ComponentOne

XmlEditor for WinForms

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 XmlEditor for WinForms Overview

Load, save, and edit your XHTML documents with **ComponentOne XmlEditor for WinForms**. Choose from a WYSIWYG **Design** mode, **Source** mode, or **Preview** mode to edit or view your documents. The C1XmlEditor control is bound to the XHTML document, so you can make changes directly to your XHTML document from within the control. Editing XHTML documents has never been so easy!

Getting Started

To get started, review the following topics:

- Key Features (page 11)
- Quick Start (page 17)
- Samples (page 34)

Installing XmlEditor for WinForms

The following sections provide helpful information on installing **ComponentOne XmlEditor for WinForms**.

XmlEditor for WinForms Setup Files

The ComponentOne Studio for WinForms installation program will create the following directory: C:\Program Files\ComponentOne\Studio for WinForms. This directory contains the following subdirectories:

bin	Contains copies of all ComponentOne binaries (DLLs, EXEs).
H2Help	Contains online documentation for all Studio components.
C1XmlEditor	Contains samples and tutorials for **ComponentOne XmlEditor for WinForms**.

Samples

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

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

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

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

Common	Contains support and data files that are used by many of the demo programs.

C1XmlEditor	Contains samples and tutorials for **XmlEditor for WinForms**.

System Requirements

System requirements include the following:

Operating Systems:	Windows® 2000 Windows Server® 2003 Windows Server 2008 Windows XP SP2 Windows Vista™ Windows 7
Environments:	.NET Framework 2.0 or later C# .NET Visual Basic .NET
Disc Drive:	CD or DVD-ROM drive if installing from CD

Installing Demonstration Versions

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

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

Uninstalling XmlEditor for WinForms

To uninstall **ComponentOne XmlEditor for WinForms**:

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

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

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

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

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

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

Using licensed components in console applications

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

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

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

Using licensed components in Visual C++ applications

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

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

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

Using licensed components with automated testing products

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

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

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

Note that the AssemblyConfiguration string may contain additional text before or after the given string, so the AssemblyConfiguration attribute can be used for other purposes as well. For example:
```
[AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime,BetaVersion")]
```

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

Troubleshooting

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

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

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

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

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

If that fails follow these steps:

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

Alternatively, follow these steps:

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

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

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

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

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

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

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

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

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

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

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

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

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

Option 2 – Continue to use the components you have.

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

Technical Support

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

Some methods for obtaining technical support include:

- **Online Support via HelpCentral**
  ComponentOne HelpCentral provides customers with a comprehensive set of technical resources in the form of FAQs, samples, Version Release History, Articles, searchable Knowledge Base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**
  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.
- **Peer-to-Peer Product Forums and Newsgroups**
  ComponentOne peer-to-peer product forums and newsgroups are available to exchange information, tips, and techniques regarding ComponentOne products. ComponentOne sponsors these areas as a forum for users to share information. While ComponentOne does not provide direct support in the forums and newsgroups, we periodically monitor them to ensure accuracy of information and provide comments when appropriate. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.
- **Installation Issues**
  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.
- **Documentation**
  ComponentOne documentation is installed with each of our products and is also available online at

HelpCentral. If you have suggestions on how we can improve our documentation, please email the Documentation team. Please note that e-mail sent to the Documentation team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

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

Redistributable Files

ComponentOne XmlEditor for WinForms 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.Win.XmlEditor.2.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 namespace for the C1XmlEditor component is **C1.Win.C1XmlEditor**. The following code fragment shows how to declare a C1XmlEditor component using the fully qualified name for this class:

- Visual Basic
```
Dim XmlEditor1 As C1.Win.C1XmlEditor.C1XmlEditor
```
- C#
```
C1.Win.C1XmlEditor.C1XmlEditor XmlEditor1;
```

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*.

For example, if you create a new class named C1XmlEditor, you can use it inside your project without qualification. However, the C1XmlEditor assembly also implements a class called **C1XmlEditor**. So, if you want to use the C1XmlEditor class in the same project, you must use a fully qualified reference to make the reference unique. If the reference is not unique, Visual Studio .NET produces an error stating that the name is ambiguous. The following code snippet demonstrates how to declare these objects:

- Visual Basic

```
' Define a new C1XmlEditor object
Dim MyXmlEditor as C1XmlEditor
' Define a new C1XmlEditor.C1XmlEditor object.
Dim C1XmlEditor as C1.Win.XmlEditor.C1XmlEditor
```

- C#

```
// Define a new C1XmlEditor object
C1XmlEditor MyXmlEditor;
// Define a new C1XmlEditor.C1XmlEditor object.
C1.Win.XmlEditor.C1XmlEditor C1XmlEditor;
```

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 C1XmlEditor = C1.Win.XmlEditor
Imports MyXmlEditor = MyProject.C1XmlEditor

Dim s1 As C1XmlEditor
Dim s2 As MyXmlEditor
```

- C#

```
using C1XmlEditor = C1.Win.XmlEditor;
using MyXmlEditor = MyProject.C1XmlEditor;

 C1XmlEditor s1;
 MyXmlEditor s2;
```

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 a .NET Project

To create a new .NET project, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio .NET, select **New Project**. The **New Project** dialog box opens.
2. Under **Project Types**, choose either **Visual Basic** or **Visual C#**. Note that one of these options may be located under **Other Languages**.
3. Select **Windows Application** from the list of **Templates** in the right pane.

4. Enter or browse for a location for your application in the **Location** field and click **OK**. A new Microsoft Visual Studio .NET project is created in the specified location. In addition, a new Form1 is displayed in the Designer view.
5. Double-click the **C1XmlEditor** component from the Toolbox to add it to Form1. For information on adding a component to the Toolbox, see Adding the XmlEditor for WinForms Components to a Project (page 10).

Adding the XmlEditor for WinForms Components to a Project

When you install ComponentOne Studio for WinForms, the **Create a ComponentOne Visual Studio Toolbox Tab** checkbox is checked, by default, in the installation wizard. When you open Visual Studio, you will notice a **ComponentOne Studio for WinForms** tab containing the ComponentOne controls has automatically been added to the Toolbox.

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

XmlEditor for WinForms provides the following controls:

- C1XmlEditor
- C1XmlEditorToolStripMain
- C1XmlEditorToolStripObjects
- C1XmlEditorToolStripStyle
- C1XmlEditorToolStripTable

To use these controls, add them to the form or add a reference to the C1.Win.C1XmlEditor assembly to your project.

Adding XmlEditor for WinForms Controls to the Toolbox

To add **XmlEditor for WinForms** 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 **XmlEditor for WinForms** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1XmlEditor**, 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 the component belonging to namespace C1.Win.C1XmlEditor. Note that there may be more than one component for each namespace.

Adding XmlEditor for WinForms Controls to the Form

To add **XmlEditor for WinForms** controls to a form:

1. Add XmlEditor for WinForms controls to the Visual Studio Toolbox.
2. Double-click a control or drag it onto your form.

Adding a Reference to the C1.Win.C1XmlEditor Assembly

To add a reference to the assembly:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne C1XmlEditor** assembly from the list on the **.NET** tab or browse to find the C1.Win.C1XmlEditor.2.dll file and click **OK**.

3. Double-click the form caption area to open the code window. At the top of the file, add the following **Imports** statements (**using** in C#):

