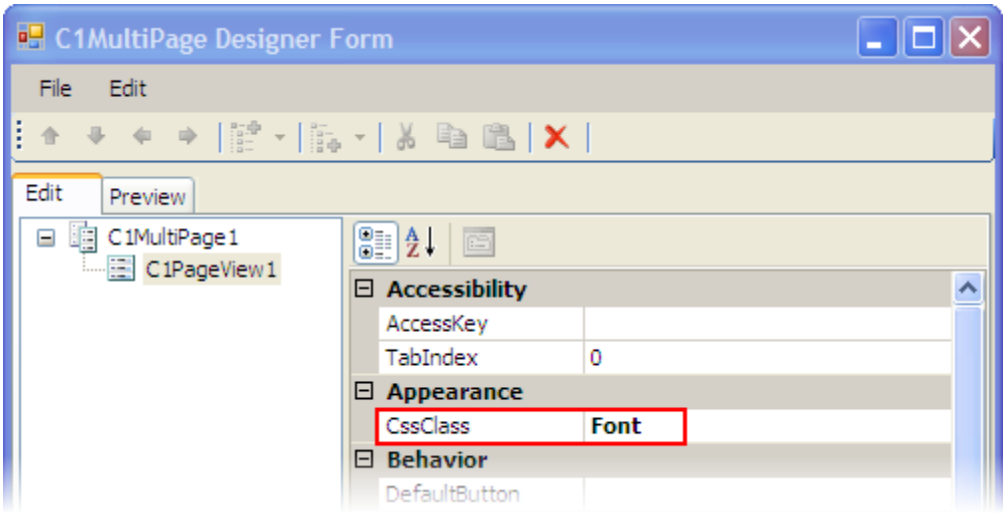
Adding Font Styles to a Page

The following steps demonstrate how to assign font styles to the individual pages of a C1MultiPage using CSS styles. Upon completion of this topic, the text on the first page of the C1MultiPage control will appear in a purple, extra-large Comic Sans font. To accomplish this task, you will create a CSS style sheet with a class and assign it to the first page's **C1PageView.CSSClass** property.

To add a gradient background to the first page of the C1MultiPage control, complete these steps:

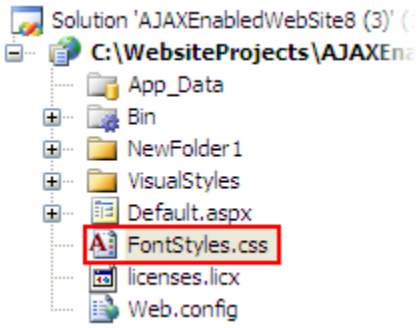
1. Add a C1MultiPage control to your Web project.
2. Click C1MultiPage"s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.
3. Use the **Add Child Item** button  to add a page to the control.

   A page named **C1PageView1** appears in treeview.
4. In treeview, select **C1PageView1** to reveal its list of properties. Set the **C1PageView.CSSClass** property to "Font"; this will be the name of the CSS class you create in a later step.



5. Click **OK** to close the **C1MutliPage Designer Form**.
6. In Solution Explorer, right-click on the project name and select **Add New Item**.

   The **Add New Item** dialog box appears.
7. In the **Add New Item** dialog box, complete the following tasks:
   a. From the Templates pane, select **Style Sheet**.
   b. Enter "FontStyles.css" into the **Name** field.
   c. Press **Add** to close the **Add New Item** dialog box.
   FontStyles.css is added to your project.



8. In Solution Explorer, double-click **FontStyles.css** to open the file in Visual Studio. The CSS body style appears by default:

```
body {
}
```

9. Place the following CSS code after the right bracket of the body style:

```
.Font
{
      font-family:Comic Sans MS;
      font-size:x-large;
      color:Purple;
```

```
}
```

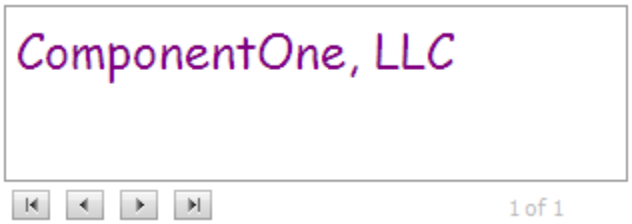
10. Now you will have to link the Web project to the **FontStyles.css** style sheet by calling it in the project's Source code. To link the style sheet to the project, click the **Source** tab to switch to Source view and place the following line of code between the `<head>` tags:

```
<link href="FontStyles.css" type="text/css" rel="stylesheet">
```

11. Click the **Design** tab to return to design view.
12. Place your cursor in the C1MultiPage control and add text to the control (page 43).
13. Press F5 to build your project. Observe that the text you entered in step 12 is in an extra-large, purple Comic Sans font.

This Topic Illustrates the Following:

By completing the steps in this topic, you have added font styling to the C1MultiPage control. The final product will resemble the following image:



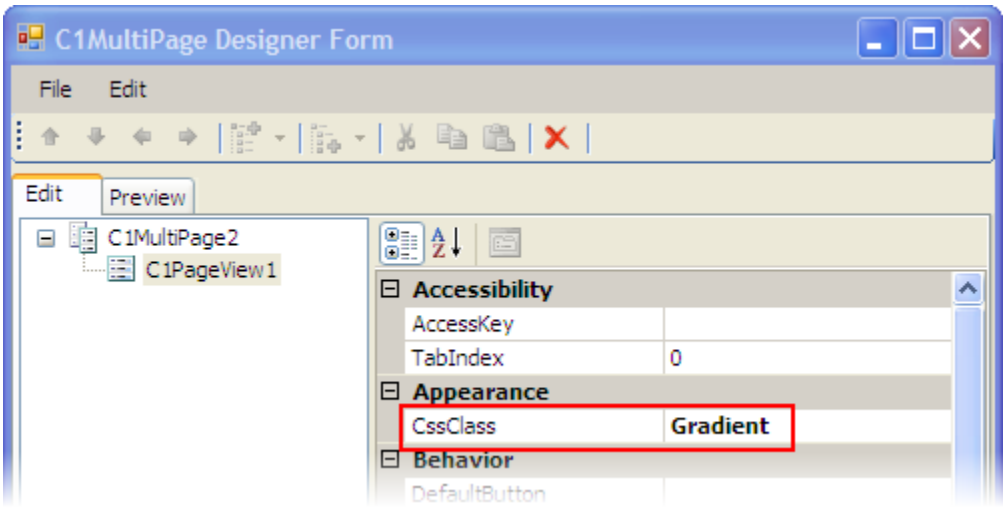
Adding a Gradient to a Page

The following steps demonstrate how to apply gradient backgrounds to the individual pages of a C1MultiPage using CSS styles. Upon completion of this topic, the first page of the C1MultiPage control will have a black and white, vertical gradient background. To accomplish this task, you will create a CSS style sheet with a class and assign it to the first page's **C1PageView.CSSClass** property.

