
ComponentOne

DynamicHelp for WinForms

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

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

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

Sales

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

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of ComponentOne LLC. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the original CD (or diskettes) are free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective CD (or disk) to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for a defective CD (or disk) by sending it and a check for \$25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. We are not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

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

Table of Contents

| | |
|--|-----------|
| ComponentOne DynamicHelp for WinForms | 1 |
| Installing DynamicHelp for WinForms..... | 1 |
| DynamicHelp for WinForms Setup Files..... | 1 |
| System Requirements..... | 2 |
| Installing Demonstration Versions..... | 2 |
| Uninstalling DynamicHelp for WinForms..... | 2 |
| End-User License Agreement..... | 3 |
| Licensing FAQs..... | 3 |
| What is Licensing?..... | 3 |
| How does Licensing Work?..... | 3 |
| Common Scenarios..... | 4 |
| Troubleshooting..... | 6 |
| Technical Support..... | 7 |
| Redistributable Files..... | 8 |
| About this Documentation..... | 8 |
| Namespaces..... | 9 |
| Creating a .NET Project..... | 10 |
| Adding the C1DynamicHelp Component to a Project..... | 10 |
| Key Features | 11 |
| DynamicHelp for WinForms Samples | 11 |
| Using DynamicHelp for WinForms | 13 |
| For Developers: Mapping at Design Time..... | 13 |
| Specifying the Source Help File..... | 14 |
| Mapping a Topic to a Control at Design Time..... | 15 |
| Preparing C1DynamicHelp for Authoring Mode..... | 17 |
| For Help Authors: Mapping in Authoring Mode..... | 18 |
| Mapping a Topic to a Control in Authoring Mode..... | 19 |
| Removing the Topic Mapping for a Selected Control..... | 21 |
| Assign a Topic to a Parent Control..... | 21 |
| Advanced Features..... | 22 |
| Displaying any Help Format..... | 22 |
| Mapping to Control Parts or Custom Controls..... | 22 |
| DynamicHelp for WinForms Tutorials | 25 |
| Tutorial 1 – Mapping Help Topics at Design Time..... | 25 |
| Step 1 of 6: Add controls to the Windows form..... | 25 |
| Step 2 of 6: Set up the C1DynamicHelp control..... | 26 |
| Step 3 of 6: Associate topics with controls on the form..... | 26 |
| Step 4 of 6: Associate a topic with the form..... | 28 |
| Step 5 of 6: Show topics programmatically..... | 29 |
| Step 6 of 6: Run the application..... | 29 |
| Tutorial 2 – Mapping Help Topics in Authoring Mode..... | 30 |
| Step 1 of 4: Add controls to the Windows form..... | 30 |
| Step 2 of 4: Set up the C1DynamicHelp control..... | 31 |
| Step 3 of 4: Using authoring mode..... | 33 |
| Step 4 of 4: Run the Application..... | 34 |

ComponentOne DynamicHelp for WinForms

ComponentOne DynamicHelp™ for WinForms is a container that allows you to view Help files in WinForms applications. Developers or Help authors can also use **ComponentOne DynamicHelp** to assign specific Help topics to controls within an application. The C1DynamicHelp WinForms control displays context-sensitive help corresponding to the control that is being hovered over or the control that has the focus. This type of Help system is an advanced feature available in applications such as Visual Studio, Microsoft Office, Doc-To-Help, and others.

C1DynamicHelp is conceptually similar to using tooltips to provide Help on individual controls and components. The main difference is that tooltips typically contain only a single sentence or paragraph. An actual Help topic, by contrast, typically contains more content, often including hyperlinks, images, tables, and so on. A C1DynamicHelp Help topic also supports selection and can be copied to the Clipboard.

Topic mapping can be done by software developers or Help authors. The following topics provide detailed instructions for both groups.

- [For Developers: Mapping at Design Time](#) (page 13)
- [For Help Authors: Mapping in Authoring Mode](#) (page 18)

Installing DynamicHelp for WinForms

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

DynamicHelp 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 documentation for all Studio components. |
| C1DynamicHelp | Contains files (at least a readme.txt) related to the DynamicHelp for WinForms product. |

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\\My Documents\ComponentOne Samples

Windows 7/Vista path: C:\Users\\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. |
| C1DynamicHelp | Contains samples and tutorials for DynamicHelp for WinForms . |

Samples can be accessed from the **ComponentOne Sample Explorer**. To view samples, on your desktop, click the **Start** button and then click **ComponentOne | Studio for WinForms | Samples | DynamicHelp Samples**.

System Requirements

System requirements include the following:

- | | |
|---------------------------|---|
| Operating Systems: | Windows® 2000 Windows Server® 2003 Windows Server 2008 Windows XP SP2 Windows Vista™ Windows 7 |
| Environment: | .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 **DynamicHelp 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 DynamicHelp for WinForms

To uninstall **DynamicHelp for WinForms**:

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

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("ClCheckForDesignLicenseAtRuntime,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 **DynamicHelp 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.DynamicHelp.dll
- C1.Win.C1DynamicHelp.4.dll
- C1.Win.C1DynamicHelp.4.Design.dll

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

About this Documentation

Acknowledgements

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

ComponentOne

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

Corporate Headquarters

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

<http://www.componentone.com/>

ComponentOne Doc-To-Help

This documentation was produced using [ComponentOne Doc-To-Help® Enterprise](#).

Namespaces

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

The namespace for the **C1DynamicHelp** component is **C1.Win.C1DynamicHelp**. The following code fragment shows how to declare a **C1DynamicHelp** component using the fully qualified name for this class:

- Visual Basic

```
Dim dynamichelp1 As C1.Win.C1DynamicHelp.C1DynamicHelp
```

- C#

```
C1.Win.C1DynamicHelp.C1DynamicHelp dynamichelp1;
```

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 **C1DynamicHelp**, you can use it inside your project without qualification. However, the **C1DynamicHelp** assembly also implements a class called **C1DynamicHelp**. So, if you want to use the **C1DynamicHelp** 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 C1DynamicHelp object  
Dim MyDynamicHelp as C1DynamicHelp  
' Define a new C1DynamicHelp.C1DynamicHelp object.  
Dim C1DynamicHelp as C1.Win.C1DynamicHelp.C1DynamicHelp
```

- C#

```
// Define a new C1DynamicHelp object  
C1DynamicHelp MyDynamicHelp;  
// Define a new C1DynamicHelp.C1DynamicHelp object.  
C1.Win.C1DynamicHelp.C1DynamicHelp C1DynamicHelp;
```

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 C1DynamicHelp = C1.Win.C1DynamicHelp  
Imports MyDynamicHelp = MyProject.C1DynamicHelp  
  
Dim s1 As C1DynamicHelp  
Dim s2 As MyDynamicHelp
```

- C#

```
using C1DynamicHelp = C1.Win.C1DynamicHelp;  
using MyDynamicHelp = MyProject.C1DynamicHelp;  
  
C1DynamicHelp s1;
```

```
MyDynamicHelp 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, 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 C1DynamicHelp component from the Toolbox to add it to Form1. For information on adding a component to the Toolbox, see [Adding the C1DynamicHelp Component to a Project](#) (page 10).

Adding the C1DynamicHelp Component 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 .NET 2.0** 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.

ComponentOne **DynamicHelp for WinForms** provides the **C1DynamicHelp** control. To use **C1DynamicHelp**, add it to the form or add a reference to the C1.Win.C1DynamicHelp assembly to your project.

Adding C1DynamicHelp to the Toolbox

To add **C1DynamicHelp** 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 **C1DynamicHelp** component appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1DynamicHelp**, 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.Win.C1DynamicHelp. Note that there may be more than one component for each namespace.

Adding C1DynamicHelp to the Form

To add **C1DynamicHelp** to a form:

1. Add **C1DynamicHelp** to the Visual Studio Toolbox.
2. Double-click the control or drag it onto your form.

Adding a Reference to the C1DynamicHelp Assembly

To add a reference to the **C1DynamicHelp** assembly:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne C1DynamicHelp** assembly from the list on the **.NET** tab or browse to find the **C1.DynamicHelp.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** statement (**using** in C#):