```
Imports C1.Win.C1XmlEditor
```

Note: This makes the objects defined in the C1.Win.C1XmlEditor assembly visible to the project. See Namespaces (page 8) for more information.

Key Features

- **"On the fly" synchronizing with XmlDocument**

  Bind the C1XmlEditor control to a document specified in the Document property. If the document is edited within the C1XmlEditor, the underlying XmlDocument syncs to match it. If the XmlDocument changes in code, these changes are visible in the C1XmlEditor control at run time. See Binding C1XmlEditor to a Document (page 36) for more information.

- **C1XmlEditor provides three edit modes: Design, Source, and Preview**

  The C1XmlEditor control features three editor modes: **Design**, **Source**, and **Preview**. You can determine which of these views users will see initially by setting the Mode property. See the C1XmlEditor Overview (page 14) for more information.

- **Load and save Xhtml documents from or to a file or stream**

  You can load an XHTML document into C1XmlEditor from a file, stream or XML string. You can save an XHTML document to a file or stream. See the XmlEditor for WinForms Task-Based Help (page 35) for examples on how to do this.

- **XmlEditor for WinForms offers Cascading Style Sheet support, including easy-to-define custom CSS styles for Design and Preview mode**

  C1XmlEditor fully supports cascading style sheets (CSS) in edited documents. In addition, you can specify external CSS rules in CSS files which will be used only in **Design** or **Preview** mode. The LoadDesignCSS and LoadPreviewCSS methods support this feature by loading a cascading style sheet from a file or stream.

  See Using a Cascading Style Sheet with C1XmlEditor (page 39) for an example on using the LoadDesignCSS method.

- **Code clean-up routines**

  On document loading, switching off **Source** mode, or on executing the ValidateSource or FixSource methods, C1XmlEditor automatically removes empty tags, closes unclosed tags, and generally improves messy or unreadable HTML or XHTML code, converting it to valid XHTML.

- **XmlEditor for WinForms' built-in spell checker allows you to check as you type**

  Spell-checking functionality is provided by ComponentOne's **C1SpellChecker** component.

  C1XmlEditor fully supports **C1SpellChecker** so you can use all of its great features, including: modal **Dialog mode** (users can choose to correct or ignore errors through a dialog box), **As-you-type mode** (spelling errors are indicated by a red, wavy underline), and the **AutoReplace** feature (misspelled words are automatically corrected as you type). In **As-you-type mode**, the built-in C1XmlEditor context menu merges with the **C1SpellChecker** context menu so you can see and select all available commands.

- **Ability to add custom tags in DTD**

Advanced programming tasks sometimes require using additional DTD elements in the edited document. You can enter the elements, or tags, in the document specifying them using the XmlExtensions property in special XML format. See the **CustomTags** sample installed with this product for a complete example of this feature.

- **Access and manage data from code selections and the caret position**

  You can access content in the C1XmlEditor by specifying a range of characters to select. For an example, see Selecting Characters in the C1XmlEditor (page 38).

- **Text decoration and block formatting commands**

  You can easily set font and text decoration and block formatting properties in a text block without worrying about how to modify the underlying XmlDocument. Use the C1TextRange class to identify target text and the C1TextRange methods to apply the decoration or formatting: ApplyTag, ApplyClass, ApplyStyle, and **ApplyFormatting**.

- **Use built-in or custom dialog boxes to insert links, pictures, tables, and other objects**

  You can show built-in or custom dialog boxes to insert or edit various objects at the current selection. The dialog boxes allow you to specify all properties of the inserted or edited object. For example, the **Picture** dialog box contains fields to select the image source, or file name, alternate text, size, and so on. C1XmlEditor also has a built-in **Find and Replace** dialog box that allows the user to specify a string to search for and a replacement string, as well as the options to use when searching for text in a document. If you prefer, you can create and use your own find and replace dialog box, specifying it in the CustomDialogs property of the C1XmlEditor.

  For more information on **C1XmlEditor**'s built-in dialog boxes, or for steps on how to use your own, see C1XmlEditor Dialog Boxes (page 22).

- **Use the mouse to move or resize pictures or tables**

  You can interact with objects directly in **Design** mode by specifying their size or position with the mouse.

  Moving is a direct manipulation. You can move an object within a document in any direction. C1XmlEditor will move relative to the XmlNode and/or set the new position attributes of the object automatically.

  Resizing allows you to resize the selected object in the direction of the mouse pointer movement. **C1XmlEditor** will set the new size attributes of the object automatically.

- **Printing support**

  Calling the Print method has the same effect as choosing **Print** from the Windows Internet Explorer **File** menu. The Print method can activate the **Print** dialog box, prompting the user to change print settings.

  The PrintPreview method lets you see the Web page before you print it so you can avoid printing mistakes.

  See the **PrintTemplate** sample installed with this product for a complete example of this feature.

- **Clipboard support**

  You can select text, tables or graphics and use the Cut or Copy methods of C1XmlEditor to move your selection to the Clipboard. Then you can paste the selection into another program. You can copy HTML from a program or even a picture in Internet Explorer and then use the Paste method to put it into the C1XmlEditor.

  The PasteAsText method automatically formats the text you paste into the C1XmlEditor as plain text.

  C1XmlEditor also supports keyboard shortcuts such as CTRL+C (copy), CTRL+X (cut), and CTRL+V (paste).

  To determine what clipboard operations are allowed, use the CanCut, CanCopy, CanPaste, and CanPasteAsText properties.

- **History of changes (undo/redo)**

  C1XmlEditor has an unlimited undo history mechanism. You can programmatically access the editing history, calling the Undo or Redo methods. The **Undo** and **Redo** methods also support keyboard shortcuts: CTRL+Z (undo) and CTRL+Y (redo).

  By undoing repeatedly, a user can gradually work back to the point before the error was made.

XmlEditor for WinForms Elements

ComponentOne XmlEditor for WinForms includes the following controls:

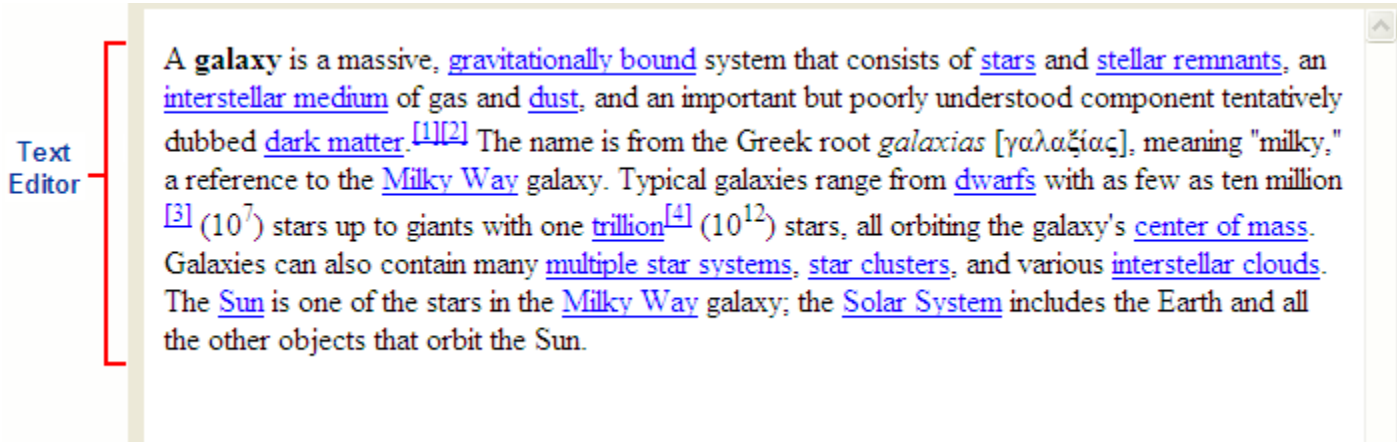
- C1XmlEditor (page 14)
- C1XmlEditorToolStripMain (page 16)
- C1XmlEditorToolStripObjects (page 16)
- C1XmlEditorToolStripStyle (page 16)
- C1XmlEditorToolStripTable (page 17)

The **C1XMLEditorToolStrips** can be added to your form and linked to the C1XmlEditor control through the Editor property. See Linking a ToolStrip to C1XmlEditor (page 37) for more information.

You also have the option of implementing your own ToolStrip using the C1XmlEditorToolStripBase as the base class. You can then add an C1XmlEditorToolStripButton or C1XmlEditorToolStripComboBox to a standard ToolStrip and set the Editor and Command properties. See Creating a Custom ToolStrip (page 37) for more information.

C1XmlEditor Overview

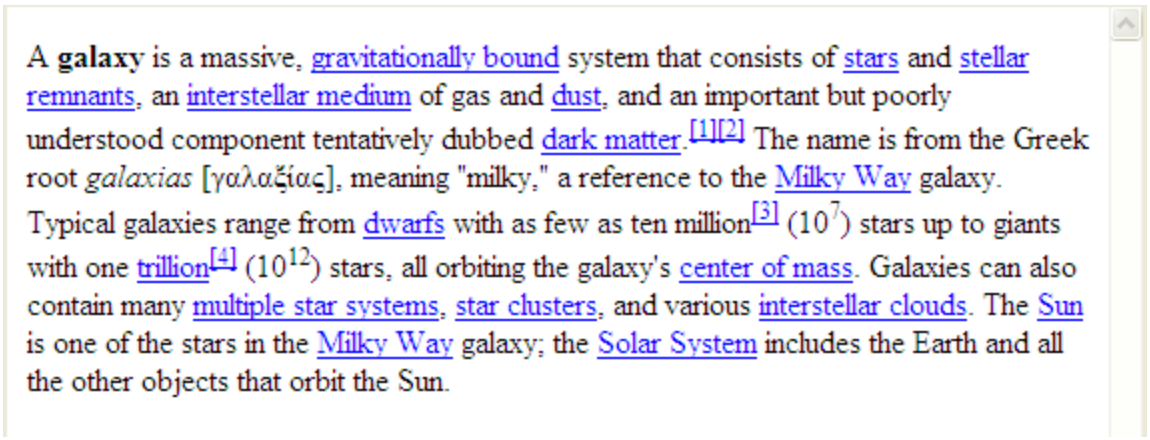
The C1XmlEditor control is a text editor that can be bound to an Xhtml document. Any document bound to the C1XmlEditor control can be loaded, saved or edited.



The C1XmlEditor control features three editor modes: **Design**, **Source**, and **Preview**. You can determine which of these views users will see initially by setting the Mode property.

- **Design View**

  This view displays the text editor's content in a What-You-See-Is-What-You-Get (WYSIWYG) format. Here you can add content without markup. The following image shows the C1XmlEditor in **Design** view:

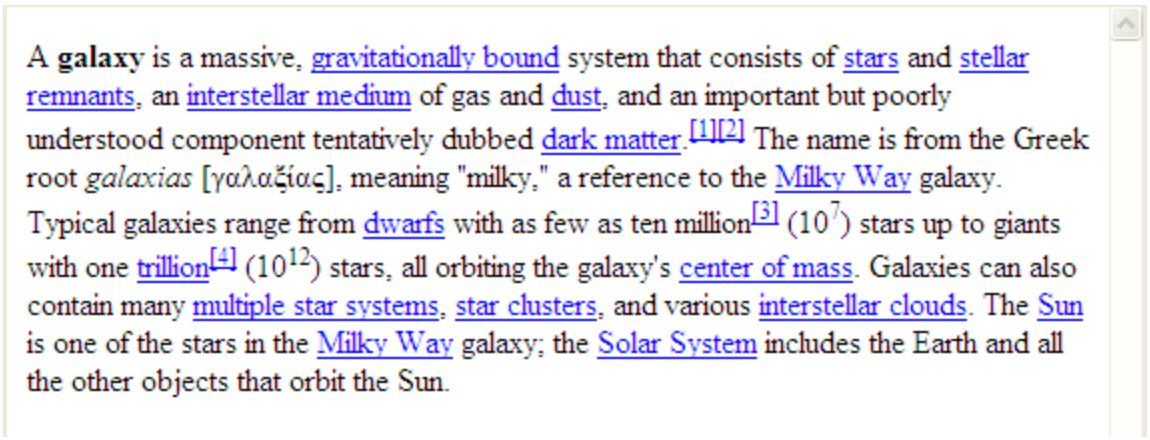


- **Source View**

  This view provides a hand-coding environment for writing and editing HTML markup. The following image shows the text editor in **Source** view:

- **Preview View**

  In this mode, no editing is allowed; **Preview** mode is strictly for viewing the content. The following image shows the text editor in **Preview** mode:



You can set the initial run-time view of the text editor by setting the Mode property. The EditorMode enumeration can be set to one of three settings: **Design**, **Source**, or **Preview**. For more information on how to set the editing mode, see Changing the C1XmlEditor Editor Mode (page 35).

Scrollbars will automatically appear if content added to the text editor exceeds the available screen space.

C1XmlEditorToolStripMain Overview

The C1XmlEditorToolStripMain control is a standard text-style ToolStrip made up of C1XmlEditorToolStripButtons allowing you to: create, open, save or print a new file; cut, copy, and paste text; undo or redo an action; select all text or find and replace text; switch between design and source view; or preview the Xml document.

C1XmlEditorToolStripObjects Overview

The C1XmlEditorToolStripObjects control is made up of C1XmlEditorToolStripButtons allowing you to insert objects, including tables, pictures, hyperlinks, bookmarks, and flash movies.

C1XmlEditorToolStripStyle Overview

The C1XmlEditorToolStripStyle control is made up of a C1XmlEditorToolStripComboBox and C1XmlEditorToolStripButtons allowing you to: set paragraph or heading styles, create bulleted or numbered lists; bold, italic, underline, or strikethrough your text; clear formatting; create subscript or superscript text; change the font size; or change paragraph alignment.

C1XmlEditorToolStripTable Overview

The C1XmlEditorToolStripTable control is made up of C1XmlEditorToolStripButtons allowing you to: insert a table; set table, row, column, and cell properties; insert a row above or below the selected row; insert a column before or after the selected column; and delete a table, row, or column.

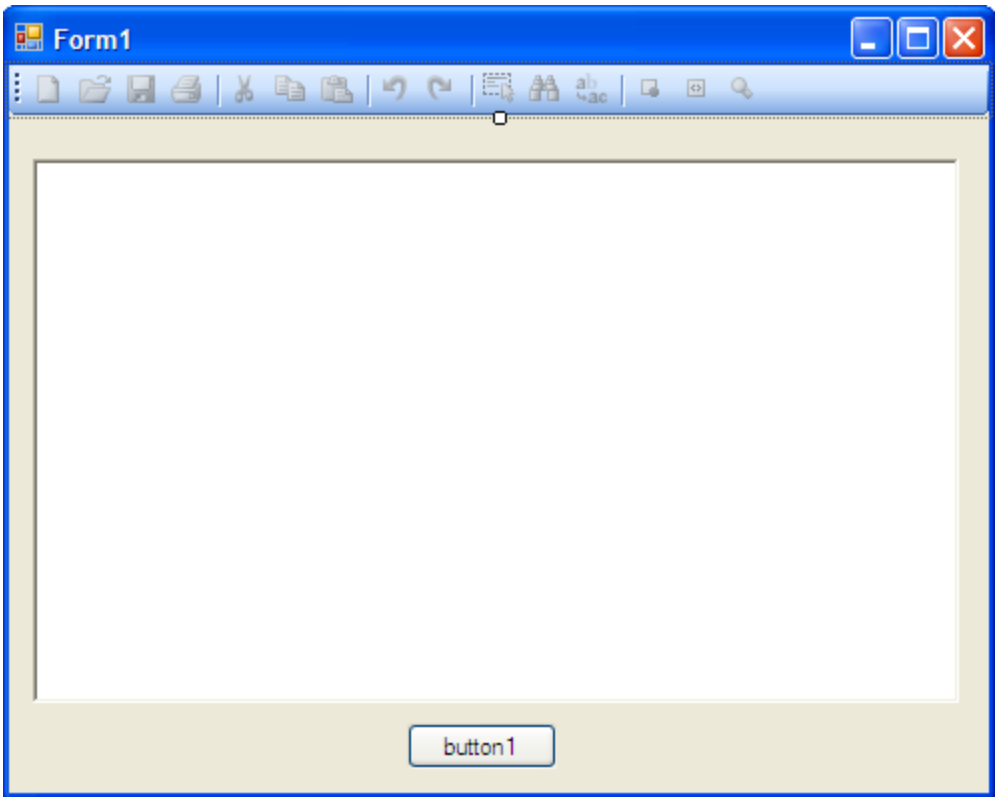
XmlEditor for WinForms Quick Start

This section details some of the features of **ComponentOne XmlEditor for WinForms**. This quick start will walk through the steps of adding a C1XmlEditor and C1XmlEditorToolStripMain control to your project, binding the editor to a document, applying a cascading style sheet, and using some of the buttons on the C1XmlEditorToolStripMain toolbar.

Step 1 of 4: Adding XmlEditor for WinForms Components to the Form

In this topic you will add the C1XmlEditor and C1XmlEditorToolStripMain controls to your form and create a basic application.

1. Create a new Windows application (page 9).
2. While in Design view, double-click the C1XmlEditor and C1XmlEditorToolStripMain controls in the Visual Studio Toolbox to add them to your form.
3. Add one button to the form and arrange the controls so they look like the following image:



4. Right-click the C1XmlEditorToolStripMain control on your form and select **Properties**.
5. In the Visual Studio Properties window, click the drop-down arrow next to the Editor property and select **c1XmlEditor1**. This links the toolstrip to the C1XmlEditor control.
6. Select **button1** and enter **Apply CSS** next to the **Text** property in the Properties window.

In the next step you will bind C1XmlEditor to a document.

Step 2 of 4: Binding C1XmlEditor to a Document

Now you can bind C1XmlEditor to a document that can be saved to a file and loaded later when needed. If this document is edited within the C1XmlEditor, the underlying XmlDocument syncs to match it.

1. Click the **View** menu and select **Code** to switch to code view.
2. Add the following Imports (Visual Basic) or using (C#) statements to your project so you can use abbreviated names (page 8).

   - Visual Basic
   ```
   Imports System.Xml
   Imports C1.Win.XmlEditor
   ```
   - C#
   ```
   using System.Xml;
   using C1.Win.XmlEditor;
   ```
3. Create a **Form_Load** event and add the following code there to create a new document and bind it to C1XmlEditor:

   - Visual Basic
   ```
   Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
   ```