To add a gradient background to the first page of the C1MultiPage control, complete these steps:

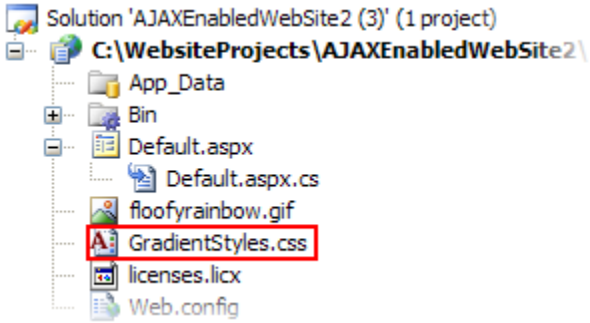
1. Add a C1MultiPage control to your Web project.
2. Click C1MultiPage"s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.
3. Use the **Add Child Item** button  to add a page to the control.

   A page named **C1PageView1** appears in treeview.
4. In treeview, select **C1PageView1** to reveal its list of properties. Set the **C1PageView.CSSClass** property to "Gradient"; this will be the name of the CSS class you create in a later step.



5. Click **OK** to close the **C1MutliPage Designer Form**.
6. In Solution Explorer, right-click on the project name and select **Add New Item**.

   The **Add New Item** dialog box appears.
7. In the **Add New Item** dialog box, complete the following tasks:
   a. From the Templates pane, select **Style Sheet**.
   b. Enter "GradientStyles.css" into the **Name** field.
   c. Press **Add** to close the **Add New Item** dialog box.
   GradientStyles.css is added to your project.



8. In Solution Explorer, double-click **GradientStyles.css** to open the file in Visual Studio. The CSS body style appears by default:

```
body {
}
```

9. Place the following CSS code after the right bracket of the body style:

```
.Gradient
{
        filter:progid:DXImageTransform.Microsoft.Gradient
        (GradientType=0, StartColorStr='#F0F0F0', EndColorStr='#000000')
}
```

10. Now you will have to link the Web project to the **GradientStyles.css** style sheet by calling it in the project's Source code. To link the style sheet to the project, click the **Source** tab to switch to Source view and place the following line of code between the `<head>` tags:

```
<link href="GradientStyles.css" type="text/css" rel="stylesheet">
```

11. Press F5 to build your project. Observe that the first page of the control features the gradient background specified in the **GradientStyles.css** style sheet.

This Topic Illustrates the Following:

By completing the steps in this topic, you have added a black and white, vertical gradient background to the first page of the C1MultiPage control. The final product will resemble the following image:

Client-Side Tasks for C1MultiPage

The following topics show how to perform various client-side tasks using **MultiPage for ASP.NET AJAX**.

Using Arbitrary Buttons for Navigation

This topic will show you how to use the **MoveFirst**, **MoveLast**, **MoveNext**, and **MovePrevious** client-side methods to create a customizable navigation system for the C1MultiPage control.

Complete the following steps:

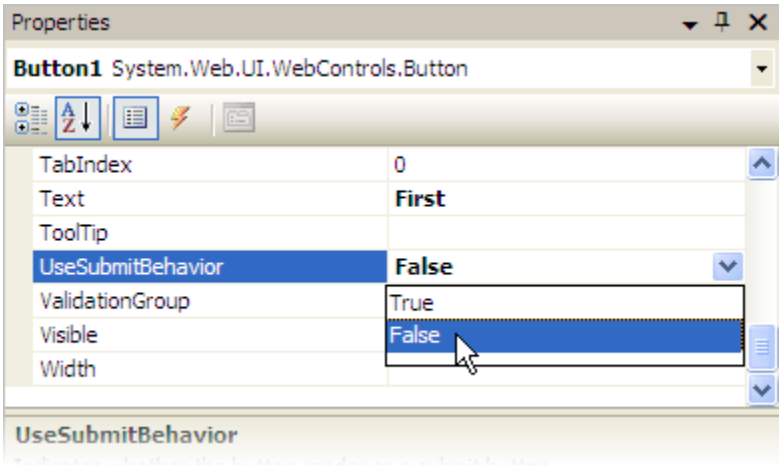
1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.

   The **C1MultiPage Designer Form** opens.

2. Use the **Add Child Item** button  to add three pages to the **C1MultiPage**.
3. Set the **ContentUrl** property of each page to the following:
   - **C1PageView1** – "http://www.google.com"
   - **C1PageView2** – "http://www.msn.com"
   - **C1PageView3** – "http://www.componentone.com"
4. Click **OK** to close the **C1MultiPage Designer Form**.
5. Navigate to the Toolbox window and add four standard **Button** controls your project. Set the **Text** property of each button as follows:

- **Button1** – "First"
- **Button2** – "Next"
- **Button3** – "Previous"
- **Button4** – "Last"

6. Set the **UseSubmitBehavior** property of each button to **False**. This will prevent the buttons from posting back to the server when clicked.



7. Place the following JavaScript code beneath `<HTML>` tag:

```
<script type="text/javascript">

function moveFirst()
{
    $find('<%= C1MultiPage1.ClientID %>').moveFirst();
}

function movePrevious()
{
    $find('<%= C1MultiPage1.ClientID %>').movePrevious();
}

function moveNext()
{
    $find('<%= C1MultiPage1.ClientID %>').moveNext();
}

function moveLast()
{
    $find('<%= C1MultiPage1.ClientID %>').moveLast();
}
   </script>
```

8. Set the **OnClientClick** property in each of the `<asp:Button>` tags so that they call the JavaScript functions you created in the last step. Your markup should resemble the following:

```
<asp:Button ID="Button1" runat="server" Text="First" Width="80px"
OnClientClick="moveFirst();return;" UseSubmitBehavior=false/>

<asp:Button ID="Button2" runat="server" Text="Previous" Width="80px"
OnClientClick="movePrevious();return;" UseSubmitBehavior=false/>

<asp:Button ID="Button3" runat="server" Text="Next" Width="80px"
OnClientClick="moveNext();return;" UseSubmitBehavior=false/>

<asp:Button ID="Button4" runat="server" Text="Last" Width="80px"
OnClientClick="moveLast();return;" UseSubmitBehavior=false/>
```

9. Run the project and use your custom buttons to navigate through the pages of the **C1MultiPage** control. Observe that the project doesn't refresh between page views.