```
Imports C1.Win.C1DynamicHelp
```

Note: This makes the objects defined in the C1DynamicHelp assembly visible to the project. See [Namespaces](#) (page 9) for more information.

Key Features

The main features of **ComponentOne DynamicHelp for WinForms** include:

- C1DynamicHelp integrates with the application interface
C1DynamicHelp improves the usability of your application by offering immediate and relevant Help as it is needed, right within the interface of the application. A separate Help system doesn't need to be opened and searched.
- Developers or Help Authors can easily perform topic mapping
Software developers can map topics to controls at design time, or Help Authors can use **authoring mode**, a special run-time mode with a simple interface, to do the mapping themselves. No additional code needs to be added once C1DynamicHelp is integrated with the application.
- Supports Help content selection and copying
The Help that is displayed in the C1DynamicHelp Help window can be selected and copied to the Clipboard to be pasted into another application.
- Quickly activate/deactivate **authoring mode**
The software developer can specify a way for Help authors to quickly activate and deactivate **authoring mode** for topic mapping. Any method can be used, such as a keystroke combination, an environment variable, a .config file, and so on.

DynamicHelp 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.

Samples can be accessed from the **ComponentOne Sample Explorer**. To view samples, on the desktop, click the **Start** button and then click **ComponentOne | Studio for WinForms | Samples | DynamicHelp Samples**. The following tables provide a short description for each sample.

Visual Basic Samples

| Sample | Description |
|--------|-------------|
|--------|-------------|

| | |
|------------------------|--|
| AddingUICommands | <p>This sample shows how to add a toolbar to a C1DynamicHelp control to provide more functionality:</p> <ul style="list-style-type: none"> • Allows users to navigate back/forward through help navigation history. • Allows users to open the Help file in an external window on a specified tab. • Allows users to pin the current topic down in order to temporarily disable changing topics. <p>This sample uses the C1DynamicHelp control.</p> |
| MultipleForms | <p>This sample demonstrates how to use C1DynamicHelp controls on multiple forms, how to set them up to use a single help source, and how to refresh the TopicMap to support controls created dynamically. This sample uses the C1DynamicHelp control.</p> |
| UsingUIElementResolver | <p>This sample demonstrates how to create a custom UIElementResolver and use it in the C1DynamicHelp control. This sample uses the C1DynamicHelp control.</p> |
| WorkingWithTopicMap | <p>This sample demonstrates how to load topic map from different locations: from a default topic map file, from a custom file, from application resources. It also shows how to save a modified topic map to different locations. This sample uses the C1DynamicHelp control.</p> |

C# Samples

| Sample | Description |
|------------------------|--|
| AddingUICommands | <p>This sample shows how to add a toolbar to a C1DynamicHelp control to provide more functionality:</p> <ul style="list-style-type: none"> • Allows users to navigate back/forward through help navigation history. • Allows users to open the Help file in an external window on a specified tab. • Allows users to pin the current topic down in order to temporarily disable changing topics. <p>This sample uses the C1DynamicHelp control.</p> |
| MultipleForms | <p>This sample demonstrates how to use C1DynamicHelp controls on multiple forms, how to set them up to use a single help source, and how to refresh the TopicMap to support controls created dynamically. This sample uses the C1DynamicHelp control.</p> |
| UsingUIElementResolver | <p>This sample demonstrates how to create a custom UIElementResolver and use it in the C1DynamicHelp control. This sample uses the C1DynamicHelp control.</p> |
| WorkingWithTopicMap | <p>This sample demonstrates how to load topic map from different locations: from a default topic map file, from a custom file, from application resources. It also shows how to save a modified topic map to different locations. This sample uses the C1DynamicHelp control.</p> |

Using DynamicHelp for WinForms

The process of mapping Help topics to controls within a software application has often been thought of as an error-prone process of passing a topic map back and forth between Help authors and software developers.

ComponentOne DynamicHelp for WinForms eliminates that painful process by providing two options:

- Software developers can use C1DynamicHelp at design time to map the topics to controls.
or
- Help authors can map topics to controls themselves using the [C1DynamicHelp authoring mode](#) (page 19), without having to go through a software developer.

C1DynamicHelp gives you the ability to map a topic to any standard .NET controls and even parts of those controls. You can also map to custom controls. See the [Advanced Features](#) (page 22) topic of this documentation for more information on mapping to control parts or custom controls.

ComponentOne DynamicHelp for WinForms supports several Help formats including HTML Help and NetHelp. No matter who is going to perform the topic mapping, the software developer must [specify the Help file](#) (page 14) that is going to be used in the application before mapping can occur.

The following topics explain both ways to perform topic mapping.

Note: If you are the software developer of the application and would like the Help author to map topics to controls, please provide them with this documentation. The [For Help Authors: Mapping in Authoring Mode](#) (page 18) topic provides all the information needed for topic mapping at run time in authoring mode.

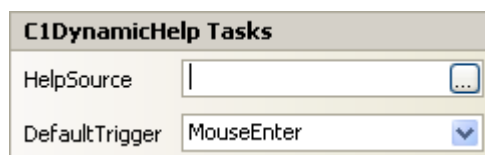
For Developers: Mapping at Design Time