```
    Dim myDoc As New XmlDocument()
    c1XmlEditor1.Document = myDoc
End Sub
```

- C#

```
private void Form1_Load(object sender, EventArgs e)
{
  XmlDocument myDoc = new XmlDocument();
  c1XmlEditor1.Document = myDoc;
}
```

In the next step you will add code to apply a cascading style sheet to the C1XmlEditor content.

Step 3 of 4: Applying a Cascading Style Sheet

In this topic you will add the code that allows you to use a cascading style sheet (CSS) when the button is clicked.

1. In the Visual Studio **File** menu, select **New | File**. The **New File** dialog box appears.
2. Select **General** under *Categories* and choose **Style Sheet** under *Templates*.
3. Click **Open** and add this markup to the style sheet so it looks like the following:

```
body
{
      font-family: Verdana;
      font-size: 10pt;
      line-height: normal;
      margin-bottom: 0pt;
      margin-left: 0pt;
      margin-right: 0pt;
      margin-top: 0pt;
      color: Fuchsia;
}
h1 {
      font-family: Verdana;
      font-size: 20pt;
      font-weight: bold;
      line-height: normal;
      margin-bottom: 8pt;
      margin-left: 0pt;
      margin-right: 0pt;
      margin-top: 10pt;
}
h2 {
      font-family: Verdana;
      font-size: 16pt;
      font-weight: bold;
      line-height: normal;
      margin-bottom: 7pt;
      margin-left: 0pt;
      margin-right: 0pt;
      margin-top: 9pt;
      page-break-after: avoid;
}
h3 {
      font-family: Verdana;
      font-size: 16pt;
      font-weight: bold;
```

```
      line-height: normal;
      margin-bottom: 7pt;
      margin-left: 0pt;
      margin-right: 0pt;
      margin-top: 9pt;
      page-break-after: avoid;
}
h4 {
      font-family: Verdana;
      font-size: 12pt;
      font-weight: bold;
      line-height: normal;
      margin-bottom: 2pt;
      margin-left: 0pt;
      margin-right: 0pt;
      margin-top: 2pt;
      page-break-after: avoid;
}
.C1HBullet {
      font-family: Verdana;
      font-size: 10pt;
        font-style: italic;
      line-height: 14pt;
      margin-bottom: 0pt;
      margin-left: 18pt;
      margin-right: 0pt;
      margin-top: 5pt;
      text-indent: -18pt;
}
p {
      font-family: Verdana;
      font-size: 10pt;
      line-height: 14pt;
      margin-bottom: 0pt;
      margin-left: 0pt;
      margin-right: 0pt;
      margin-top: 5pt;
        text-indent: 18pt;
}
.C1SectionCollapsed {
      font-weight: bold;
}
```

4. Click the **Save** button and save the style sheet as *myCSS.css*, for example.
5. Right-click the project name in the Solution Explorer and select **Add | Existing Item**.
6. Choose the CSS and click **Add**.
7. Create a **Button1_Click** event and add the following code there so the cascading style sheet is applied when the button is clicked. The location may be different, depending on where you save your Visual Studio project.

   - Visual Basic

```
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    c1XmlEditor1.LoadDesignCSS("C:\myCSS.css")