Selecting C1PageViews by ID

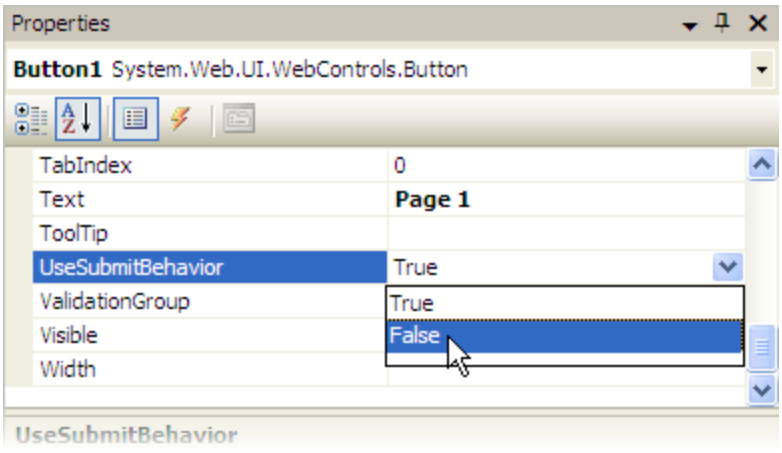
You can select a C1PageView with its ID on the client side using the **SelectPageView** method. To illustrate the utility of this method, this topic will show you how to use **SelectPageView** in combination with buttons to create a simple and customizable navigation system for your C1MultiPage.

Complete the following steps:

1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.

   The **C1MultiPage Designer Form** opens.

2. Use the **Add Child Item** button  to add three pages to the **C1MultiPage**.
3. Set the **ContentUrl** property of each page to the following:
   - **C1PageView1** – "http://www.google.com"
   - **C1PageView2** – "http://www.msn.com"
   - **C1PageView3** – "http://www.componentone.com"
4. Click **OK** to close the **C1MultiPage Designer Form**.
5. Navigate to the Toolbox window and add three standard **Button** controls your project. Set the **Text** property of each button as follows:
   - **Button1** – "Page 1"
   - **Button2** – "Page 2"
   - **Button3** – "Page 3"
6. Set the **UseSubmitBehavior** property of each button to **False**. This will prevent the buttons from posting back to the server when clicked.



7. Place the following JavaScript code beneath `<HTML>` tag:

```
<script type="text/javascript">

    function selectPage1()
    {
        $find('<%= C1MultiPage1.ClientID
%>').selectPageView("C1PageView1");
    }
    function selectPage2()
    {
        $find('<%= C1MultiPage1.ClientID
%>').selectPageView("C1PageView2");
    }
    function selectPage3()
    {
    $find('<%= C1MultiPage1.ClientID %>').selectPageView("C1PageView3");
    }

</script>
```

8. Set the **OnClientClick** property in each of the `<asp:Button> tags` so that they call the JavaScript functions you created in the last step. Your markup should resemble the following:

```
<asp:Button ID="Button1" runat="server" Text="Page1" Width="80px"
OnClientClick="selectPage1();return;" UserSubmitBehavior="False" />

<asp:Button ID="Button2" runat="server" Text="Page2" Width="80px"
OnClientClick="selectPage2();return;" UserSubmitBehavior="False" />

<asp:Button ID="Button3" runat="server" Text="Page3" Width="80px"
OnClientClick="selectPage3();return;" UserSubmitBehavior="False" />
```

9. Run the project and use your custom buttons to navigate through the pages of the **C1MultiPage** control. Observe that the project doesn't refresh between page views.

Creating Play and Stop Buttons

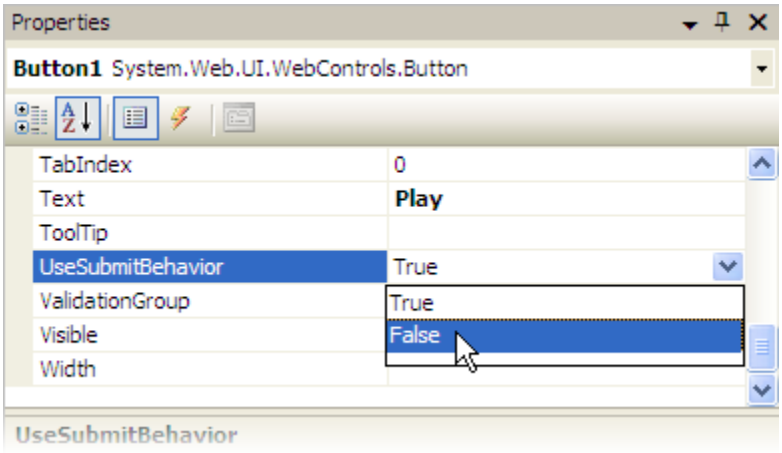
In this topic, you will create two buttons that will allow users to govern the **AutoPlay** feature of the **C1MultiPage** control. One of these buttons will begin the **AutoPlay** operation, whereas the other will stop it.

Complete the following steps:

1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.

   The **C1MultiPage Designer Form** opens.

2. Use the **Add Child Item** button  to add three pages to the **C1MultiPage**.
3. Set the **ContentUrl** property of each page to the following:
   - **C1PageView1** – "http://www.google.com"
   - **C1PageView2** – "http://www.msn.com"
   - **C1PageView3** – "http://www.componentone.com"
4. Click **OK** to close the **C1MultiPage Designer Form**.
5. Navigate to the Toolbox and add two standard **Button** controls your project. Set the **Text** property of each button as follows:
   - **Button1** – "Play"
   - **Button2** – "Stop"
6. Set the **UseSubmitBehavior** property of both buttons to **False**. This will prevent the buttons from posting back to the server when clicked.



7. Click the **Source** tab to enter Source view and place the following JavaScript beneath the <HTML> tag:

```
<script type="text/javascript">
function Play()
{
    $find('<%= C1MultiPage1.ClientID %>').play();
}
function Stop()
{
    $find('<%= C1MultiPage1.ClientID %>').stop();
```

```
}
</script>
```

8. Set the **OnClientClick** property in each of the `<asp:Button> tags` so that they call the JavaScript functions you created in the last step. Your markup should resemble the following:

```
<asp:Button ID="Button1" runat="server" Text="Play"
OnClientClick="Play();return;" UseSubmitBehavior="False" BackColor="Lime"
/>

<asp:Button ID="Button2" runat="server" Text="Stop"
OnClientClick="Play();return;" UseSubmitBehavior="False" BackColor="Red"
/></div>
```

9. Run the project and press the **Play** button to start **AutoPlay** mode. Now press **Stop** and observe that it halts the automatic playback.

Setting the Duration of Animations at Run-Time

In this topic, you will write a client-side script that allows users to set the duration of a **C1MultiPage** control's animation effect at run-time.