C1DynamicHelp provides visual editing at design time to make it easier to map specific Help topics to controls within an application. You can use the **C1DynamicHelp smart tag** and **Select Help Topic** dialog box to quickly perform topic mapping.

C1DynamicHelp Smart Tag

In Visual Studio, the C1DynamicHelp component includes a smart tag. A smart tag (📌) represents a short-cut tasks menu that provides the most commonly used properties for a control.

To access the **C1DynamicHelp Tasks** menu, click the smart tag in the upper-right corner of the C1DynamicHelp control.



- **HelpSource Property**

Click the **ellipsis** button to locate and select the .chm or NetHelp .htm file that will be used for topic mapping.

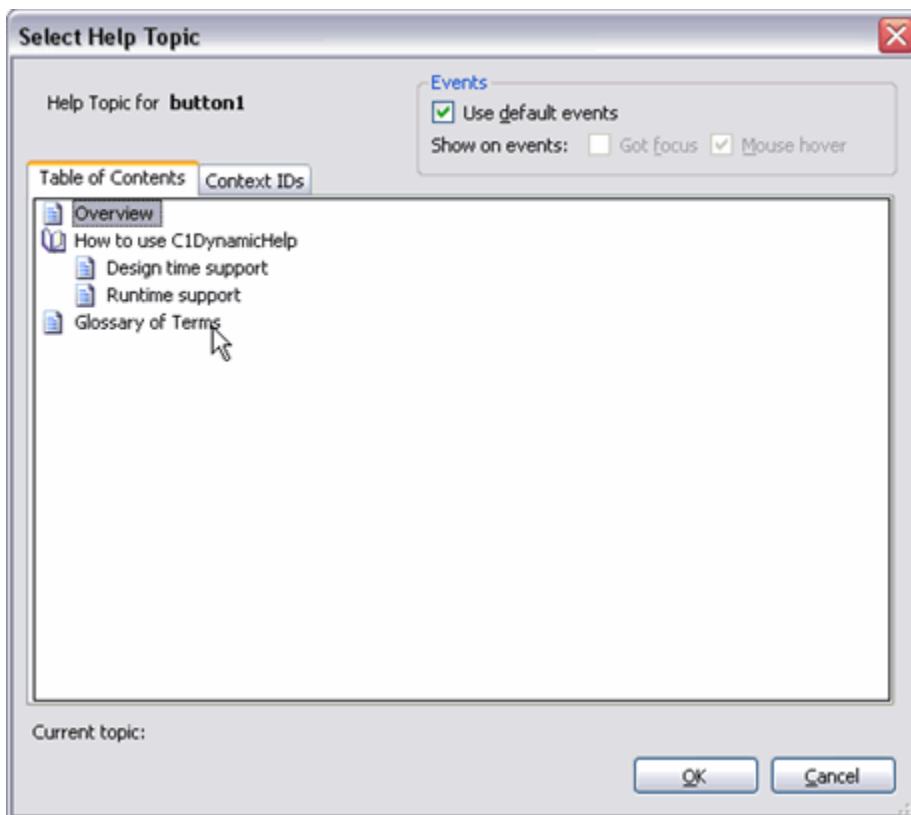
Note: If using NetHelp, use the default.htm or the file name that was used as the base URL for the NetHelp target.

- **DefaultTrigger Property**

The DefaultTrigger property allows you to specify whether the Help topic will appear when the control has the focus (**Enter**), the mouse is hovering over the control (**MouseEnter**), or not at all (**None**).

Select Help Topic Dialog Box

Within this dialog box, you can specify the topic to map to the control through the **Table of Contents** tab by simply selecting it, or you can choose the topic's context ID from the **Context IDs** tab. Additionally, you can specify here whether a topic should be shown when: the control gets focus; the mouse hovers over it; both; or neither, in which case, you can show the topic programmatically.



Specifying the Source Help File

No matter who is going to perform the topic mapping, the software developer must specify the Help file that is going to be used in the application before mapping can occur. Either NetHelp or a .chm (compiled HTML Help) can be used by C1DynamicHelp. If using NetHelp, all NetHelp project files must be deployed with your application. If using HTML Help, only the .chm must be included.

Any other types of Help files can be used, but this requires some additional programming; you can use the Provider property and IHelpProvider interface.

Note: In Doc-To-Help, NetHelp project files are stored by default in the NetHelp folder; the compiled HTML Help file is stored in the HTMLHelp folder.

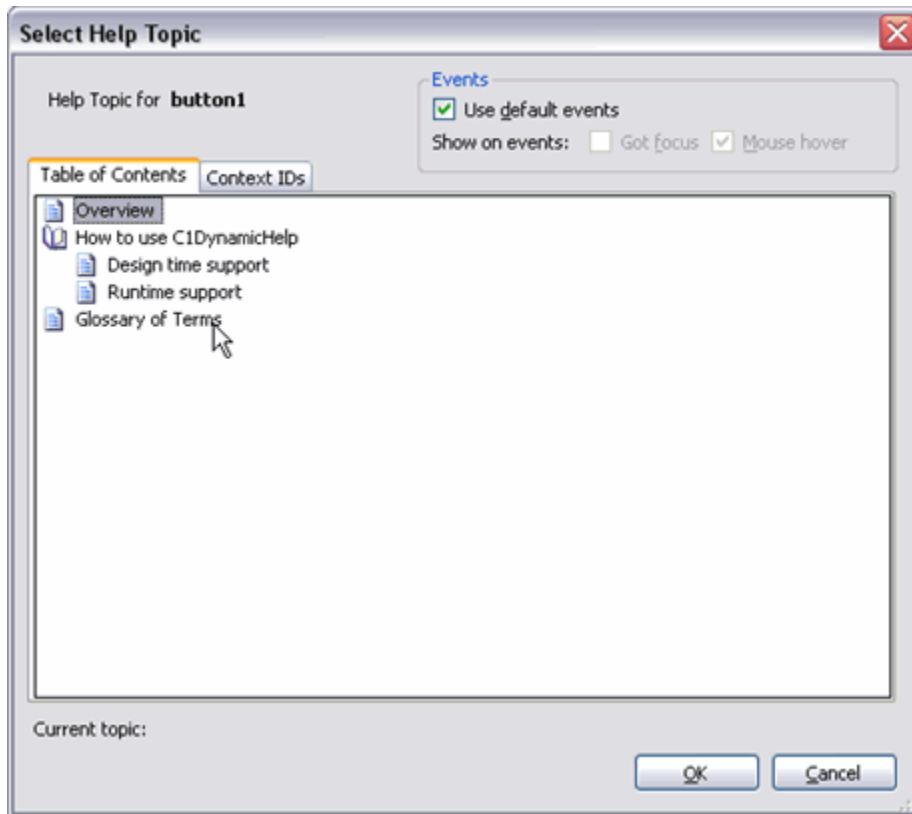
To specify the source Help file to be used:

1. From the Toolbox, double-click the C1DynamicHelp control to add it to your form. If you have not already added the C1DynamicHelp control to the Toolbox, see [Adding the C1DynamicHelp Component to a Project](#) (page 10).
2. Click the C1DynamicHelp smart tag (▾) to open the **Tasks** menu.
3. Click the **ellipsis** button next to the HelpSource property.
4. Locate and select the desired Help file. You can use any *.chm file or NetHelp *.htm file (use the default.htm or the file name that was used as the base URL for the NetHelp target).
5. Click **Open**.

Mapping a Topic to a Control at Design Time

Once a source Help file is specified, you can easily map topics to controls within your application through C1DynamicHelp.

1. From the Toolbox, double-click the C1DynamicHelp control to add it to your form. If you have not already added the C1DynamicHelp control to the Toolbox, see [Adding the C1DynamicHelp Component to a Project](#) (page 10). An extender property, **Help Topic on C1DynamicHelp**, is added to all controls when C1DynamicHelp is added to the form.
2. Click the C1DynamicHelp smart tag (▾) to open the **Tasks** menu.
3. Click the **ellipsis** button next to the HelpSource property.
4. Locate and select the desired Help file, and click **Open**.
5. Select one of the controls on your form and click the **ellipsis** button next to the **HelpTopic on C1DynamicHelp** property in the Properties window. The **Select Help Topic** dialog box appears and shows the **Table of Contents** tab and TOC from the specified Help file.

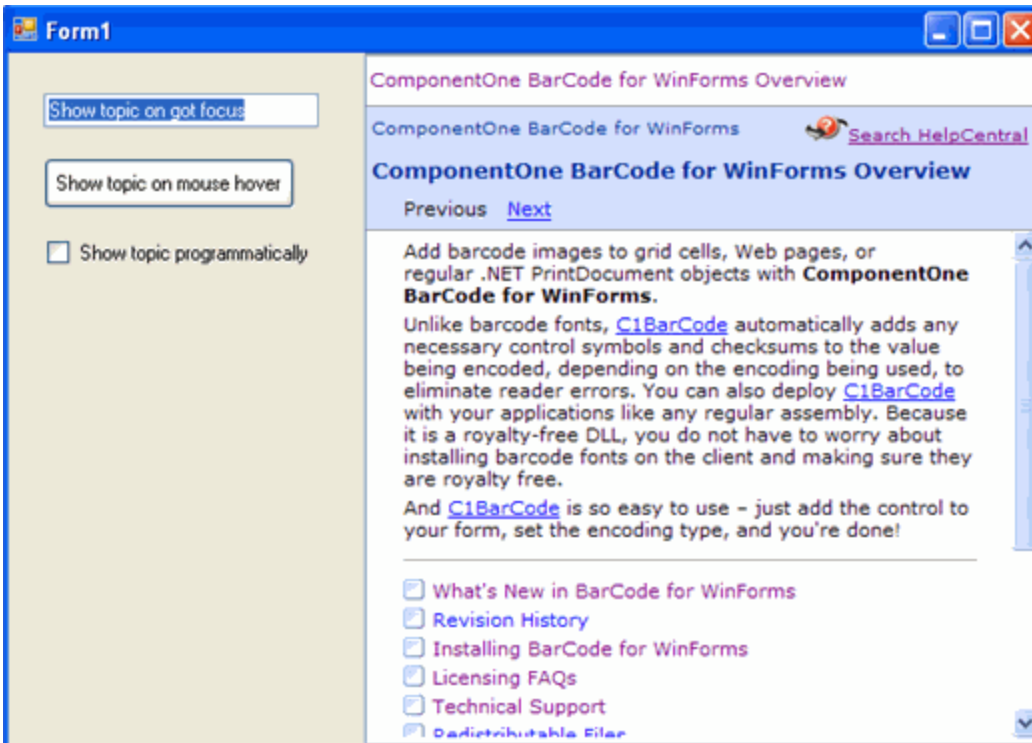


By default, the **Use default events** check box is selected (the selected options will vary by the control you've chosen to map to). **Got focus** sets the Help topic to display when the control has been selected; **Mouse hover** sets the Help topic to display when the control has been hovered over. To change the default options, clear the **Use default events** check box and select either the **Got focus** or **Mouse hover** check box.

- **If mapping to the TOC:** From the **Select Help Topic** dialog box, **Table of Contents** tab, choose a topic.
- **If mapping to the context ID list:** From the **Select Help Topic** dialog box, **Context ID** tab, choose the context ID/Topic pairing.

Note: In order to map to a topic, the topic must be in the Help TOC or have a context ID assigned to it; otherwise, it will not appear in the **Select Help Topic** dialog box. Also, if the Help author renames one of the TOC items, the mapping will break. You must get the updated Help and re-create the mapping with the updated TOC.

6. Click **OK** to close the **Select Help Topic** dialog box. The topic is mapped to the control and appears in the C1DynamicHelp Help window at run time when a user clicks or mouses over the control.



Preparing C1DynamicHelp for Authoring Mode

To allow Help authors to activate and deactivate C1DynamicHelp **authoring mode** so that they can map topics in your application, you have a wide variety of options. You can use anything from a special keystroke combination to a registry node that, if present, activates authoring mode.

This example shows how to specify a keystroke combination.

1. [Specify a source Help file for C1DynamicHelp](#). (page 14)
2. Select your **Form** and set the **KeyPreview** property to **True** in the Properties window.
3. Select **View | Code** so you can add code to override the **OnKeyDown** method and specify a keystroke combination to activate and deactivate **authoring mode**.
4. Add the following code:

- Visual Basic