End Sub
```

- C#

```
private void button1_Click(object sender, EventArgs e)
        {
           c1XmlEditor1.LoadDesignCSS(@"C:\myCSS.css");
        }
```

In the next step you will run the project and add text to the editor.

Step 4 of 4: Running the Project

Press **F5** to run the project and follow these steps:

1. Add some text to the editor.
2. Click the **Apply CSS** button. Notice the cascading style sheet is applied to the text.
3. In the C1XmlEditorToolStripMain control, click the **Source view** button to view the code and then click the **Design view** or **Preview** button to switch editor modes.
4. Click the **Find** button to search for an item in the text. This opens the **Find and Replace** dialog box.
5. Click the **Save** button and enter a name for your document. You can open it again later by clicking the **Open file** button.

Congratulations! You have successfully completed the **XmlEditor for WinForms Quick Start**.

C1XmlEditor Design-Time Support

The following sections describe how to use **C1XmlEditor**'s design-time environment to configure the C1XmlEditor control.

Smart Tags

The **XmlEditor for WinForms** controls include a smart tag () in Visual Studio. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in the controls.

To access the **Tasks** menu, click the smart tag in the upper-right corner of the control. For example, here is the **C1XmlEditor Tasks** menu:

The **Tasks** menus is similar for all **XmlEditor for WinForms** controls and it operates as follows:

- **About C1XmlEditor**

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

Context Menus

The **XmlEditor for WinForms** controls have additional commands available on their context menus that Visual Studio provides for all .NET and ASP.NET controls.

Right-click anywhere on one of the controls to display the context menu. The context menu commands operate as follows:

- **About C1XmlEditor**

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

C1XmlEditor Run-Time Elements

The following topics provide information regarding the run-time environment of the C1XmlEditor control.

C1XmlEditor Dialog Boxes

Within the C1XmlEditor control are several dialog boxes that users can employ to edit Xhtml documents.

You can easily show a dialog box when a button is clicked, for example, simply by using the ShowDialog method and specifying the DialogType.

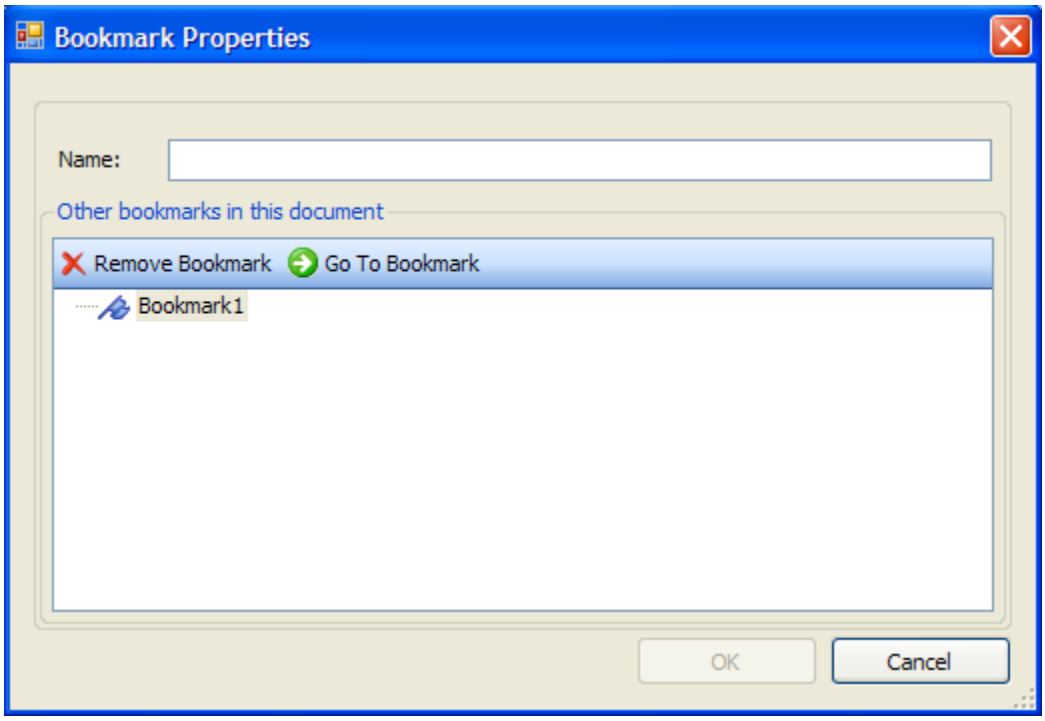
For example, to show the **PageSetup** dialog box when a button is clicked, you would use the following code:

```
  private void button7_Click(object sender, EventArgs e)
        {

c1XmlEditor1.ShowDialog(C1.Win.XmlEditor.DialogType.PageSetup);