To set the duration of an animation at run-time, complete the following steps:

1. Add a new **C1MultiPage** control to your project.
2. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.

   The **C1MultiPage Designer Form** opens.
3. Use the **Add Child Item** button  to add three pages to the **C1MultiPage**.
4. In treeview, select **C1MultiPage1** and set its Animation property to **SlideRight**.
5. Click **OK** to close the **C1MultiPage Designer Form**.
6. Place your cursor in the C1MultiPage and then add text or an arbitrary control to the first page of the control (see Adding and Manipulating Page Content (page 40) for more information).
7. Click **C1MultiPage**'s smart tag (▶) and select **Move to next page from the menu**. Once the second page is in focus, add text or an arbitrary control to the page. Repeat this step to add content to the third page.
8. Click **OK** to close the **C1MultiPage Designer Form**.
9. Click the **Source** tab to enter Source view and place the following JavaScript beneath the `<HTML>` tag:

```
<script language="javascript" type="text/javascript">
function SetDuration()
{
    var oPage =
Sys.Application.findComponent("<%=C1MultiPage1.ClientID%>");
    oPage.set_duration(document.getElementById('TxtDuration').value);
}
</script>
```

10. Add the following markup beneath the `</cc1:C1MultiPage>` tag to add a text box and button to the project.

```
<input id="TxtDuration" type="text" size="3" value=""/><input
id="SetEnabled" type="button" value="Set Duration"
onclick="SetDuration()" />
```

11. Run the program, enter a low number (such as 5) into the text box and press the **Set Duration** button. Use the navigation buttons to switch page views and observe that the animation barely registers. Now type "3500" into the text box, press the **Set Duration** button, and use the navigation buttons to switch page views. This time, the new page will slide in slowly from the left side of the control.

Customizing the Appearance of C1MultiPage

MultiPage for ASP.NET AJAX features five built-in schemes, all of which can be adopted by setting a single property. For more customization, you can also add a custom CSS style sheet to the C1MultiPage control. The following topics provide steps on how to complete these tasks.

Adding Custom Visual Styles

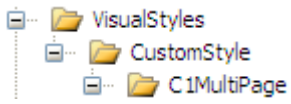
You can use the VisualStyle, **VisualStylePath**, and **UseEmbeddedVisualStyles** properties to create a custom visual style for your C1MultiPage. In this topic, you will add a custom visual style in Design view, in Source view, and in code.

For more information on custom visual styles, see Custom Visual Styles.

Adding Custom Visual Styles in Design View

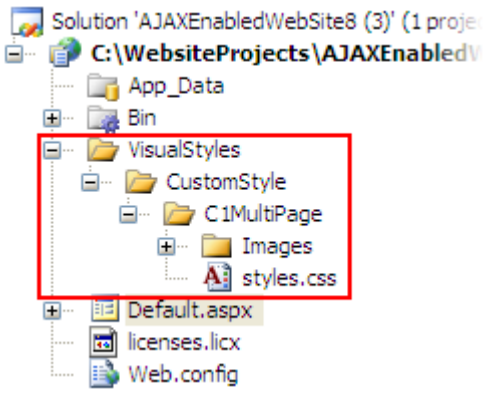
To add a custom visual style, complete the following steps:

1. In order to add a custom visual style to your project, you must first create new folders and add your .css file to the project. To do this, follow these steps:
    a. In the Solution Explorer window, right-click on your project to open its context menu and select **New Folder**. Name the new folder "VisualStyles".
    b. Add a new folder within the newly created **VisualStyles** folder and name it "CustomStyle".
    c. Add another folder,"C1MultiPage", under the **CustomStyle** folder that you made in the last step. Your folder hierarchy will match the following image:



> **Note**: The second folder in the hierarchy *must* match the name of the custom visual style.

    d. Right-click the **C1MultiPage** folder and select **Add New Item** to add a custom .css file to the project folder. Name the CSS style sheet "styles.css" and press **Add** to add the style sheet to your project.
    e. If your custom visual style includes images, add a folder named "Images" underneath the **C1MultiPage** folder. The folder hierarchy resembles the following:



2. Add CSS styling and images to the project by completing these steps:

    a. In Solution Explorer, double-click **styles.css** to open the style sheet and replace the existing CSS body style with the following:

    CSS Styles

```
/* ========== CustomStyle C1MultiPage
========================================== */
.C1MultiPage_CustomStyle
{
      font-size:12px;
      font-family: Segoe UI, Myriad, Myriad Pro, Calibri, Arial,
Sans-Serif;
      background:#7D9EC0;
      border: ridge 7px #FFE1FF;
      padding:2px;
      padding-right:6px !important;
      padding-bottom:6px !important;
      padding-right:2px;
      padding-bottom:2px;
}
.C1MultiPage_CustomStyle .C1pvOuter
{
      width:100%;
      height:100%;
      padding-bottom:6px !important;
      padding-bottom:0;
}
.C1MultiPage_CustomStyle .C1pvInner
{
      width:100%;
      height:100%;
      border-top:dotted 2px #B0E0E6;
      border-left:dotted 2px #B0E0E6;
      border-bottom:dotted 2px #B0E0E6;
      border-right:dotted 2px #B0E0E6;
      padding-bottom:2px !important;
      padding-right:2px !important;
      padding-bottom:0;
      padding-right:0;
}
.C1MultiPage_CustomStyle .C1pvContent
{
```

```
        width:100%;
        height:100%;
        border-top:solid 1px #f6f7f9;
        border-left:solid 1px #f6f7f9;
        border-bottom:solid 1px #bdbfc1;
        border-right:solid 1px #bdbfc1;
        padding:0;
}
.C1MultiPage_CustomStyle .C1NavBar
{
        width:100%;
        height:30px;
        line-height: 20px;
        margin-top:14px !important;
        margin-bottom:-6px !important;
        margin-top:4px;
        margin-bottom:2px;
        font-size: 14px;
        font-family:Comic Sans MS, "lucida
grande",verdana,arial,helvetica,sans-serif;
        color:Purple;
        text-align: right
}
.C1MultiPage_CustomStyle .C1NavBar a span
{
        width:24px;
        height:30px;
        text-decoration:none;
        float:left;
        cursor:pointer;
        margin:5px 5px;
}
.C1MultiPage_CustomStyle .C1NavBar a .C1FirstButton
{
        background: white url('Images/first.png') top center no-repeat;
}
.C1MultiPage_CustomStyle .C1NavBar a:hover .C1FirstButton
{
        background: white url('Images/first.png') center center no-
repeat;
}
.C1MultiPage_CustomStyle .C1NavBar a .C1PreviousButton
{
        background: white url('Images/previous.png') top center no-
repeat;
}
.C1MultiPage_CustomStyle .C1NavBar a:hover .C1PreviousButton
{
        background: white url('Images/previous.png') center center no-
repeat;
}
.C1MultiPage_CustomStyle .C1NavBar a .C1NextButton
{
        background: white url('Images/next.png') top center no-repeat;
}
.C1MultiPage_CustomStyle .C1NavBar a:hover .C1NextButton
{
```

```
        background: white url('Images/next.png') center center no-
repeat;
}
.C1MultiPage_CustomStyle .C1NavBar a .C1LastButton
{
        background: white url('Images/last.png') top center no-repeat;
}
.C1MultiPage_CustomStyle .C1NavBar a:hover .C1LastButton
{
        background: white url('Images/last.png') bottom center no-
repeat;
}
.C1MultiPage_CustomStyle .C1NavBar .C1PageInfo
{
        padding-right:10px;
        color:#7A67EE;
        text-decoration:none;
        font-weight:bolder;
}
```

b. In Solution Explorer, right-click the **Images** folder and select **Add Existing Item** to add images to the project.