```
' toggle authoring mode when the user hits Ctrl+Shift+A
Protected Overrides Sub OnKeyDown(ByVal e As
System.Windows.Forms.KeyEventArgs)
    If (e.KeyCode = Keys.A And e.Control And e.Shift) Then
        C1DynamicHelp1.AuthoringMode = Not
C1DynamicHelp1.AuthoringMode
    End If
    MyBase.OnKeyDown(e)
End Sub
```

- C#

```
// toggle authoring mode when the user hits Ctrl+Shift+A
override protected void OnKeyDown(KeyEventArgs e)
{
    if (e.KeyCode == Keys.A && e.Control && e.Shift)
    {
```

```
        c1DynamicHelp1.AuthoringMode =
!c1DynamicHelp1.AuthoringMode;
    }
    base.OnKeyDown(e);
}
```

C1DynamicHelp **authoring mode** will be activated or deactivated when the user presses the **Ctrl+Shift+A** keys.

For Help Authors: Mapping in Authoring Mode

Topic mapping is generally performed by the application Help author or Information Developer; for that reason, the roles of the Information Developer and Software Developer in the process are clearly defined here, although one person may perform both roles.

The files used for mapping include:

- NetHelp project files or .chm (compiled HTML Help)

The Help file format is [specified by the software developer](#) (page 14). If using NetHelp, all of the files in the output folder of your Help project must be deployed with the application. If using HTML Help, only the .chm must be included.

Note: In Doc-To-Help, NetHelp project files are stored by default in the NetHelp folder; the compiled HTML Help file is stored in the HTMLHelp folder.

- Default.htm.xml or HelpFileName.chm.xml

When you create mappings, they are saved to the **default.htm.xml** or **HelpFileName.chm.xml** located in the same directory as your source Help file, by default. This XML is used to show the topics when the application runs.

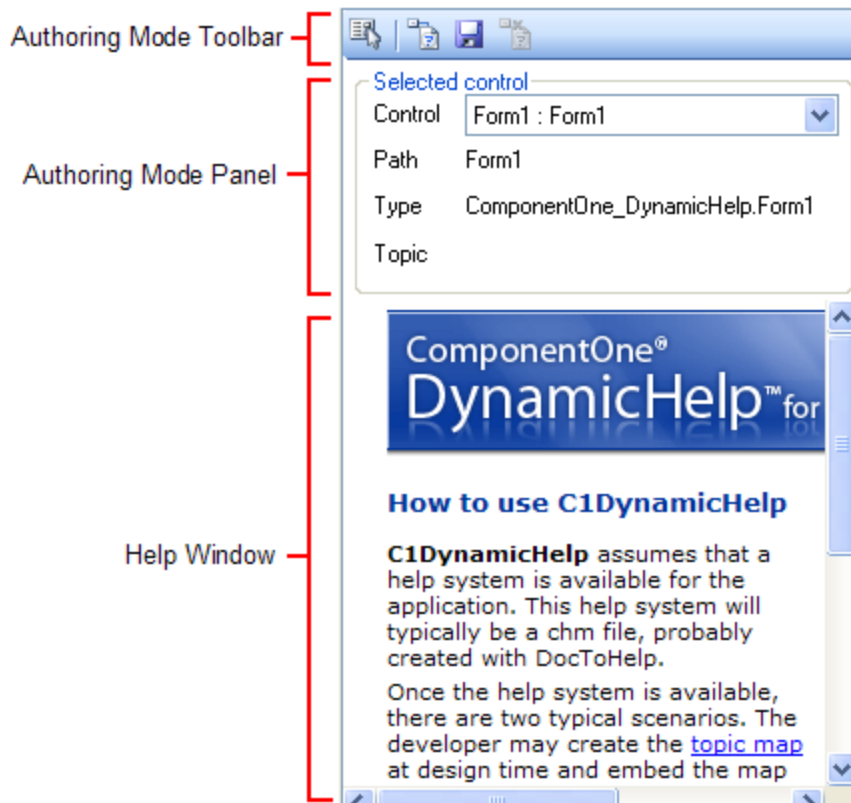
You cannot begin mapping until you place your Help into the correct folder. NetHelp or .chm files should be copied to the install folder specified by the software developer for your application (for example: \\program files\componentone\DocToHelp\Help).

When you have finished mapping topics, the .chm or NetHelp project files and the **.xml** containing the topic mappings must be given to the software developer.

Note: The **.xml** mapping file should never be edited manually. If mappings need to be deleted, use the **C1DynamicHelp authoring mode** (see below).

C1DynamicHelp Authoring Mode

DynamicHelp for WinForms provides a special run-time mode, **authoring mode**, for Help authors to use for assigning topics to controls within an application. When activated, the authoring mode panel and toolbar appear.




Information about the control being mapped to is provided in the **Selected control** area of the panel, which is made up of the following fields:


| | |
|----------------|---|
| Control | Control name : type of control. |
| Path | The path of the control relative to its placement on the form. For example, a Node within a TreeView control may have a path that looks like this: Form1\TreeView1\Node0. |
| Type | Type of control. |
| Topic | Topic to associate with the control. |

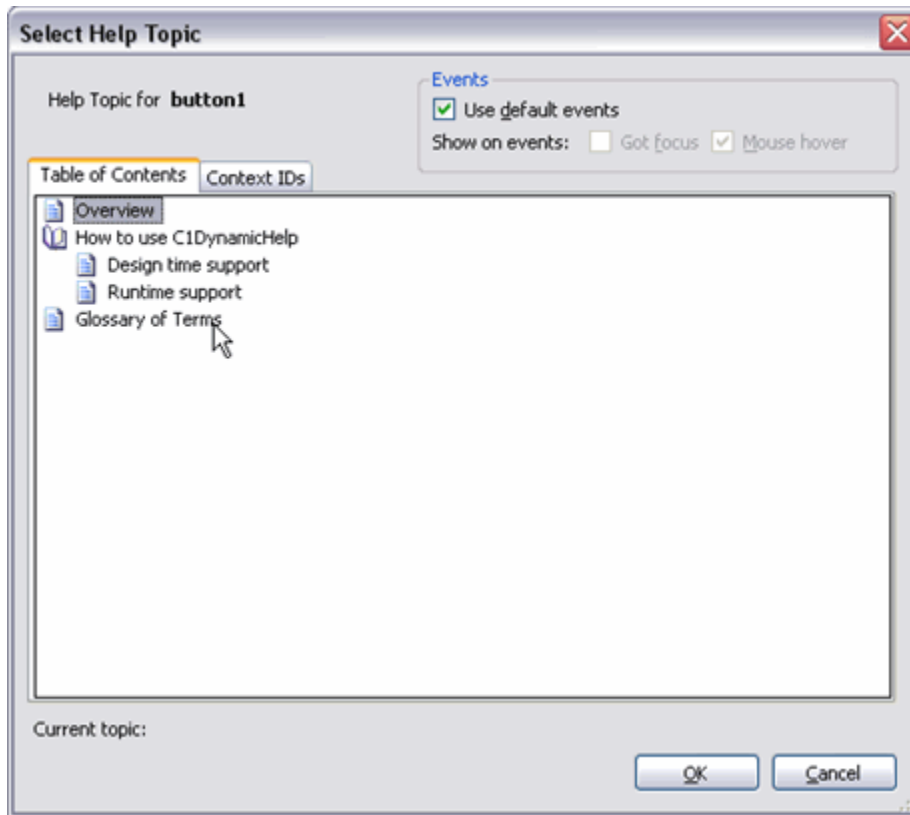
The software developer controls how you to activate and deactivate **authoring mode**.

Mapping a Topic to a Control in Authoring Mode

To map a control to a topic using the **C1DynamicHelp authoring mode**, use the following steps:

1. Display the area of the application's interface that you would like to map. For example, if you are going to map ribbons, make sure they are all available. If you are going to map windows, make sure they are all open.
2. Activate **authoring mode** using the method specified by the software developer. The authoring mode panel appears.
3. Click the **Select control** button .


4. Click the item you would like to map a Help topic to. As you move the mouse over the controls, you will see information about them in the authoring mode panel: **Control** name, **Path**, control **Type**, and associated **Topic**, if applicable.
5. Click the **Attach topic to control** button . The **Select Help Topic** dialog box appears and shows the **Table of Contents** tab and TOC from the source Help file.



By default, the **Use default events** check box is selected (the selected options will vary by the control you've chosen to map to). **Got focus** sets the Help topic to display when the control has been selected; **Mouse hover** sets the Help topic to display when the control has been hovered over. To change the default options, clear the **Use default events** check box and select either the **Got focus** or **Mouse hover** check box.

- **If mapping to the TOC:** From the **Select Help Topic** dialog box, **Table of Contents** tab, choose a topic. Click **OK**. The topic chosen will display in the **Topic** field of the authoring mode panel.
- **If mapping to the context ID list:** From the **Select Help Topic** dialog box, **Context ID** tab, choose the context ID/Topic pairing. Click **OK**. The topic chosen will display in the **Topic** field of the authoring mode panel.

Note: In order to map to a topic, the topic must be in the Help TOC or have a context ID assigned to it; otherwise, it will not appear in the **Select Help Topic** dialog box. Also, if you rename one of the TOC items, the mapping will break. Simply drop the updated Help into the proper folder and re-create the mapping with the updated TOC.




6. Click the **Save** button .
7. Continue mapping.

8. To close the window, use the method specified by the software developer.
9. Deliver the **.xml** mapping file to the software developer.

Note: After mapping, you should backup the **.xml** mapping file elsewhere on your machine. If you uninstall or reinstall your software product, the **.xml** mapping file will be deleted and replaced.