        }
```

The following topics detail the dialog boxes that can be accessed through the C1XmlEditor control. In some cases you may need to show your own customized versions of the dialog boxes. See Using a Custom Dialog Box (page 32) for steps on how to do this.

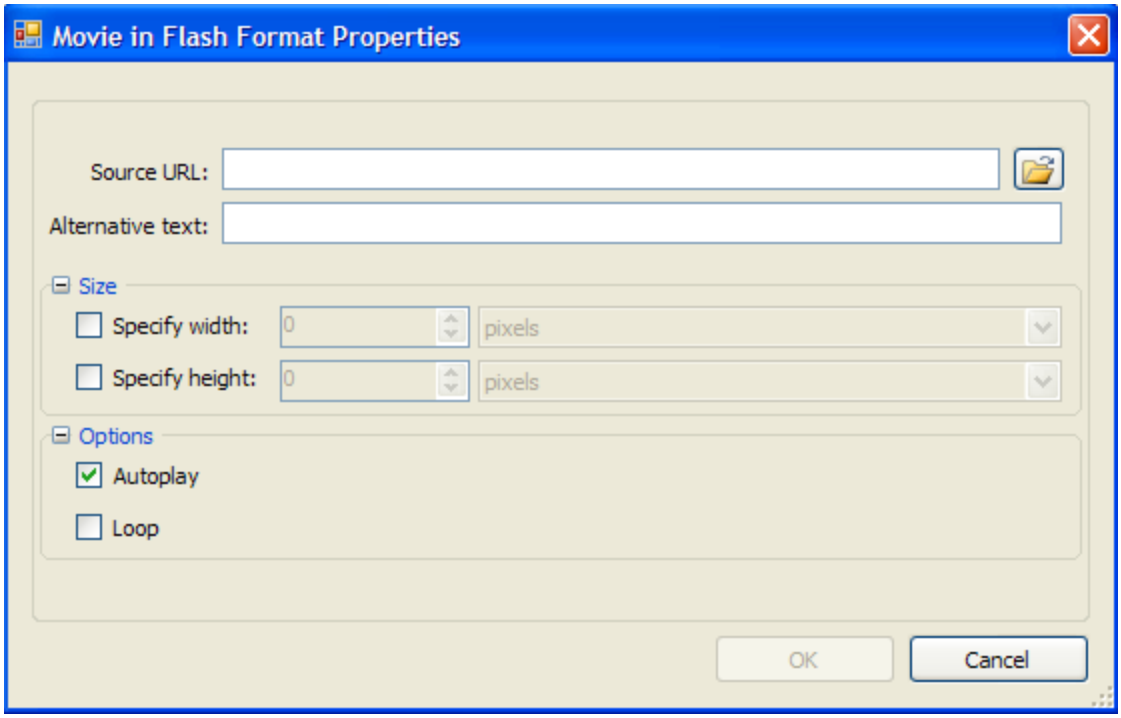
Bookmark Properties Dialog Box

In the **Bookmark Properties** dialog box, users can create a new bookmark by entering a name in the **Name** text box. They can also jump to or remove an existing bookmark.



Movie in Flash Format Properties Dialog Box

In the **Movie in Flash Format Properties** dialog box, users can specify the Flash movie to be inserted in the document as well as set the window size and play options.



Source URL	Click the **Browse** button to locate a Flash movie.

Alternative text	Enter the text that should appear in case the movie is unavailable.

Size

Specify width	Select this checkbox and enter a number to specify the movie window width. Choose pixels or percent.
Specify height	Select this checkbox and enter a number to specify the movie window height. Choose pixels or percent.

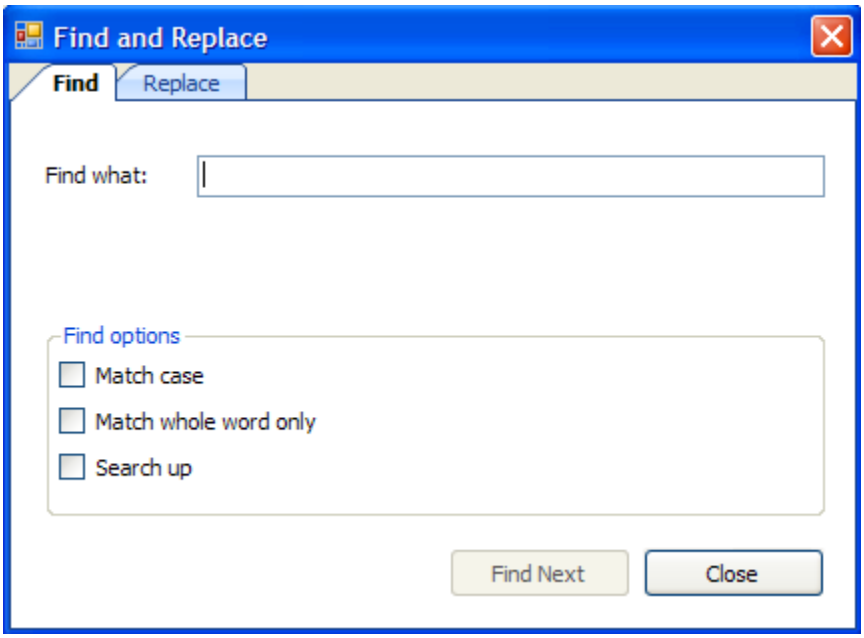
Options

Autoplay	Select this check box to automatically start the movie when the document is loaded.
Loop	Select this check box to continuously play the movie.

Find and Replace Dialog Box

The **Find** and **Replace** dialog boxes are the same dialog box with two tabs allowing you to find and/or replace text.

In the **Find and Replace** dialog box, users can find text and replace it with other text if they choose. To access the **Find and Replace** dialog box, click inside the C1XmlEditor control and click CTRL+F.



To find specific text, enter the text in the **Find what** text box, select one of the check boxes under **Find options**, and click **Find Next**.

To replace specific text, enter the text to be replaced in the **Find what** text box, enter the text to replace it in the **Replace with** text box, and click **Replace** to replace certain instances or click **Replace All** to replace all instances.

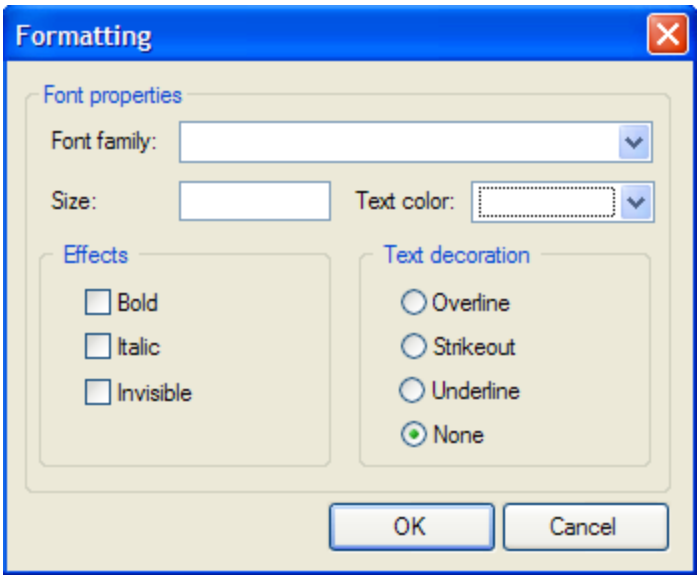
Find Options

The **Find options** section of the **Find and Replace** dialog box allows you to specify certain search criteria as described in the following table:

Option	Description
Match case	Locates text that exactly matches the combination of uppercase and lowercase letters you type in the **Find what** box.
Match whole word only	Locates distinct occurrences of words, not groups of characters inside words.
Search up	Searches for text specified in the **Find what** box from the bottom of the control upwards.

Formatting Dialog Box

In the **Formatting** dialog box, users can specify the font family, size, color, text effects, and decoration.



Font properties

Font family	Click the drop-down arrow next to **Font family** to select from the list of available font families.
Size	Enter a number to specify the text size.
Text color	Click the drop-down arrow next to **Text color** to select from Web, System, or custom colors.

Effects

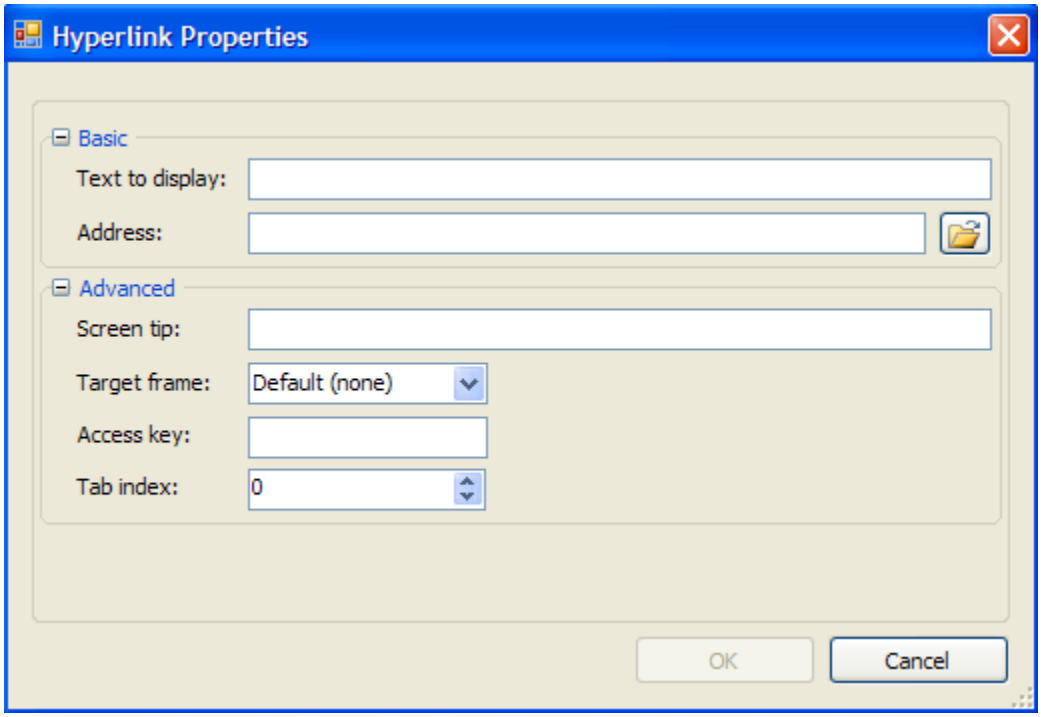
Select the **Bold**, **Italic**, or **Invisible** check boxes to add the effect to the text. If **Invisible** is selected, the text will be hidden in **Preview** mode.

Text decoration

Select **Overline**, **Strikeout**, **Underline**, or **None** (default) to add decoration to the text.

Hyperlink Properties Dialog Box

In the **Hyperlink Properties** dialog box, users can enter a URL, specify the text to display, add a tooltip, specify the frame where the Web page will open, create an access key, and enter a tab index.



Basic

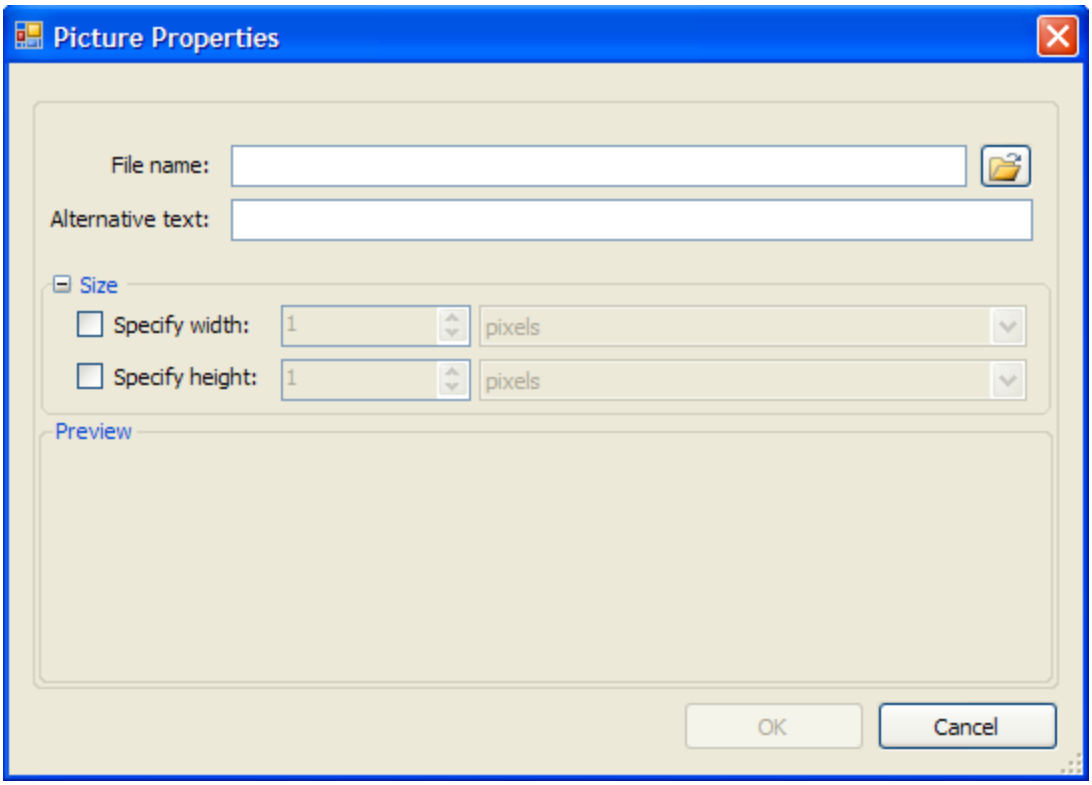
Text to display	Enter the text that will appear as a hyperlink in the document.
Address	Enter the URL address for the hyperlink.

Advanced

Screen tip	Enter the tooltip text that will appear when a user hovers over the hyperlink.
Target frame	Click the drop-down arrow to select a frame where the web page will appear when the hyperlink is clicked.
Access key	Enter a key that can be pressed with the ALT key to jump to a specific control on the page without using the mouse.
Tab index	Enter a number to define the tab order for the hyperlink.

Picture Properties Dialog Box