You can add custom images to the project, or you can use the images from the following table. To use these images, right-click on each image to open its context menu and select **Save Picture As**. Save the images with the filenames listed in the table below.

Image	Filename
	first.png
	previous.png
	next.png
	last.png

3. Click **C1MultiPage**'s smart tag (▶) and to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.

   The **C1MultiPage Designer Form** opens.

4. In treeview, select **C1MultiPage1** to reveal its list of properties, then complete the following:

   - Set the ShowToolbar property to **True** to enable the toolbar.
   - Set the **UseEmbeddedVisualStyles** property to **False**.
   - Set the **VisualStylePath** property to "~/VisualStyles".
   - Set the **VisualStyle** property to **CustomStyle (external)**.

   Note: If **CustomStyle(external)** does not appear in the drop-down list, run the project and then repeat step 4.

5. Click **OK** to close the **C1MultiPage Designer Form**. Observe that the **C1MultiPage** has adopted your custom visual style.

Adding Custom Visual Styles in Source View:

To add a custom visual style, follow these steps:

1. Complete steps 1 and 2 under "Adding Custom Visual Styles in Design View".
2. Right-click C1MultiPage and select **Properties**, and set the ShowToolBar property to **True**. This will enable you to see the custom CSS style for the toolbar.
3. Enter Source view and enter `VisualStyle="CustomStyle", VisualStylePath="~/VisualStyles", ShowToolBar="True", UseEmbeddedVisualStyles="False"` into the `<cc1:C1MultiPage>` tag. Your HTML will resemble the following:

```
<cc1:C1MultiPage ID="C1MultiPage1" runat="server"
VisualStyle="CustomStyle" VisualStylePath="~/VisualStyles"
ShowToolBar="True" UseEmbeddedVisualStyles="False">
```

4. Run the project and observe that the C1MultiPage has adopted your custom visual style.

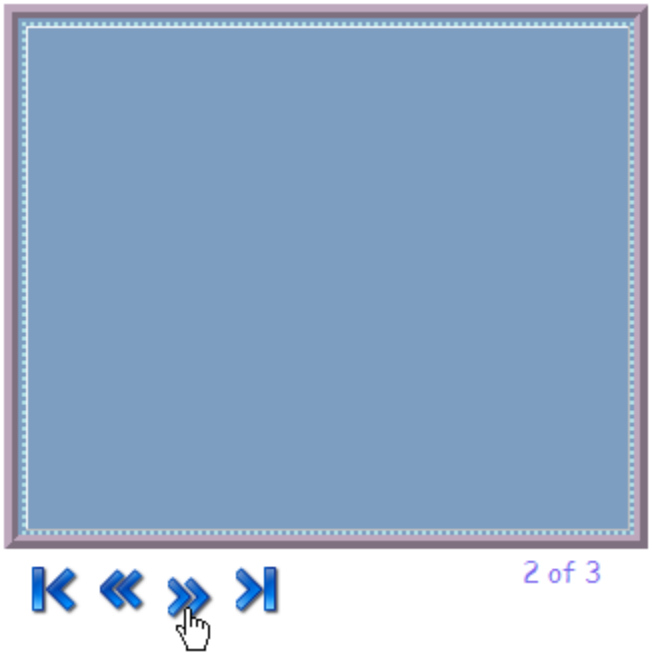
Adding Custom Visual Styles in Code

To add a custom visual style, follow these steps:

1. Complete steps 1 and 2 under "Adding Custom Visual Styles in Design View".
2. Right-click C1MultiPage and select **Properties**, and set the ShowToolBar property to **True**. This will enable you to see the custom CSS style for the toolbar.
3. Double-click the Web project to place a **Page_Load** event in the code editor.
4. Set the **UseEmbeddedVisualStyles** to **False** by adding the following code to the **Page_Load** event:
   - Visual Basic
     ```
     C1MultiPage1.UseEmbeddedVisualStyles = False
     ```
   - C#
     ```
     C1MultiPage1.UseEmbeddedVisualStyles = false;
     ```
5. Change the **VisualStylePath** property:
   - Visual Basic
     ```
     C1MultiPage1.VisualStylePath = "~/VisualStyles"
     ```
   - C#
     ```
     C1MultiPage1.VisualStylePath = "~/VisualStyles";
     ```
6. Select the **VisualStyle** property:
   - Visual Basic
     ```
     C1MultiPage1.VisualStyle = "CustomStyle"
     ```
   - C#
     ```
     C1MultiPage1.VisualStyle = "CustomStyle";
     ```
7. Run the project and observe that the C1MultiPage has adopted your custom visual style.

This topic illustrates the following:

Once your project is built, observe that a custom visual style has been added to the project. Hover over the navigation buttons and observe that a hover style has been applied to the buttons. Click the navigation buttons and note that the toolbar works just as it would under the built-in visual styles. The C1MultiPage control will resemble the following image:



Changing the Visual Style

A C1MultiPage control has five embedded visual styles that you can apply to your tabstrip with just a few clicks. This topic illustrates how to change the visual style in Design view, in Source view, and in code. For more information on visual styles, see Visual Styles.

Changing the Visual Style in Design View

To change the visual scheme of your **C1MultiPage**, follow these steps:

1. Click the **C1MultiPage** smart tag (▶) to open the **C1MultiPage Tasks** menu.
2. Click the **VisualStyle** drop-down arrow and select a visual style from the list. For this example, select **Office2007Blue**.

   The **Office2007Blue** visual scheme is applied to the C1MultiPage control.

Changing the Visual Style in Source View

To change the visual scheme of your **C1MultiPage** in Source view, add `VisualStyle="Office2007Blue"` to the `<cc1:C1MultiPage>` tag so that it resembles the following:

```
<cc1:C1MultiPage ID="C1MultiPage1" runat="server" VisualStyle="Office2007Blue"
VisualStylePath="~/C1WebControls/VisualStyles">
```

The **Office2007Blue** visual scheme is applied to the C1MultiPage control.