Removing the Topic Mapping for a Selected Control

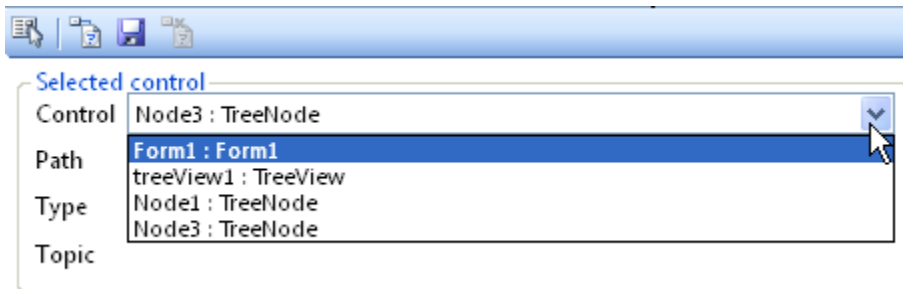
To remove the topic mapping for a selected control, use the following steps:



1. Activate **authoring mode** using the method specified by the software developer. The authoring mode panel appears.
2. Click the **Select Control** button  and select the control that you would like to remove the topic mapping from. The control name appears in the **Control** field of the authoring mode panel, so you can make sure the correct control has been selected.
3. Click the **Detach Topic from Control** button . The topic mapping is removed. Notice there is no longer a topic specified in the **Topic** field.
4. Click the **Save** button .
5. Close the mapping window and deliver the proper Help and **.xml** mapping file to the software developer.

Assign a Topic to a Parent Control

There may be times you need to attach a topic not to a control you can select, but to its parent or ancestor. Since the parent can be completely covered by child controls and thus unselectable, there is a special provision to do it in authoring mode.

1. Activate **authoring mode** using the method specified by the software developer. The authoring mode panel appears.
2. Click the **Select control** button .
3. Click the child control of the parent or ancestor control you would like to map a Help topic to.
4. Click the drop-down arrow in the **Control** combo box and select the parent or ancestor to map to.



5. Click the **Attach topic to control** button . The **Select Help Topic** dialog box appears and shows the **Table of Contents** tab and TOC from the source Help file.
6. Select the topic from the **Table of Contents** tab or select the context ID/Topic pairing on the **Context IDs** tab, and specify any desired events.
7. Click the **Save** button .
8. Close the mapping window and deliver the proper Help and **.xml** mapping file to the software developer.

Advanced Features

ComponentOne DynamicHelp for WinForms requires almost no programming to be used in most cases, but sometimes you may want to use the best of it. The following advanced features are described in this topic:

- **Displaying any Help format**

C1DynamicHelp allows you to use different help formats. This topic describes two built-in help providers, ChmProvider and NetHelpProvider, and how to implement your own help provider to display a help file in the format you want.

- **Mapping to part of a control or custom controls**

C1DynamicHelp gives you the ability to map a topic not only to standard controls but also to parts of controls or to custom controls. C1DynamicHelp provides this feature for standard .Net controls derived from the System.Windows.Forms.Control class. But if you want to extend it to your needs, you can create your own UI element resolver class.

Displaying any Help Format

You may want to use different help formats in your application:

- HTML help;
- NetHelp;
- MS Help 2.0;
- WinHelp;
- RoboHelp WebHelp;
- and so on.

C1DynamicHelp provides means for displaying these or any other help formats.

HTML Help and NetHelp formats can be displayed by the C1DynamicHelp control without any additional programming. There are two built-in help providers in C1DynamicHelp, ChmProvider and NetHelpProvider, which are used automatically when the HelpSource property points to a .chm or an .htm/html file, correspondingly.

Any other types of help files can be used, but this requires some additional programming. In this case you will have to create your own help provider class that implements the IHelpProvider interface. This interface provides all necessary information about a help format to the C1DynamicHelp control. So any help provider can read data from a help source and provides methods to get help topics, to get context IDs, to open help in an external window, to get a topic URL that can be displayed by the C1DynamicHelp control, and so on. After creating your own help provider, all you need is to set it to the Provider property.

For more details see:

- the IHelpProvider interface, which must be implemented by every help provider;
- the Provider property;
- the NetHelpProvider sample, which is the full code actually used in the C1DynamicHelp control.

Mapping to Control Parts or Custom Controls

C1DynamicHelp allows you to map a help topic to any control derived from the System.Windows.Forms.Control class. In most cases it will be enough, but sometimes you may want to map a topic not to a control itself but to some part of it or map a topic to a custom control which is not derived from the System.Windows.Forms.Control. We will refer to these parts of controls and the custom controls as UI elements. To inform the C1DynamicHelp control how to handle UI elements, you will need to create your own class derived from the UIElementResolver class and set it as the Resolver property.

You will need to create a class derived from the `UIElementResolver` only if you are using custom controls that cannot be handled by the `C1DynamicHelp` control automatically and then only if you need to associate help topics with parts (UI elements) of those controls, not with the controls themselves.

You have no need to create objects of `UIElementResolver` type, it is sufficient to define a class derived from `UIElementResolver` and override its virtual methods. These overridden methods must provide necessary information about UI elements inside custom controls used in your application: methods to find UI elements inside a control by name, coordinates, etc., and other methods necessary for associating dynamic help to UI elements inside a control.

`C1DynamicHelp` automatically handles most popular, standard controls. For example, it allows mapping a topic to all standard .Net controls and to their parts. So, for example, you can associate a help topic to a separate node of the standard **TreeView** control or to a separate **ListViewItem** of the **ListView** control; that you can do without creating your own `UIElementResolver` class.