Users can insert and preview a picture using the **Picture Properties** dialog box.



File name	Click the Browse button to locate a picture to insert.
Alternative text	Enter the text that should appear in case the image is unavailable.

Size

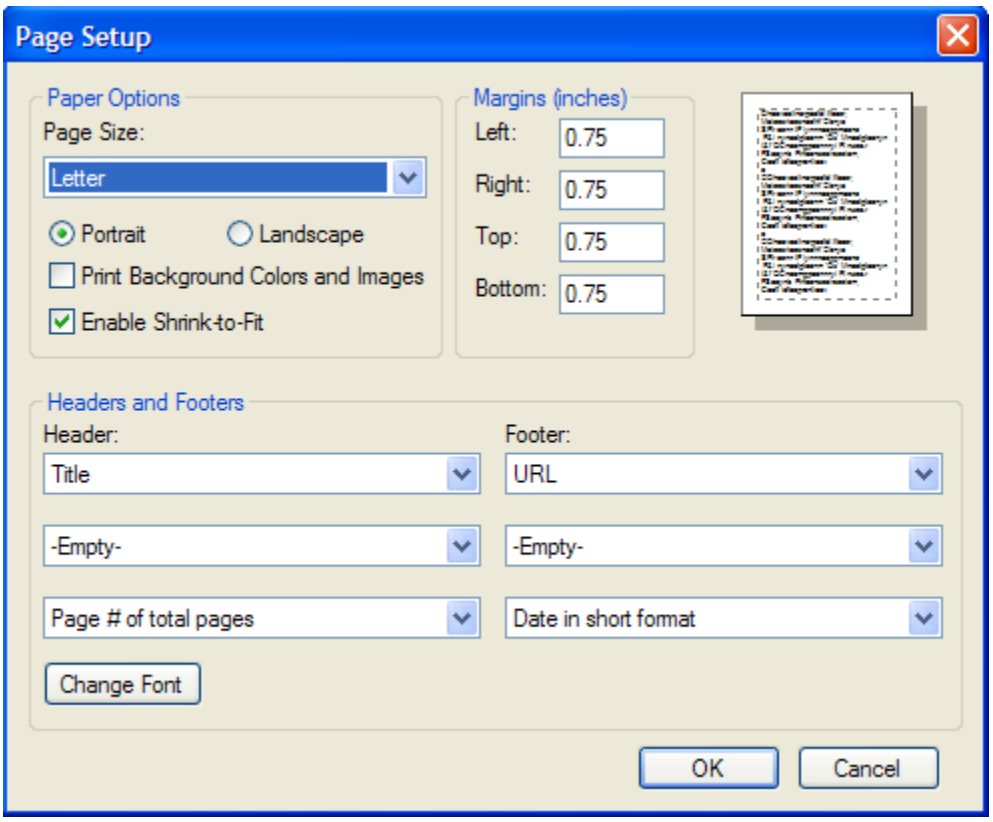
Specify width	Select this checkbox and enter a number to specify the image width. Choose pixels or percent.
Specify height	Select this checkbox and enter a number to specify the image height. Choose pixels or percent.

Preview

A preview of the image appears in this area.

Page Setup Dialog Box

In the **Page Setup** dialog box, users can specify the page size and orientation, set page margins, and create headers and footers.



Paper Options

Page Size	Click the drop-down arrow and select a paper size.
Portrait or **Landscape**	Select **Portrait** or **Landscape** to determine the page orientation.
Print Background Colors and Images	Select this checkbox to print background colors and pictures.
Enable Shrink-to-Fit	Select this checkbox to scale the content to fit on the page.

Margins

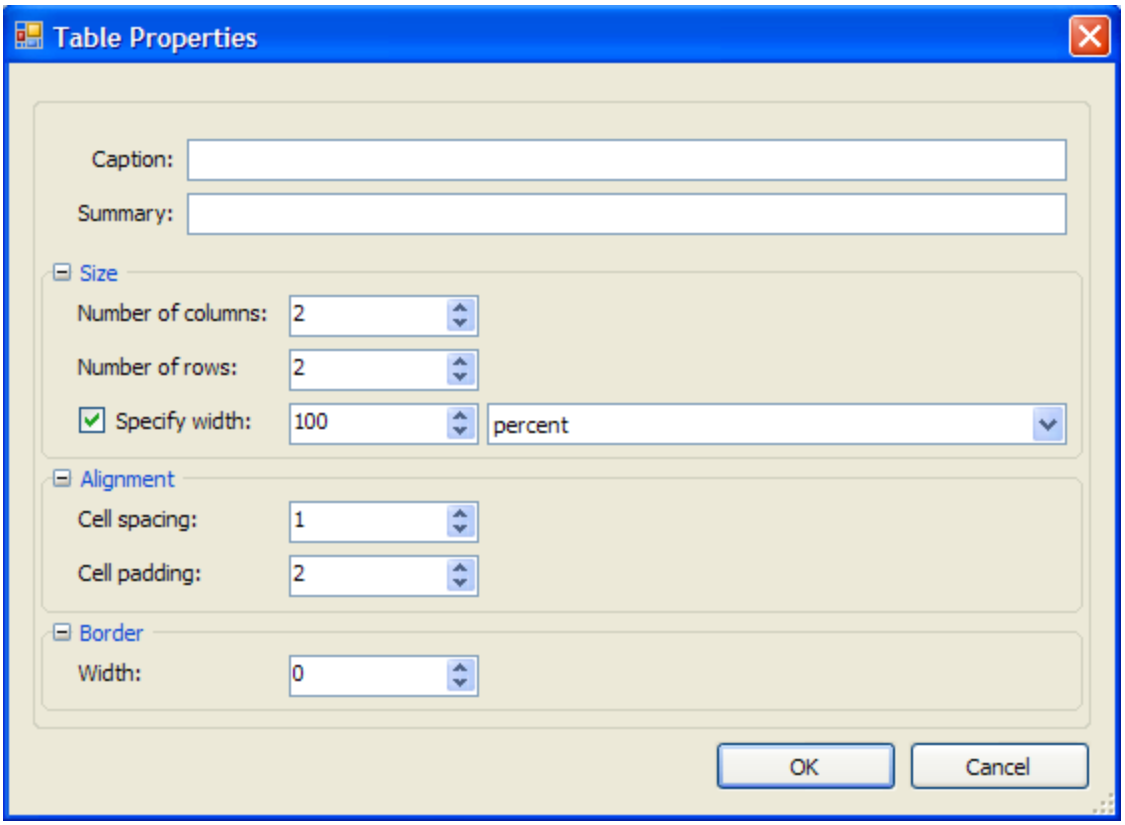
Specify the page margins in the **Left**, **Right**, **Top**, and **Bottom** boxes.

Headers and Footers

Click the drop-down arrows under **Header** or **Footer** to select the header or footer to be inserted, or create your own custom header or footer. Click the **Change Font** button to change the header or footer font, style, and size.

Table Properties Dialog Box

In the **Table Properties** dialog box, users create a new table, specify the number of columns and rows, set up cell spacing and padding, create a border, and add a caption for the table. Right-click the table to access additional properties for the row, column, or cell.



Caption	Enter the caption to appear above the table.
Summary	Enter a description for the table.

Size

Number of columns	Enter the number of table columns.
Number of rows	Enter the number of table rows.
Specify width	Select this checkbox to specify the width of the table, either in percent or pixels.

Alignment

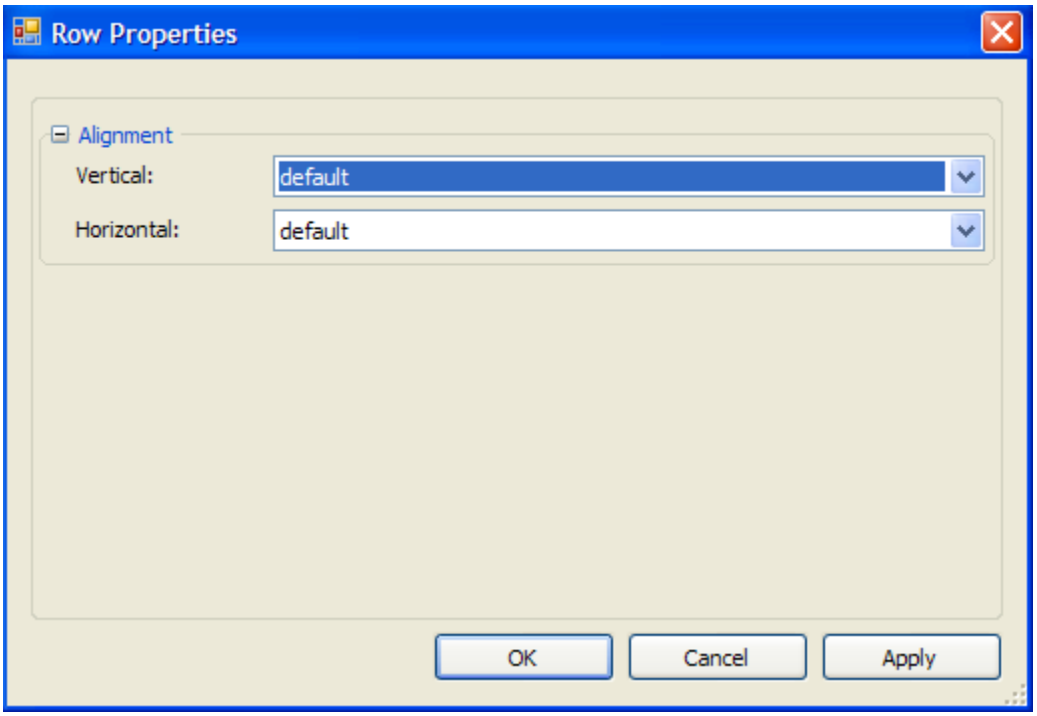
Cell spacing	Enter a number to determine spacing between cells.
Cell padding	Enter a number to determine the spacing that appears around cell content.

Border

Width	Enter the width of the table border.

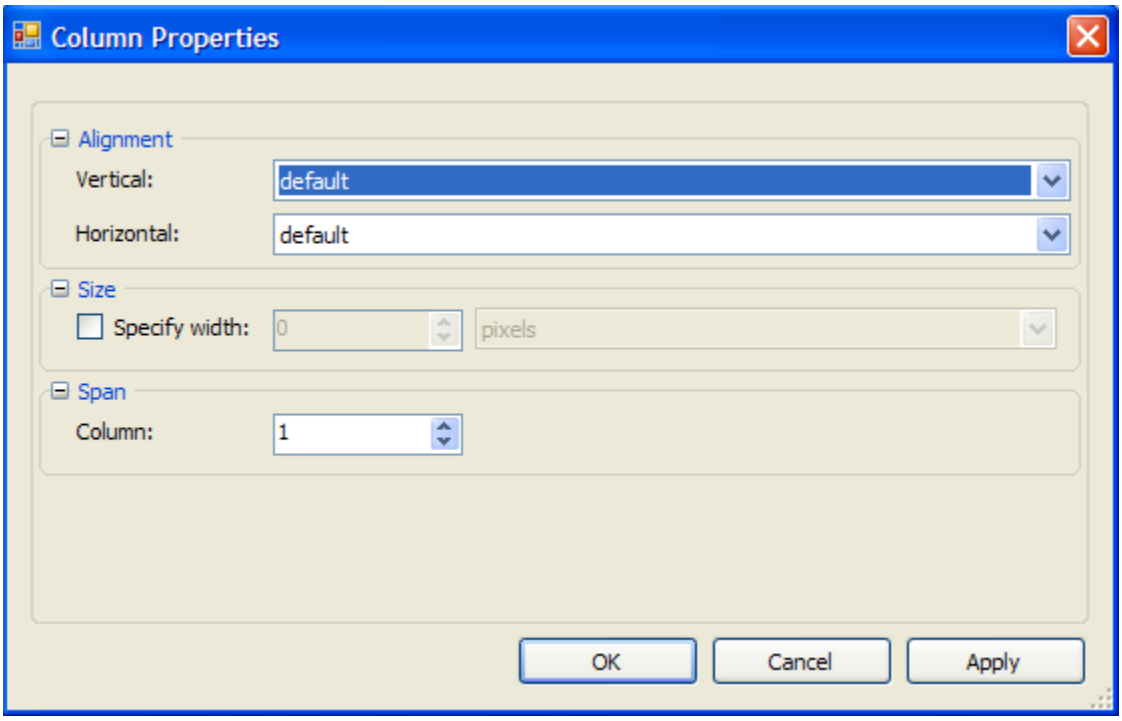
Row Properties

Right-click a table and select **Row Properties** to access this dialog box. Click the drop-down arrows next to **Vertical** or **Horizontal** to specify the alignment.



Column Properties

Right-click a table and select **Column Properties** to access this dialog box.



Alignment

Vertical	Click the drop-down arrows to specify the vertical alignment: top, middle, or bottom.
Horizontal	Click the drop-down arrows next to **Horizontal** to specify the alignment: left, center, or right.

Size

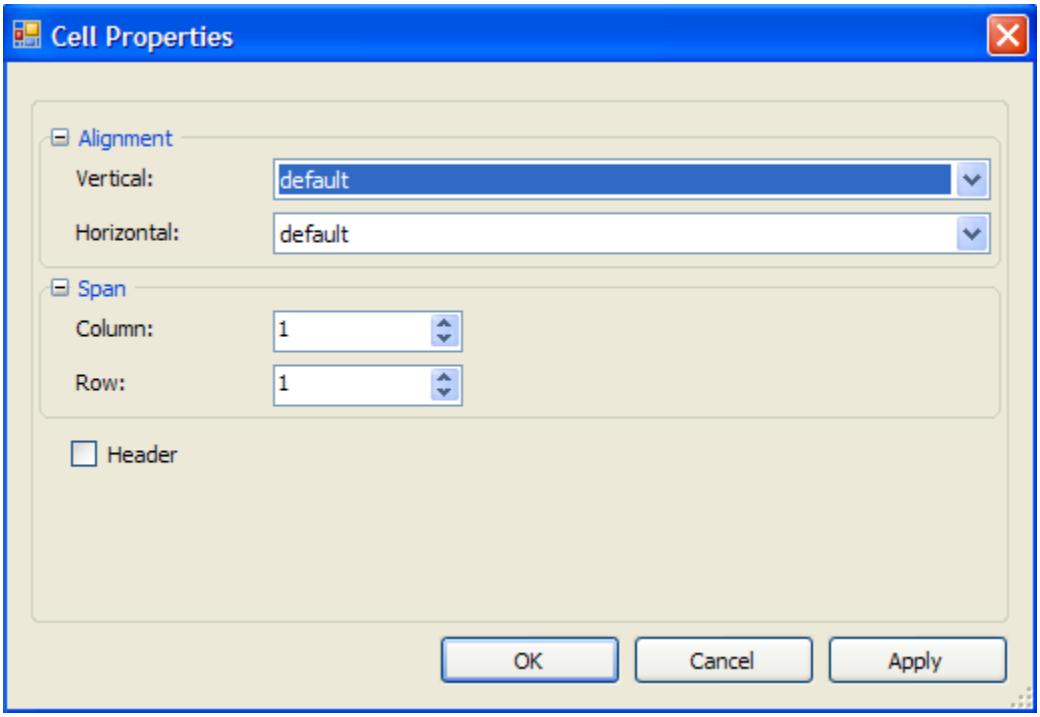
Specify width	Select this checkbox and enter a number in pixels or percent for the width of the column.

Span

Column	Enter a number to determine how many columns the property changes should affect.

Cell Properties

Right-click a table and select **Cell Properties** to access this dialog box.



Alignment

Vertical	Click the drop-down arrows to specify the vertical alignment: top, middle, or bottom.
Horizontal	Click the drop-down arrows next to **Horizontal** to specify the alignment: left, center, or right.

Span

Column	Enter a number to determine how many columns the property changes should affect.
Row	Enter a number to determine how many rows the property changes should affect.

Header

Select this checkbox format the text in a cell as a header, or centered and bold.

Using a Custom Dialog Box

There may be instances where you want to use your own custom version of the built-in C1XmlEditor dialog boxes. This can easily be done with the CustomDialogs property.

First implement the custom dialog box and make sure it supports the appropriate interface, **IFindReplaceDialog** for a custom **Find and Replace** dialog box, for example.

In this example, we'll assume you have created three custom dialog boxes: **BookmarkDialog**, **FindReplaceDialog**, and **FormattingDialog**.

Add the following code to your project to set the CustomDialogs property.