Changing the Visual Style in Code

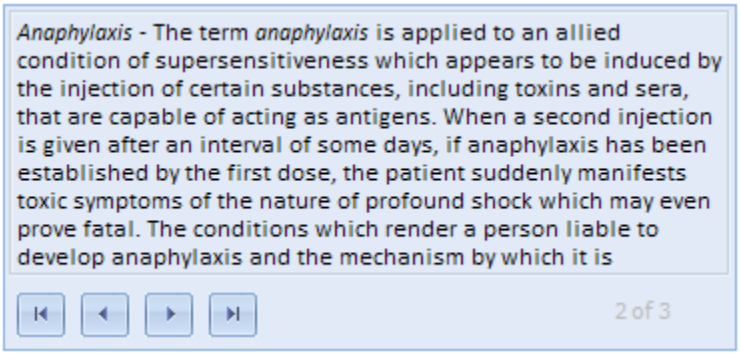
To change the visual scheme, follow these steps:

1. Import the following namespace into your project:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls
     ```
   - C#
     ```
     using C1.Web.UI.Controls;
     ```
2. Add the following code, which sets the VisualStyle property, to the **Page_Load** event:
   - Visual Basic
     ```
     Me.C1MultiPage1.VisualStyle = "Office2007Blue"
     ```
   - C#
     ```
     this.C1MultiPage1.VisualStyle = "Office2007Blue";
     ```
3. Run the program.

   The **Office2007Blue** visual scheme is applied to the C1MultiPage control.

This topic illustrates the following:

The following image shows a **C1MultiPage** with the **Office2007Blue** visual style:



Setting C1MultiPage Behaviors

The C1MultiPage control has a list of properties that affect how the control behaves at run time. Some of the properties affect how the control acts when loaded, whereas others affect the users' interactions with the control. The following topics will instruct you on how to modify the run-time actions of the control.

Changing C1MultiPage's Selected Index

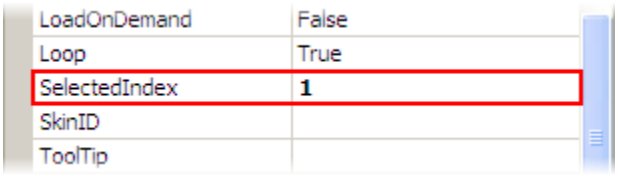
The SelectedIndex property of a C1MultiPage can be used to determine which page will be viewed when your project is run. The following topic shows you how to manipulate this property in Design view, in Source view, and in code.

Note: If you change the SelectedIndex property on a C1MultiPage and have it bound to a **C1TabStrip**, you will also have to manipulate **C1TabStrip**'s SelectedIndex property in order to get the page to load with the proper tab.

Changing the Selected Index in Design View

To change the SelectedIndex, use the following steps:

1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer.**
2. Use the **Add Child Item** button  to add three pages to your C1MultiPage.
3. Select **C1TabStrip** from the treeview to reveal its list of properties.
4. Locate the SelectedIndex property and set its value to "1".



   Observe that the default value of the SelectedIndex property is **0**. If you had kept this setting, your **C1MultiPage** would have loaded with the first page in view.

5. Press **OK** and run the project.

Changing the Selected Index in Source View

Add `SelectedIndex="1"` to the `<cc1:C1MultiPage>` tag. The resulting HTML should resemble the following:

```
<cc1:C1MultiPage ID="C1MultiPage1" runat="server" Height="29px"
                 MultiPageID="C1MultiPage1" OverlapOrder="Ascending"
VisualStyle="Office2007Blue"
                 VisualStylePath="~/C1WebControls/VisualStyles" Width="563px"
SelectedIndex="1">
```

Changing the Selected Index in Code

To change the selected index in code, use the following steps:

1. Import the following namespaces into your project:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1MultiPage
     ```
   - C#
     ```
     using C1.Web.UI.Controls.C1MultiPage;
     ```
2. Add the following code to the **Page_Load** event:
   - Visual Basic
     ```
     C1MultiPage.SelectedIndex = 1
     ```
   - C#
     ```
     C1MultiPage.SelectedIndex = 1;
     ```
3. Run the program.

Autoplaying the Pages of a C1MultiPage

By setting a single property, you can make the C1MultiPage control automatically navigate between pages. You can also customize autoplay by setting the delay between page transitions. This topic will show you how to set autoplay and customize its delay time in Design view, in Source view, and in code.

Setting up Autoplay in Design View

To set up auto play, complete the following steps:

1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer.**
   The **C1MultiPage Designer Form** opens.
2. Use the **Add Child Item** button  to add three **C1PageViews** to the **C1MultiPage** control.
3. Set the **ContentUrl** property to the following for each of your new pages:
   - **C1PageView1** – "http://www.microsoft.com"
   - **C1PageView2** – "http://www.google.com"
   - **C1PageView3** – "http://www.componentone.com"
4. In treeview, select **C1MultiPage1** to reveal a list of its properties and set the following.
   - Set the AutoPlay property to **True**.
   - Set the Delay property. For this example, we are going to set it to "1500" milliseconds so that 1.5 seconds will pass before the page will switch.

   Note: If you want the **C1MultiPage** control to stop autoplaying after one course, set its **Loop** property to **False**.
5. Press **OK** and run the project.

Setting up Autoplay in Source View

To set up auto play in source view, please complete the following steps:

1. Add a C1MultiPage control to your project.
2. Create three C1PageViews, add them to the C1MultiPage, and set their **ContentUrl** properties by adding the following HTML between the `<cc1:C1MultiPage>` tags:

```
<PageViews>
   <cc1:C1PageView ID="C1PageView1" runat="server" _designerRegion="0"
   ContentUrl="http://msdn.microsoft.com">
   </cc1:C1PageView>
   <cc1:C1PageView ID="C1PageView2" runat="server"
   ContentUrl="http://www.google.com">
   </cc1:C1PageView>
   <cc1:C1PageView ID="C1PageView3" runat="server"
   ContentUrl="http://www.componentone.com">
   </cc1:C1PageView>
</PageViews>
```

3. To set up AutoPlay in Source view, add `AutoPlay="True"` and `"Delay="1500"` to the `<cc1:C1MultiPage>` tag. Your HTML should resemble the following:

```
<cc1:C1MultiPage ID="C1MultiPage1" runat="server" Height="250px"
```

```
                VisualStylePath="~/C1WebControls/VisualStyles" Width="300px"
AutoPlay="True" Delay="1500">
```

Setting up AutoPlay Property in Code

To set up autoplay on your C1MultiPage control, complete the following steps:

1. Add a C1MultiPage control to your project.
2. Import the following namespace into your project:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1MultiPage
     ```
   - C#
     ```
     Using C1.Web.UI.Controls.C1MultiPage;
     ```
3. Create three C1PageViews by adding the following code to the **Page_Load** event:
   - Visual Basic
     ```
     'Create three new C1PageViews
     Dim C1PageView01 As New C1PageView()
     Dim C1PageView02 As New C1PageView()
     Dim C1PageView03 As New C1PageView()
     ```
   - C#
     ```
     //Create three new C1PageViews
     C1PageView C1PageView01 = new C1PageView();
     C1PageView C1PageView02 = new C1PageView();
     C1PageView C1PageView03 = new C1PageView();
     ```
4. Set the **ContentUrl** property of each page:
   - Visual Basic
     ```
     'Add content to the C1PageViews
     C1PageView01.ContentUrl = "http://msdn.microsoft.com"
     C1PageView02.ContentUrl = "http://www.google.com"
     C1PageView03.ContentUrl = "http://www.componentone.com"
     ```
   - C#
     ```
     //Add content to the C1PageViews
     C1PageView01.ContentUrl = "http://msdn.microsoft.com";
     C1PageView02.ContentUrl = "http://www.google.com";
     C1PageView03.ContentUrl = "http://www.componentone.com";
     ```
5. Add the C1PageViews to the C1MultiPage control:
   - Visual Basic
     ```
     'Add the new C1PageViews to the C1MultiPage
     C1MultiPage1.Controls.Add(C1PageView01)
     C1MultiPage1.Controls.Add(C1PageView02)
     C1MultiPage1.Controls.Add(C1PageView03)
     ```
   - C#
     ```
     //Add the new C1PageViews to the C1MultiPage
     C1MultiPage1.Controls.Add(C1PageView01);
     C1MultiPage1.Controls.Add(C1PageView02);
     C1MultiPage1.Controls.Add(C1PageView03);
     ```
6. Set the AutoPlay and Delay properties by adding the following code to your **Page_Load** event:

- Visual Basic
```
C1MultiPage1.AutoPlay = True
C1MultiPage1.Delay = 1500
```
- C#
```
C1MultiPage1.AutoPlay = true;
C1MultiPage1.Delay = 1500;
```

7. Run the program.

This Topic Illustrates the Following:

A **C1MultiPage** control can be set up to automatically play through its pages in a variety of speeds. When you run the program, observe that your **C1MultiPage** automatically cycles through each page.

Displaying ToolTips for Pages

This topic demonstrates how to create custom ToolTips for the pages of your C1MultiPage control. When the **ToolTip** property is set to a string, users will be able to see a brief description of the page by hovering over it with their cursor. In this topic, you create a ToolTip in Design view, in Source view, and in code.

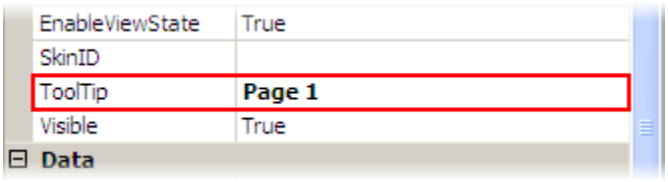
> **Note**: Since this feature is attached to the **C1PageView**, it doesn't work if you set the **ContentUrl** property.

Adding a ToolTip in Design View

To add a ToolTip, follow these steps:

1. Click the **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer.**

   The **C1MultiPage Designer Form** appears.
2. Use the **Add Child Item** button  to add a page to your C1MultiPage.
3. In treeview, select **C1PageView1** to reveal its list of properties.
4. Locate the **ToolTip** property and type "Page 1" into its text box.



5. Press **OK** to close the **C1MultiPageDesigner Form** and then run the project.

Adding a ToolTip through Source View

To add a ToolTip in code, add `ToolTip="Page 1"` to the `<cc1:C1PageView>` tags. Your HTML will resemble the following:

```
<cc1:C1PageView ID="C1PageView1" runat="server" ToolTip="Page 1">
```

Adding a ToolTip in Code

To add a ToolTip in code, follow these steps:

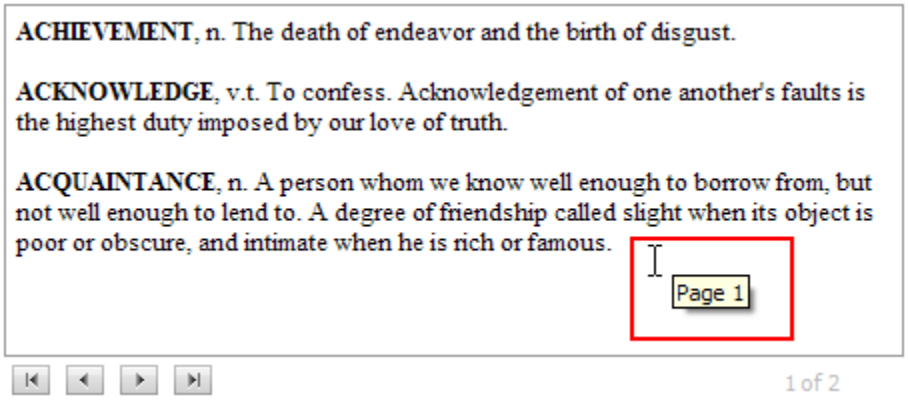
1. Import the following namespaces into your project:
   - Visual Basic
```
Imports C1.Web.UI.Controls.C1MultiPage
```

- C#
```
using C1.Web.UI.Controls.C1MultiPage;
```

2. Create the ToolTip by adding the following code to the **Page_Load** event:
   - Visual Basic
```
C1PageView1.ToolTip = "Page 1"
```
   - C#
```
C1PageView1.ToolTip = "Page 1";
```
3. Run the project.

This Topic Illustrates the Following:

Using the **ToolTip** property, you can easily create custom ToolTips that will appear when your users mouse over your tabs. The image below features a C1MultiPage with a ToolTip:



Loading Pages on Demand

To reduce initial page size and load time, you can set C1MultiPage to render only the currently selected page by setting the LoadOnDemand and AutoPostBack properties to **True**. This topic will describe how to set these properties in Design view, Source view, and in code.

Setting a C1MultiPage to Load on Demand in Design View

To set pages to load on demand, please use the following instructions:

1. Click the smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer.** The **C1MultiPage Designer Form** opens.
2. Click the **Add Child Item** button  *twice* to add two pages to the **C1MultiPage** control.
3. In treeview, select **C1MultiPage1** to reveal its list of properties and set the following:
   - Set the AutoPostBack property to **True**.
   - Set the LoadOnDemand property to **True**.
4. Press **OK** to close the **C1MultiPage Designer Form** and run the project.

Setting a C1MultiPage to Load on Demand in Source View

To set a C1MultiPage to load on demand in Source view , add `AutoPostBack="True" and LoadOnDemand="True"` to the `<cc1:C1MultiPage>` tag. Your HTML will resemble the following:

```
<cc1:C1MultiPage ID="C1MultiPage1" runat="server" AutoPostBack="True"
LoadOnDemand="True">
```

Run the project.