For more details see:

- the `UIElementResolver` class, from which you derive your own UI element resolver class;
- the `Resolver` property;
- the **UsingUIElementResolver** sample, which shows how to create custom UI element resolver for the **C1Ribbon** control to allow mapping topics to items inside **C1Ribbon** such as tabs, buttons, groups, etc.

DynamicHelp for WinForms Tutorials

The following tutorials provide step-by-step instructions on mapping Help topics to controls within an application. Tutorial 1 explains how to map Help topics at design time within Visual Studio. This is usually done by a software developer. Tutorial 2 explains how software developers can set up C1DynamicHelp **authoring mode** and how Help authors can then use it to assign topics to controls on a form, without having to manually edit a topic map.

Tutorial 1 – Mapping Help Topics at Design Time

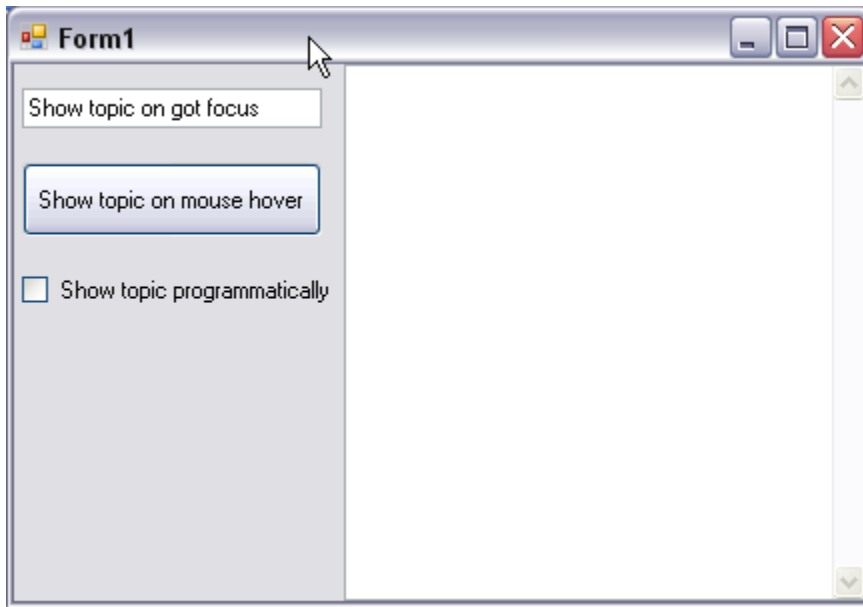
This tutorial explains how to use C1DynamicHelp to map Help topics to controls on a form at design time within Visual Studio. This is usually done by a software developer.

Step 1 of 6: Add controls to the Windows form

In this step, you will add the C1DynamicHelp control to your form, along with a **TextBox**, **Button**, and **CheckBox**.

1. [Create a .NET project](#) (page 10) and [add the C1DynamicHelp control to the Toolbox](#) (page 10). An extender property, **Help Topic on C1DynamicHelp**, is added to all controls when C1DynamicHelp is added to the form.
2. From the Toolbox, double-click the C1DynamicHelp control. It docks at the right of your form.
3. Add a **TextBox** control to the form:
 - a. From the Toolbox, double-click the **TextBox** control to add it to your form.
 - b. From the Properties window, set the **textBox1.Text** property to **Show topic on got focus**.
4. Add a **Button** control to the form:
 - a. From the Toolbox, double-click the **Button** control to add it to your form.
 - b. From the Properties window, set the **button1.Text** property to **Show topic on mouse hover**.
5. Add a **CheckBox** control to the form:
 - a. From the Toolbox, double-click the **CheckBox** control to add it to your form.
 - b. From the Properties window, set the **checkBox1.Text** property to **Show topic programmatically**.

You have successfully added the controls to your form, which should look similar to the following:



In the next step you will set up the C1DynamicHelp control.

Step 2 of 6: Set up the C1DynamicHelp control

In order to associate topics from your Help file with form controls, you must specify the source Help file.

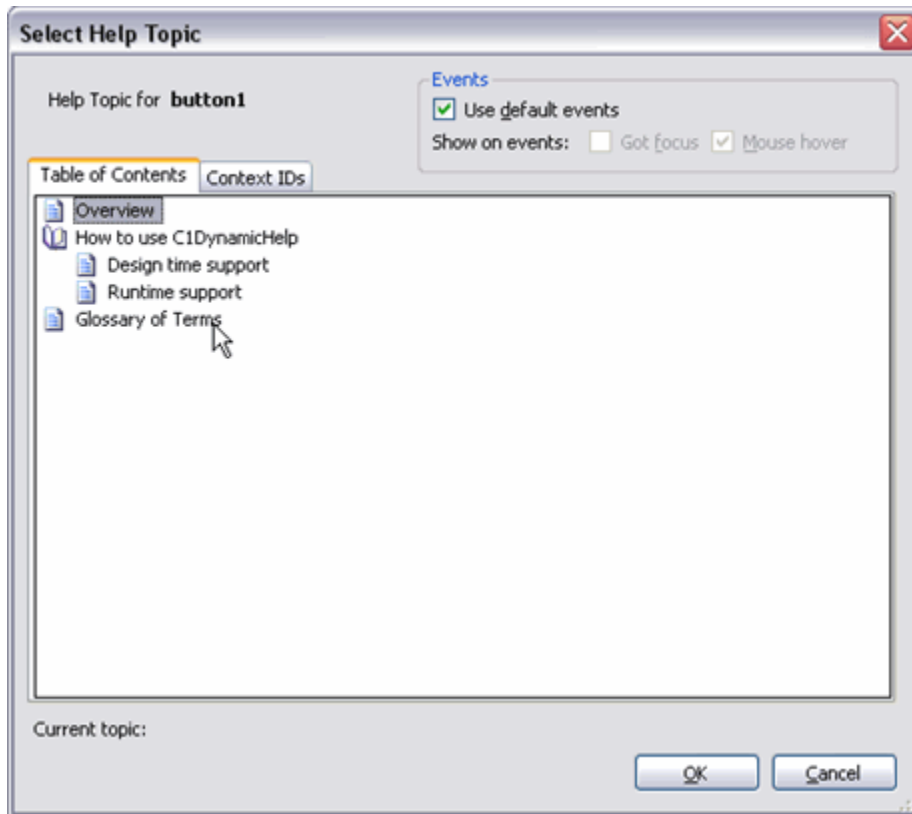
You can do this from the **C1DynamicHelp Tasks** menu by setting the HelpSource property to any *.chm file or NetHelp *.htm file (use the default.htm or the file name that was used as the base URL for the NetHelp target.). In this example, we will use the **C1Sample.chm** installed with **DynamicHelp for WinForms**, by default.

1. Click the C1DynamicHelp smart tag (📌) to open the **Tasks** menu.
2. Click the **ellipsis** button next to the HelpSource property.
3. Locate and select the **C1Sample.chm**. It is located in the Tutorials/Data folder, by default.
4. Click **Open**.

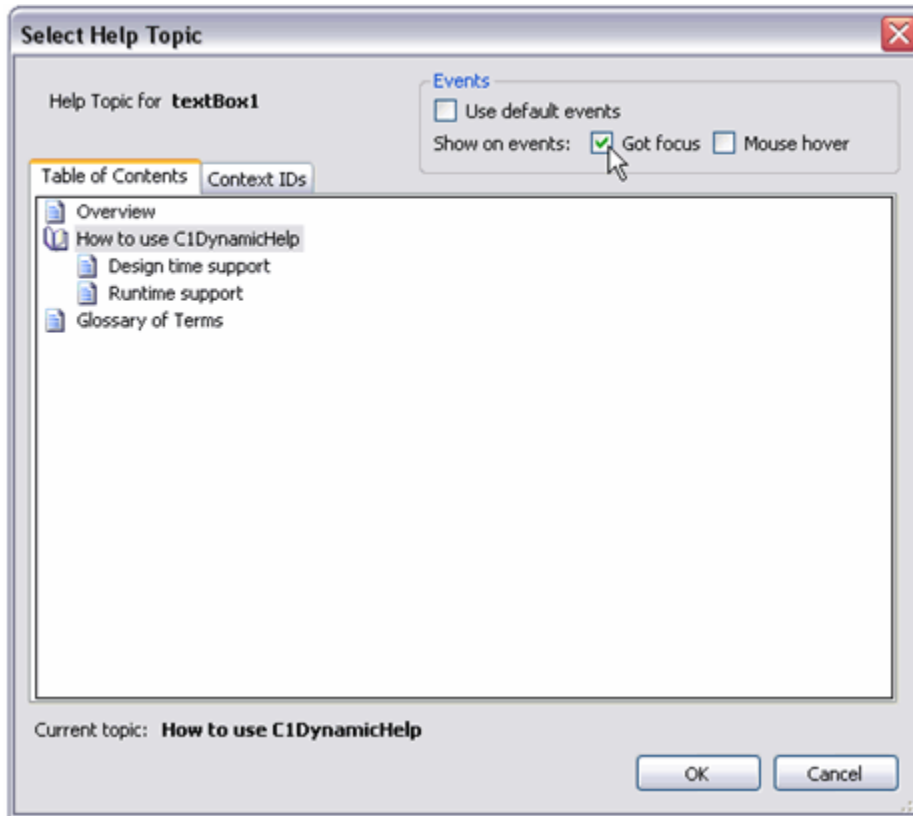
Step 3 of 6: Associate topics with controls on the form

You can associate Help topics with controls using the **Select Help Topic** dialog box.

1. Select **Button1** and click the ellipsis button next to the **HelpTopic on C1DynamicHelp1** property in the Properties window. The **Select Help Topic** dialog box opens:



2. Select any help topic from the **Table of Contents** tab and click **OK**. In this example, we will select the **Overview** topic.
3. Select **TextBox1** and set its **HelpTopic on C1DynamicHelp1** property to any topic from the **Table of Contents** tab.
4. In the **Events** group of the **Select Help Topic** dialog box, uncheck **Use default events** and check the **Got focus** checkbox next to **Show on events**.



Unchecking **Use default events** allows you to specify the events on a per-control basis. Otherwise, the `DefaultTrigger` property is used for all controls.

Using the **Events** settings makes showing topics automatic when a control obtains focus or the mouse hovers over it, or in both cases. But if you want manual control over what triggers topic display, you can uncheck both check boxes (or set `DefaultTrigger=None`; this will do it for all controls if it's not overridden on a per-control basis) and [show topics programmatically](#) (page 29).

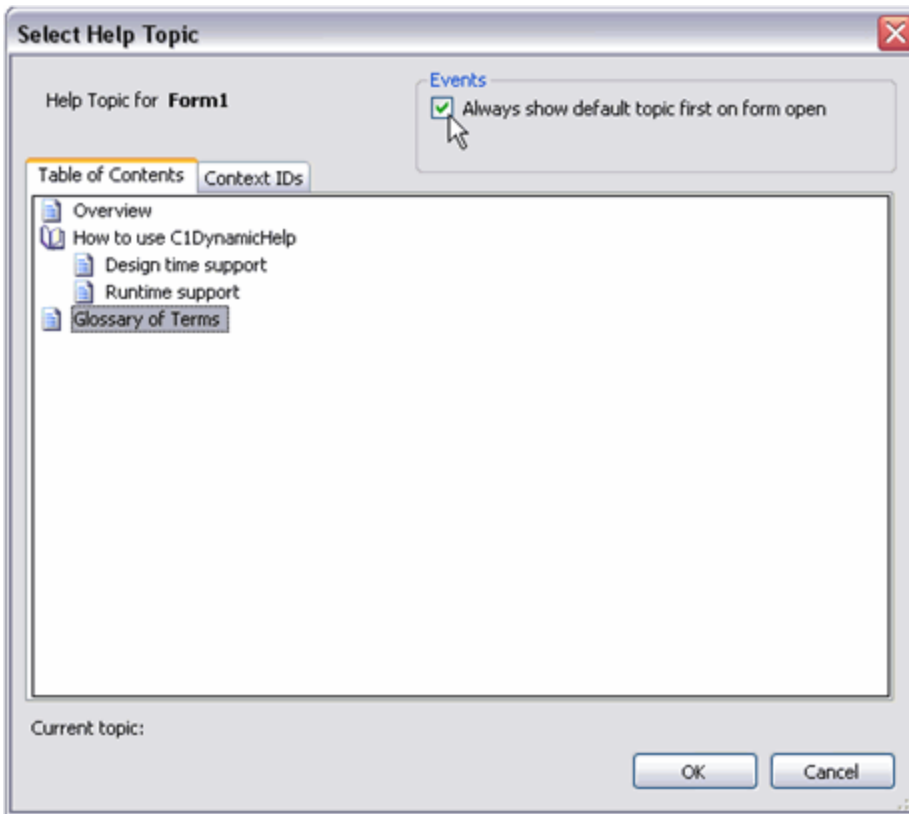
5. Click **OK**.

Step 4 of 6: Associate a topic with the form

You can also use the **Select Help Topic** dialog box to associate a topic with your form.

1. Select **Form1** and click the **ellipsis** button next to the **HelpTopic on C1DynamicHelp1** property in the Properties window. The **Select Help Topic** dialog box appears, but notice the **Events** group is different; in this case, it has one check box **Always show default topic first on form open**.

The topic associated with the form is the default topic. It is shown when the control that displays the current topic due to a focus/mouse hover event loses focus/mouse hover. If the check box **Always show default topic first on form open** is checked, the default topic is also shown when the form is opened.



2. Select the default topic from the **Table of Contents** tab.

Step 5 of 6: Show topics programmatically

You can show a specific topic programmatically by using the ShowTopic method.

Create a **CheckedChanged** event handler for **CheckBox1** and add code so it looks like the following:

- Visual Basic