```
private void InitCustomDialogs()
```

```
        {
            editor.CustomDialogs.BookmarkDialog = new
BookmarkEditorForm();
            editor.CustomDialogs.FindReplaceDialog = new
FindReplaceForm();
            editor.CustomDialogs.FormattingDialog = new
FormattingForm();
        }
```

Then you can use the ShowDialog method to open each new dialog box. In this example, we have a toolStrip with three buttons that, when clicked, open the custom dialog boxes.

```
private void toolStrip1_ItemClicked(object sender,
ToolStripItemClickedEventArgs e)
        {
            // opens the Bookmark dialog box
            if (e.ClickedItem == buttonBookmark)
               editor.ShowDialog(C1.Win.XmlEditor.DialogType.Bookmark);
            // opens the Find dialog box
            else if (e.ClickedItem == buttonFind)
               editor.ShowDialog(C1.Win.XmlEditor.DialogType.Find);
            // opens the Formatting dialog box
            else if (e.ClickedItem == buttonFormatting)
               editor.ShowDialog(C1.Win.XmlEditor.DialogType.Format);
        }
```

For a detailed example on creating and using custom dialog boxes, see the **Custom Dialogs** sample installed with the product.

Keyboard Shortcuts

The **XmlEditor for WinForms** controls allow users to complete several functions through the use of keyboard shortcuts when the KeyboardShortcutsEnabled property is set to **True** (default). The following table details the functions that can be accessed through keyboard shortcuts:

Function	Keyboard Shortcut
Bold	Ctrl+B
Italicize	Ctrl+I
Underline	Ctrl+U
Undo	Ctrl+Z
Redo	Ctrl+Y
Select All	Ctrl+A
Copy	Ctrl+C, Ctrl+Shift+Insert
Cut	Ctrl+X, Ctrl+Delete
Paste	CTRL+V, Ctrl+Shift+Insert
Clear Formatting	Ctrl+Space
Find	Ctrl+F
Replace	Ctrl+H

Print	Ctrl+P
Save	Ctrl+S
New	Ctrl+N
Open	Ctrl+O

XmlEditor for WinForms Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos which may make use of other development tools included with the ComponentOne Studios.

Note: The ComponentOne Samples are also available at http://helpcentral.componentone.com/Samples.aspx.

C# Samples

Sample	Description
C1WordsX	This sample shows how you can use the XHtmlEditor control to build a rich text editor. The editor contains a Ribbon control and shows how you can implement formatting, search and replace, tables, images, spell-checking, PDF export, and other advanced text editing features using the C1XmlEditor control.
CustomDialogs	This sample demonstrates how to create custom dialogs for: - inserting/editing hyperlinks, bookmarks, picture, objects and tables; - implementing find/replace commands; - implementing basic text formatting. The C1XmlEditor control contains built-in dialogs for all these tasks, but on some occasions you may want to implement your own in order to customize their appearance or behavior.
CustomTags	This sample shows how you can define and use your own custom tags.
EditModes	This sample shows how the Mode property works. The application contains a form with a C1XmlEditor control. You can set the Mode property of the C1XmlEditor using the buttons at the bottom of the form.
PrintTemplate	This sample demonstrates how you can customize the print and print preview features.
ToolStrips	This sample shows how you can use toolstrips from the C1.Win.XmlEditor.ToolStrips namespace.
SyntaxHighlight	This sample shows how to implement a syntax-highlighting editor using a C1XmlEditor.
UserCSS	This sample shows how you can customize the appearance of the document using custom CSS files.
XmlEditorQuickStart	This sample shows how to implement a basic text editor application using the C1XmlEditor control.

VB.NET Samples

Sample	Description

C1WordsX	This sample shows how you can use the XHtmlEditor control to build a rich text editor. The editor contains a Ribbon control and shows how you can implement formatting, search and replace, tables, images, spell-checking, PDF export, and other advanced text editing features using the C1XmlEditor control.
CustomDialogs	This sample demonstrates how to create custom dialogs for: - inserting/editing hyperlinks, bookmarks, picture, objects and tables; - implementing find/replace commands; - implementing basic text formatting. The C1XmlEditor control contains built-in dialogs for all these tasks, but on some occasions you may want to implement your own in order to customize their appearance or behavior.
CustomTags	This sample shows how you can define and use your own custom tags.
EditModes	This sample shows how the Mode property works. The application contains a form with a C1XmlEditor control. You can set the Mode property of the C1XmlEditor using the buttons at the bottom of the form.
PrintTemplate	This sample demonstrates how you can customize the print and print preview features.
ToolStrips	This sample shows how you can use toolstrips from the C1.Win.XmlEditor.ToolStrips namespace.
SyntaxHighlight	This sample shows how to implement a syntax-highlighting editor using a C1XmlEditor.
UserCSS	This sample shows how you can customize the appearance of the document using custom CSS files.
XmlEditorQuickStart	This sample shows how to implement a basic text editor application using the C1XmlEditor control.

XmlEditor for WinForms Task-Based Help

The task-based help section assumes that you are familiar with programming in the Visual Studio .NET environment and have a general understanding of the C1XmlEditor control.

Each topic provides a solution for specific tasks using the C1XmlEditor control. By following the steps outlined in each topic, you will be able to create projects using a variety of C1XmlEditor features.

Each task-based help topic also assumes that you have created a new .NET project and added a C1XmlEditor control to the form. For additional information on how to do this, see **Creating a .NET Project (page 9)** and **Adding the XmlEditor for WinForms Components to a Project (page 10)**.

Changing the C1XmlEditor Editor Mode

The C1XmlEditor control features three editor modes: **Design**, **Source**, and **Preview**. You can determine which of these views users will see initially by setting the Mode property.

1. In Visual Studio, click the **View** menu and select **Code** to switch to **Source** view, if necessary.
2. Add the following statement to your project.

- Visual Basic
```
Imports C1.Win.XmlEditor
```
- C#
```
using C1.Win.XmlEditor;
```

3. Add the following code to the **Page_Load** event to set the Mode property.
   - Visual Basic
```
C1XmlEditor1.Mode = EditorMode.Source
```
   - C#
```
c1XmlEditor1.Mode = EditorMode.Source;
```

Note: Please note this sample changes the editor mode to Source. You can also set this property to Design or Preview.

4. Press **F5** to build the project and observe C1XmlEditor opens in **Source** view.

Binding C1XmlEditor to a Document

You can bind the C1XmlEditor control to a document specified in the Document property. If the document is edited within the C1XmlEditor, the underlying XmlDocument syncs to match it. If the XmlDocument changes in code, these changes are visible in the C1XmlEditor control at run time.

1. In Visual Studio, click the **View** menu and select **Code** to switch to **Source** view, if necessary.
2. Add the following statements to your project.
   - Visual Basic
```
Imports C1.Win.XmlEditor
Imports System.Xml
```
   - C#
```
using C1.Win.XmlEditor;
using System.Xml;
```
3. Add the following code to the **Page_Load** event to set the Document property.
   - Visual Basic
```
Dim myDoc as new XmlDocument()
C1XmlEditor1.Document = myDoc
```
   - C#
```
XmlDocument myDoc = new XmlDocument();
c1XmlEditor1.Document = myDoc;
```

Loading an XHTML Document from a File

You can load an XHTML document into the C1XmlEditor using the LoadXml method.

1. In the Visual Studio Solution Explorer, right-click the project name, select **New Folder**, and name it **Xhtml**.
2. Place the XHTML document you would like to load into this folder. For this example, we will use a document named "galaxy.htm".
3. Click the **View** menu and select **Code** to switch to **Source** view, if necessary.
4. Add the following statements to your project.
   - Visual Basic