Setting a C1MultiPage to Load on Demand in Code

To set a C1MultiPage to load on demand in code, follow these instructions:

1. Import the following namespace into your project:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1MultiPage
     ```
   - C#
     ```
     using C1.Web.UI.Controls.C1MultiPage;
     ```
2. Add the following code to your **Page_Load** event:
   - Visual Basic
     ```
     C1MultiPage1.AutoPostBack = True
     C1MultiPage1.LoadOnDemand = True
     ```
   - C#
     ```
     C1MultiPage1.AutoPostBack = true;
     C1MultiPage1.LoadOnDemand = true;
     ```
3. Run the program.

Setting Automatic Postback

Using the AutoPostBack property, you can choose whether or not the C1MultiPage will automatically post back to the server between page views. By default, the AutoPostBack property is set to **False**, meaning that the C1MultiPage control does not automatically post back to the server. However, you can easily set up the C1MultiPage to post back to the server by setting this property to **True**. In this topic, you will set the AutoPostBack property in Design view, in Source view, and in code.

Setting AutoPostBack in Design View

Complete the following steps to set **AutoPostBack**:

1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer.**

   The **C1MultiPage Designer Form** appears.
2. In treeview, select **C1MultiPage1** to reveal its list of properties.
3. Locate AutoPostBack and set it to **True**.
4. Press **OK** to close the **C1MultiPage Designer Form**.

Setting AutoPostBack in Source View

In Source view add `AutoPostBack="True"` to the `<cc1:C1MultiPage>` tag so it appears similar to the following:

```
<cc1:C1MultiPage ID="C1MultiPage1" runat="server" AutoPostBack="True">
```

Setting AutoPostBack in Code

Complete the following steps to set **AutoPostBack**:

1. Import the following namespace into your project:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1MultiPage
     ```
   - C#
     ```
     using C1.Web.UI.Controls.C1MultiPage;
     ```
2. Set **AutoPostBack** to **True** by adding the following code to the **Page_Load** event:
   - Visual Basic
     ```
     C1MultiPage1.AutoPostBack = True
     ```
   - C#
     ```
     C1MultiPage1.AutoPostBack = true;
     ```
3. Run the project.

Using Animation Effects

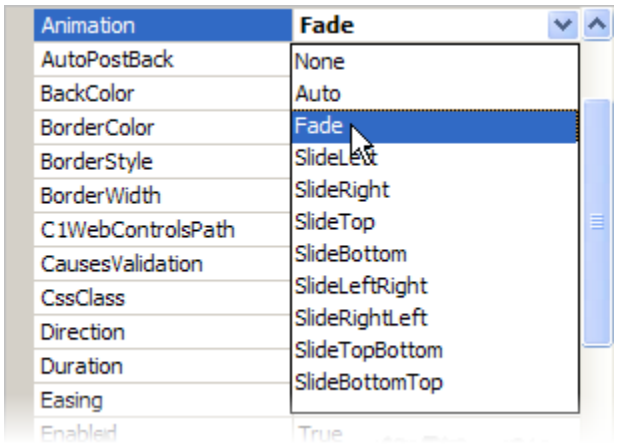
C1MultiPage contains ten animation and thirty-one transition effects that allow you to customize interaction with the control. In this topic, you will set the Animation, Easing, and Duration properties to create an animation effect between page views.

Setting Animation Effects in Design View

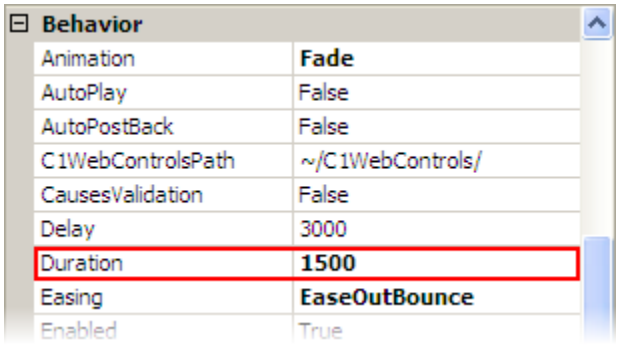
Complete the following steps to set animation effects:

1. Click **C1MultiPage**'s smart tag (▶) to open the **C1MultiPage Tasks** menu and select **MultiPage Designer**.
   The **C1MultiPage Designer Form** appears.
2. Use the **Add Child Item** button  to add two pages to your **C1MultiPage**.

   Note: In order to scroll through the **C1MultiPage** control at run-time, you must add navigation buttons to the control. For more information, see Using the Toolbar for Navigation (page 46).
3. In treeview, select **C1MultiPage1** to reveal a list of its properties and then complete the following:
   - Set the Animation property to **Fade**.



- Set the Easing property to **EaseOutBounce**. This property determines the animation transition effect.
- Set the Duration property to "1500". This will lengthen the duration of the animation effect, guaranteeing that you will notice the effect when you run the project.



4. Click **OK** to close the **C1MultiPage Designer** Form.
5. Run the project and switch to the second page of the C1MultiPage. Observe that the new page fades in and then bounces before settling into its natural state.

Setting Animation Effects in Source View

In Source view add `Animation="Fade"`, `Duration="1500",` and `Easing="EaseOutBounce"` to the `<cc1:C1MultiPage>` tag so that it appears similar to the following:

```
<cc1:C1MultiPageID="C1MultiPage1" runat="server" Animation="Fade"
Duration="1500" Easing="EaseOutBounce">
```

Run the project and switch to the second page of the C1MultiPage. Observe that the new page fades in and then bounces before settling into its natural state.

Setting Animation Effects in Code

Complete the following steps to set animation effects in code:

1. Import the following namespace into your project:
   - Visual Basic
     ```
     Imports C1.Web.UI
     Imports C1.Web.UI.Controls;
     ```

- C#
```
using C1.Web.UI;
using C1.Web.UI.Controls;
```

2. Add the following code to the **Page_Load** event to choose the animation:

   - Visual Basic
```
C1MultiPage1.Animation = PageSwitchEffect.Fade
```
   - C#
```
C1MultiPage1.Animation = PageSwitchEffect.Fade;
```

3. Set the duration of the animation:

   - Visual Basic
```
C1MultiPage1.Duration = 1500
```
   - C#
```
C1MultiPage1.Duration = 1500;
```

4. Select an animation transition effect:

   - Visual Basic
```
C1MultiPage1.Easing = Easing.EaseOutBounce
```
   - C#
```
C1MultiPage1.Easing = Easing.EaseOutBounce;
```

5. Run the project and switch to the second page of the C1MultiPage. Observe that the new page fades in and then bounces before settling into its natural state.