```
Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
    C1DynamicHelp1.ShowTopic("WordDocuments/glossaryofterms.htm")
End Sub
```

- C#

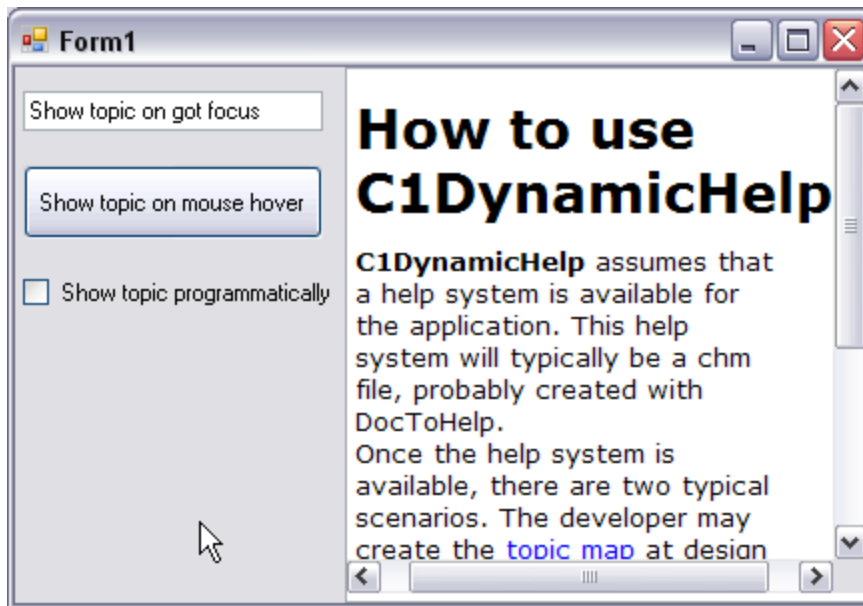
```
private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
{
    c1DynamicHelp1.ShowTopic("WordDocuments/glossaryofterms.htm");
}
```

Step 6 of 6: Run the application

To run the application, click the **Start Debugging** button and notice the following results:

- When Form1 is loaded, the default topic is shown.
- When the mouse is over the button, the topic is displayed.

- Check or uncheck **Show topic programmatically** to display its associated topic.
- When **TextBox1** gets the focus, it displays the specified topic.



Congratulations, you have successfully completed Tutorial 1! In this tutorial you have learned how to:

- Add and set up the C1DynamicHelp control.
- Associate topics with controls on a form at design-time.
- Show topics programmatically.

Tutorial 2 – Mapping Help Topics in Authoring Mode

This tutorial shows the second method of assigning Help topics to controls on a form. It is done entirely by Help authors; once a software developer sets up his application to use the C1DynamicHelp **authoring mode**, he doesn't need to do anything to map topics to controls! This eliminates the error-prone process of passing the topic/control map back and forth between Help authors and software developers.

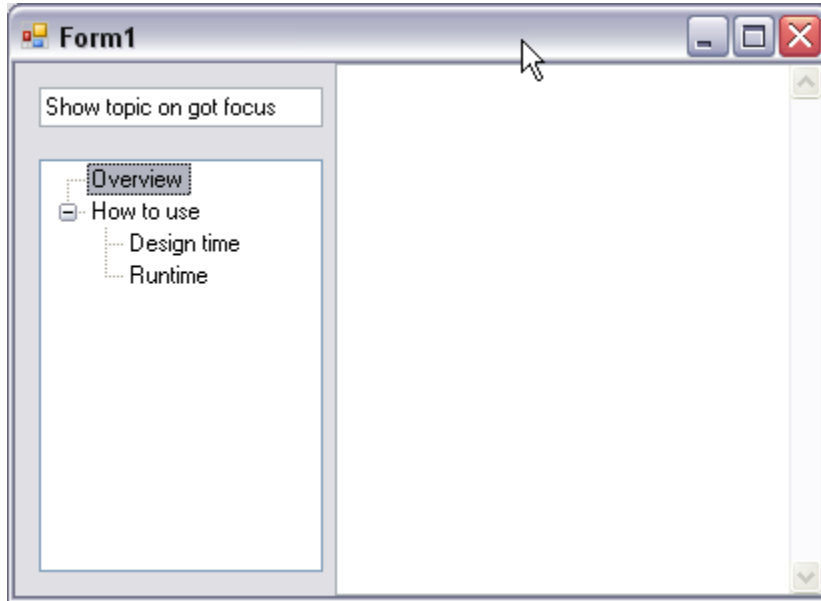
The first two steps of this tutorial explain how software developers can set up their applications for **authoring mode**. Step 3 explains how Help authors can assign topics to controls in **authoring mode**. The final step shows how the Help appears in the application.

Step 1 of 4: Add controls to the Windows form

To set up your new Form with the C1DynamicHelp control, a **TextBox**, and a **TreeView** control, complete the following steps. These steps should be performed by the software developer.

1. [Create a .NET project](#) (page 10) and [add the C1DynamicHelp control to the Toolbox](#) (page 10).
2. From the Toolbox, double-click the C1DynamicHelp control. It docks at the right of your form.
3. Add a **TextBox** control to the form:
 - a. From the Toolbox, double-click the **TextBox** control to add it to your form.
 - b. From the Properties window, set the `textBox1.Text` property to **Show topic on got focus**.

4. Add the **TreeView** control to the form:
 - a. From the Toolbox, double-click the **TreeView** control to add it to your form.
 - b. From the Properties window, add nodes to the **treeView1** according to the following image:



In the next step you will set up the C1DynamicHelp control.

Step 2 of 4: Set up the C1DynamicHelp control

In order to associate topics from your Help file with form controls, you must specify the source Help file.

You can do this from the **C1DynamicHelp Tasks** menu by setting the HelpSource property to any *.chm file or NetHelp *.htm file (use the default.htm or the file name that was used as the base URL for the NetHelp target.). In this example, we will use the **C1Sample.chm** installed with DynamicHelp for WinForms, by default.

Once a source Help file is specified, you can prepare the C1DynamicHelp control to be used in **authoring mode**. This step should be performed by the developer.

1. Click the C1DynamicHelp smart tag (▾) to open the **Tasks** menu.
2. Click the **ellipsis** button next to the HelpSource property.
3. Locate and select the **C1Sample.chm**. It is located in the Tutorials/Data folder, by default.
4. Prepare the C1DynamicHelp control for using in **authoring mode**:
 - a. From the Properties window, set the **Form1.KeyPreview** property to **True**.
 - b. Override the **OnKeyDown** method by adding the following code:
 - Visual Basic