```
Imports C1.Win.XmlEditor
Imports System.IO
```

- C#

```
using C1.Win.XmlEditor;
using System.IO;
```

5. Add the following code to the **Page_Load** event. This code uses the LoadXml method to load your XHTML document into the C1XmlEditor. You will need to update the file path to the location of the XHTML document on your machine.

   - Visual Basic

   ```
   c1XmlEditor1.LoadXml("C:\galaxy.htm")
   ```

   - C#

   ```
   c1XmlEditor1.LoadXml(@"C:\galaxy.htm");
   ```

6. Run the project and the C1XmlEditor will show your XHTML document.

Linking a ToolStrip to C1XmlEditor

You can link one of the four built-in ToolStrips to C1XmlEditor by setting the Editor property.

Setting the Editor property using the Properties Window

To set the Editor property in the Visual Studio Properties window, follow these steps:

1. Right-click your **XmlEditorToolStrip** and select **Properties**.
2. In the Visual Studio Properties window, click the drop-down arrow next to the Editor property and select your C1XmlEditor.

Note: If you expand the Editor property node, you can set other properties for the C1XmlEditor here.

Setting the Editor property Programmatically

Add the following code to your form, in the Form_Load event, for example:

- Visual Basic

```
XmlEditorToolStripMain1.Editor = C1XmlEditor1
```

- C#

```
xmlEditorToolStripMain1.Editor = c1XmlEditor1;
```

Creating a Custom ToolStrip

You can implement your own toolstrip to use with C1XmlEditor by using the C1XmlEditorToolStripBase as a base class and adding buttons.

The following code is an example of a toolbar with three buttons: Cut, Copy, and Paste.

```
using C1.Win.XmlEditor;
using C1.Win.XmlEditor.ToolStrips;


namespace C1XmlEditorCustomToolStrip
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
```

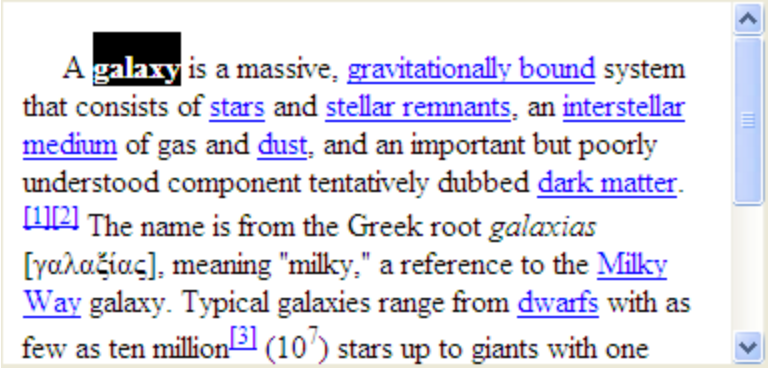
```
        }

         public enum CommandButton
    {
        Cut,
        Copy,
        Paste
    }

        public class MyToolStrip : XmlEditorToolStripBase
        {
            protected override void OnInitialize()
            {

AddButton(C1.Win.XmlEditor.ToolStrips.CommandButton.Cut);

AddButton(C1.Win.XmlEditor.ToolStrips.CommandButton.Copy);

AddButton(C1.Win.XmlEditor.ToolStrips.CommandButton.Paste);
            }
        }

  public class XmlEditorToolStripButton : ToolStripButton
        {
            public C1XmlEditor Editor { get; set; }
            public CommandButton Command { get; set; }
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Using MyToolStrip in an application:

            MyToolStrip toolStrip = new MyToolStrip();
            this.Controls.Add(toolStrip);
            toolStrip.Editor = c1XmlEditor1;
        }
    }
}
```

Selecting Characters in the C1XmlEditor

You can use the SelectionStart property to specify the first character to select in the range. Use the SelectionLength property to specify how many characters should be selected.

For example, the following code starts the selection after the second character in the C1XmlEditor and continues until six characters are selected:

```
private void button1_Click(object sender, EventArgs e)
        {
            C1XmlEditor1.SelectionStart = 2;
            C1XmlEditor1.SelectionLength = 6;
        }
```



Using a Cascading Style Sheet with C1XmlEditor

You can easily format content in the C1XmlEditor using a cascading style sheet (CSS). In this example, we'll use the LoadDesignCSS method to access a CSS. First we're going to add some content to the C1XmlEditor. Then we'll load the CSS.

Add content to the XmlEditor

1. Create a .NET project and add a C1XmlEditor and **Button** control to the form.
2. In the Solution Explorer, double-click **Resources.resx**, click **Add Resource**, and select **Add New String**.
3. Enter **sEditorText** as the **Name** of the resource.
4. In the **Value** column, enter some content for the .xml document. You can copy this sample content and enter it if you prefer.

```
<?xml version="1.0" encoding="utf-8"?><!DOCTYPE html SYSTEM
"C:\Users\jjj\AppData\Local\Temp\tmpB1BB.tmp"[]><html
xmlns="http://www.w3.org/1999/xhtml"><head><title>Famous
Pittsburgers</title><meta http-equiv="Content-Type"
content="text/html; charset=utf-8" /></head><body><h1>Famous
Pittsburghers</h1><p class="BodyText">Many famous and successful
people were born and raised in Pittsburgh; although they found
fame in other cities, they called Pittsburgh home.</p><p
class="C1SectionCollapsed">Actors/Comedians</p><ul><li><p
class="C1HBullet">Gene Kelly (Dancer and star of "Singin' in the
Rain")</p></li><li><p class="C1HBullet">Michael Keaton (Star of
"Batman," "Mr. Mom," and many other movies)</p></li><li><p
class="C1HBullet">Dennis Miller ("Saturday Night Live" cast
member and TV/radio talk show host) </p></li></ul><p
class="C1SectionCollapsed">TV Legend</p><ul><li><p
class="C1HBullet">Fred Rogers (Producer and host of PBS's "Mister
Rogers' Neighborhood")</p></li></ul><p
class="C1SectionCollapsed">Artists</p><ul><li><p
class="C1HBullet">Mary Cassatt (Impressionist
painter)</p></li><li><p class="C1HBullet">Andy Warhol (Pop
artist; Pittsburgh is the home of the Andy Warhol Museum)
</p></li></ul><p class="C1SectionCollapsed">Sports
Legends</p><ul><li><p class="C1HBullet">Dan Marino (Quarterback
of the Miami Dolphins and University of Pittsburgh
Panthers)</p></li><li><p class="C1HBullet">Joe Montana (Super
Bowl champion quarterback of the San Francisco
49ers)</p></li><li><p class="C1HBullet">Joe Namath (Outspoken New
York Jets quarterback; victorious in Super Bowl
III)</p></li><li><p class="C1HBullet">Johnny Unitas (Quarterback
```

```
of the Baltimore Colts for 17 years)</p></li><li><p
class="C1HBullet">Arnold Palmer (Golf legend and winner of more
than 60 PGA events, including the Masters and the U.S.
Open)</p></li></ul><p
class="C1SectionCollapsed">Singers</p><ul><li><p
class="C1HBullet">Perry Como (Recorded over 100 hit
singles)</p></li><li><p class="C1HBullet">Bobby Vinton ("Roses
are Red" and "Blue Velvet" hit #1 on the Billboard
charts)</p></li></ul><p
class="C1SectionCollapsed">Writers</p><ul><li><p
class="C1HBullet">Rachel Carson (Author of "Silent Spring," as
well as a pioneering
environmentalist)</p></li></ul></body></html>
```

5. Go back to your form and double-click it.
6. In the Form_Load event, enter code to load the .xml document so it looks like the following:

```
private void Form1_Load(object sender, EventArgs e)
        {
            c1XmlEditor1.LoadXml(Resources.sEditorText, null);
        }
```

Load a CSS

1. Create a CSS and add it to your project by right-clicking the project name in the Solution Explorer, select **Add Existing Item**, choosing the CSS, and clicking **Add**. You can use this sample CSS if you don't already have one.

```
html {
      font-family: Verdana;
      font-size: 10pt;
      line-height: normal;
      margin-bottom: 0pt;
      margin-left: 0pt;
      margin-right: 0pt;
      margin-top: 0pt;
}

h1 {
      font-family: Verdana;
      font-size: 20pt;
      font-weight: bold;
      line-height: normal;
      margin-bottom: 8pt;
      margin-left: 0pt;
      margin-right: 0pt;
      margin-top: 10pt;
}

h2 {
      font-family: Verdana;
      font-size: 16pt;
      font-weight: bold;
      line-height: normal;
      margin-bottom: 7pt;
      margin-left: 0pt;
      margin-right: 0pt;
      margin-top: 9pt;
      page-break-after: avoid;
}
```

```
h3 {
	font-family: Verdana;
	font-size: 16pt;
	font-weight: bold;
	line-height: normal;
	margin-bottom: 7pt;
	margin-left: 0pt;
	margin-right: 0pt;
	margin-top: 9pt;
	page-break-after: avoid;
}

h4 {
	font-family: Verdana;
	font-size: 12pt;
	font-weight: bold;
	line-height: normal;
	margin-bottom: 2pt;
	margin-left: 0pt;
	margin-right: 0pt;
	margin-top: 2pt;
	page-break-after: avoid;
}

.C1HBullet {
	font-family: Verdana;
	font-size: 10pt;
	  font-style: italic;
	line-height: 14pt;
	margin-bottom: 0pt;
	margin-left: 18pt;
	margin-right: 0pt;
	margin-top: 5pt;
	text-indent: -18pt;
}

p {
	font-family: Verdana;
	font-size: 10pt;
	line-height: 14pt;
	margin-bottom: 0pt;
	margin-left: 0pt;
	margin-right: 0pt;
	margin-top: 5pt;
	  text-indent: 18pt;
}

.C1SectionCollapsed {
	font-weight: bold;
}
```

2. Add a **button1_Click** event and use the LoadDesignCSS method:

```
 private void button1_Click(object sender, EventArgs e)
        {
            c1XmlEditor1.LoadDesignCSS(@"C:\CSS1.css");
        }
```

Run the Project

1. Press F5 to run the project. Notice the document is loaded in the C1XmlEditor.
2. Click the button. The document is formatted with the CSS.

For a detailed example of using custom cascading style sheets, see the **UserCSS** sample that comes with this product.