```
' toggle authoring mode when the user hits Ctrl+Shift+A
Protected Overrides Sub OnKeyDown(ByVal e As
System.Windows.Forms.KeyEventArgs)
    If (e.KeyCode = Keys.A And e.Control And e.Shift) Then
        C1DynamicHelp1.AuthoringMode = Not
C1DynamicHelp1.AuthoringMode
```

```

    End If
    MyBase.OnKeyDown(e)
End Sub

```

- C#

```

// toggle authoring mode when the user hits Ctrl+Shift+A
override protected void OnKeyDown(KeyEventArgs e)
{
    if (e.KeyCode == Keys.A && e.Control && e.Shift)
    {
        c1DynamicHelp1.AuthoringMode =
!c1DynamicHelp1.AuthoringMode;
    }
    base.OnKeyDown(e);
}

```

- c. Create a handler for the **Form_Load** event and insert the following code to instruct the C1DynamicHelp control to subscribe the controls to the events for showing topics:

- Visual Basic

```

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    C1DynamicHelp1.TopicMap.Refresh()
End Sub

```

- C#

```

private void Form1_Load(object sender, EventArgs e)
{
    c1DynamicHelp1.TopicMap.Refresh();
}

```

5. Create a handler for the **Form_Closing** event and insert the following code to ask the user whether to save changes made in the topic map:

- Visual Basic

```

Private Sub Form1_FormClosing(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
    If (C1DynamicHelp1.TopicMap.HasChanges) Then
        Dim dr As DialogResult
        dr = MessageBox.Show("Would you like to save the changes you
made to control/topic map?", "C1DynamicHelp Tutorial",
MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question)
        If (dr = DialogResult.Yes) Then
            C1DynamicHelp1.TopicMap.Save()
        ElseIf (dr = DialogResult.Cancel) Then
            e.Cancel = True
        End If
    End If
End Sub

```

- C#

```

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (c1DynamicHelp1.TopicMap.HasChanges)
    {
        DialogResult result = MessageBox.Show("Would you like to save
the changes you made to control/topic map?", "C1DynamicHelp Tutorial",
MessageBoxButtons.YesNoCancel , MessageBoxIcon.Question);
        if (result == DialogResult.Yes)
            c1DynamicHelp1.TopicMap.Save();
    }
}

```

```

else if (result == DialogResult.Cancel)
    e.Cancel = true;
}
}

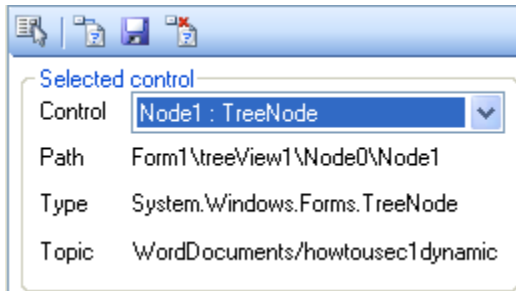
```





No controls have topics associated with them yet. The application can now be used by a Help author to map topics to controls.

Step 3 of 4: Using authoring mode


This step explains how the Help author can activate **authoring mode** and associate topics with controls.

1. Run the application from the software developer and press the **Ctrl+Shift+A** keys together to activate **authoring mode**. The authoring mode panel appears.



2. Map a topic to the **TextBox** control:
 - a. Click **Select control** button  and select **TextBox1**.
 - b. Click the **Attach topic to control** button . The **Select Help Topic** dialog box appears.
 - c. On the **Table of Contents** tab, select the *Glossary of Terms* topic.
 - d. Uncheck the **Use default events** and **Mouse hover** checkboxes so that only **Got focus** is checked.
 - e. Click **OK**. Note that a topic is now displayed only when **TextBox1** gets focus - not when the mouse hovers over it.
3. Map a topic to the **TreeView** nodes:
 - a. Click **Select control** button  and select the **Overview** node.
 - b. Click the **Attach topic to control** button . The **Select Help Topic** dialog box appears.
 - c. On the **Table of Contents** tab, select the *Overview* topic and click **OK**.
 - d. Assign the *How to use CIDynamicHelp* topic to the **How to Use** node using steps a-c.
 - e. Assign the *Design time support* topic to the **Design time** node using steps a-c.
 - f. Assign the *Runtime support* topic to the **Runtime** node using steps a-c. Note that the corresponding topics are displayed when the mouse hovers over each of the **TreeView** nodes.

Note: In **authoring mode** you have the ability to associate topics with objects whose class is not derived from **Component**, such as **TreeNode** or **ListBox** items. This cannot be done at design time, because there is no **Component** to set for the **HelpTopic on CIDynamicHelp1** property.

4. Click the **Save control/topic mapping**  button to save the changes to an .xml file that serves as the topic map.

Note: If you close the application without saving the control/topic mapping, a dialog box will appear, prompting you to save the changes.

The topic map is saved to the same location as the Help file specified in the HelpSource property. It will also have the same name as the specified Help file, only with an .xml extension. In this example, the topic map can be found in the **C1Sample.chm.xml** file. It should now look like this:

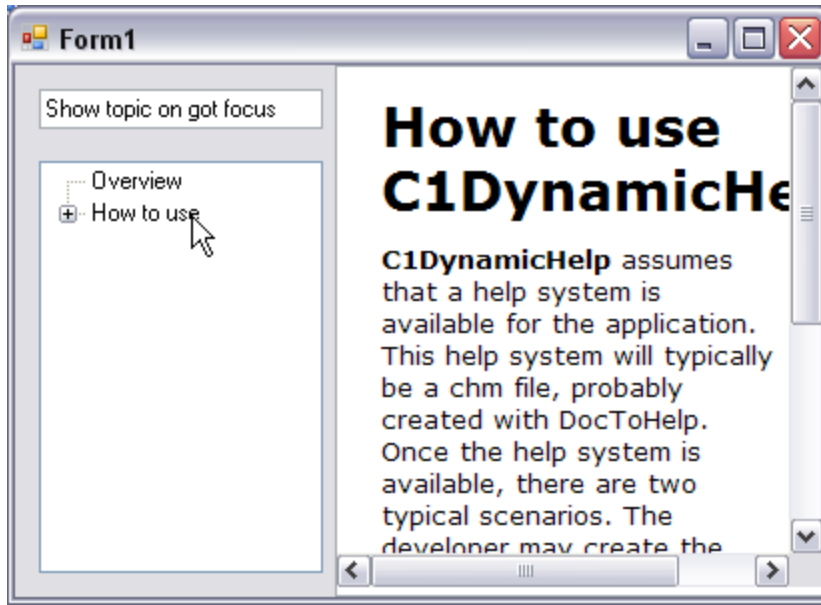
```
<Dictionary>
  <item>
    <key>Form1\textBox1</key>
    <value>WordDocuments/glossaryofterms.htm</value>
    <events useDefaultEvents="False">1</events>
  </item>
  <item>
    <key>Form1\treeView1\Node1</key>
    <value>WordDocuments/howtousecdynamichelp.htm</value>
    <events useDefaultEvents="True">3</events>
  </item>
  <item>
    <key>Form1\treeView1\Node0</key>
    <value>WordDocuments/overview.htm</value>
    <events useDefaultEvents="True">3</events>
  </item>
  <item>
    <key>Form1\treeView1\Node1\Node2</key>
    <value>WordDocuments/designsupport.htm</value>
    <events useDefaultEvents="True">3</events>
  </item>
  <item>
    <key>Form1\treeView1\Node1\Node3</key>
    <value>WordDocuments/runtimesupport.htm</value>
    <events useDefaultEvents="True">3</events>
  </item>
</Dictionary>
```

5. Give the **C1Sample.chm.xml** file, along with the updated Help file(s), if changed, to the software developer.

Step 4 of 4: Run the Application

To run the application, click the **Start Debugging** button and notice the following results. This step should be performed by the developer.

- When the mouse hovers over the **TreeView** nodes, the specified topics are displayed.
- Only when **TextBox1** gets the focus will it display a topic.



Congratulations, you have successfully completed Tutorial 2! In this tutorial you have learned how to:

- Add and set up the C1DynamicHelp control.
- Activate/deactivate **authoring mode**.
- Associate topics to controls on a form in **authoring mode**.
- Save changes in the topic map.