

---

ComponentOne

# Sizer for WinForms

Copyright © 2012 ComponentOne LLC. All rights reserved.

*Corporate Headquarters*

**ComponentOne LLC**

201 South Highland Avenue

3<sup>rd</sup> Floor

Pittsburgh, PA 15206 • USA

**Internet:** [info@ComponentOne.com](mailto:info@ComponentOne.com)

**Web site:** <http://www.componentone.com>

**Sales**

E-mail: [sales@componentone.com](mailto: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 Sizer for WinForms Overview .....	1
Installing Sizer for WinForms.....	1
Sizer for WinForms Setup Files .....	1
System Requirements .....	2
Installing Demonstration Versions.....	2
Uninstalling Sizer for WinForms .....	2
Known Issues .....	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 2.0 Project.....	10
Adding the C1Sizer Components to a Project .....	11
Migrating a C1Sizer Project to Visual Studio 2005.....	12
Key Features.....	17
Design-Time Support.....	18
C1Sizer Smart Tag .....	18
C1SizerLight Smart Tag.....	19
C1Sizer Context Menus .....	19
C1Sizer Grid Editor .....	20
C1Sizer Gradient Editor .....	22
Using the C1SizerLight Component .....	23
Quick Start Tutorial .....	23
Using the C1Sizer Control .....	27

Tutorial 1: Set up the grid, then add the controls.....	28
Tutorial 2: Add the controls, then set up the grid .....	36
Sizer for WinForms Samples.....	40
Sizer for WinForms Task-Based Help .....	43
Add a C1SizerLight Component to a Form in code.....	43
Position Controls on the C1Sizer Grid at Run Time.....	43
Create a Three Dimensional Border for the Rows and Columns .....	45
Store Layout Information for the C1Sizer Control.....	45

# ComponentOne Sizer for WinForms

## Overview

Create resolution-independent applications without having to write any code with **ComponentOne Sizer™ for WinForms**. The powerful **C1Sizer** and **C1SizerLight** components, provided with **Sizer for WinForms**, make this possible. Now you can resize all contained controls proportionally, so your Windows Form retains its appearance at any resolution.

**C1Sizer** is a container control with a powerful grid layout manager that extends the basic layout capabilities provided by the .NET Framework (Dock and Anchor properties). The **C1Sizer** control allows you to define a grid made up of bands and then add controls that snap to these bands. When the **C1Sizer** control is resized, the bands are automatically recalculated, and contained controls move automatically to their new positions. You can set up bands at design time and configure them to act as splitters or to keep their size constant when the control is resized.

For a list of the latest features added to **ComponentOne Studio for WinForms**, visit [What's New in Studio for WinForms](#).

## Installing Sizer for WinForms

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

### Sizer 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:

<b>bin</b>	Contains copies of all binaries (DLLs, EXEs) in the ComponentOne Visual Studio.NET package.
<b>Common</b>	Contains support and data files that are used by many of the demo programs.
<b>C1Sizer</b>	Contains samples and tutorials for <b>ComponentOne Sizer for WinForms</b> .

The **ComponentOne Studio for WinForms Help Setup** program installs integrated Microsoft Help Viewer help to the C:\Program Files\ComponentOne\Studio for WinForms directory in the following folders:

### Samples

You can access samples from the **ComponentOne Sample Explorer**. To view samples, click the **Start** button and then click **ComponentOne | Studio for WinForms | Samples | Sizer Samples**.

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

<b>Common</b>	Contains support and data files that are used by many of the demo programs.
<b>C1Sizer</b>	Contains samples and tutorials for <b>Sizer for WinForms</b>

## System Requirements

System requirements for ComponentOne Studio for WinForms components include the following:

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

## Installing Demonstration Versions

If you wish to try **ComponentOne Sizer for WinForms** and do not have a registration 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 Sizer for WinForms

To uninstall **ComponentOne Studio for WinForms**:

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

To uninstall **ComponentOne Studio for WinForms** integrated help:

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

## Known Issues

### Sizer for Winforms doesn't display correctly at 120 DPI

There can be conflicts between C1Sizer and the Form class at 120 DPI. These problems are caused by both the Form and C1Sizer trying to scale the contents of the Form. It can lead to issues where controls are rendered incorrectly or are not visible at all.

The simple fix to this issue is to tell the Form not to scale its contents. You can do this by setting the Form's **AutoScaleMode** property to **None**.

## 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 Properties window, set the **Build Action** property to **Embedded Resource**.

### ***Using licensed components in Visual C++ applications***

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

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

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

### ***Using licensed components with automated testing products***

Automated testing products that load assemblies dynamically may cause them to display license 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 affected project.
2. Select an instance of the updated component.
3. In the Visual Studio Properties window, change any property. It does not matter which property you change; you can change it back to the previous value.
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**) and run the solution.

#### **Alternative 1: Follow these steps:**

1. Open a new Visual Studio.NET project.
2. Add the updated component to the form.
3. Compile and run the new project.
4. Open the licenses.licx file in the new project.
5. Copy the line that starts with the namespace of the updated component (for example, C1.Win.C1Report) and ends with a public key token.
6. Open the existing, incorrectly licensed project.
7. Open the licenses.licx file in the new project.
5. Paste the line from step 5 into this file (replace the old licensing information with the new).
6. Rebuild the project using the **Rebuild All** option (not just **Rebuild**) and run the solution.

#### **Alternative 2: Follow these steps:**

1. Open the affected project.
2. Delete the licenses.licx file from the project.

3. Add a new instance of the updated component to the form.
4. Rebuild and run the solution. The nag screen should not appear.
5. Remove the newly added component from the form.

Try each of these options multiple times, if necessary. If that still does not help, are you creating any of the controls in code rather than design-time? If so, you must add an entry for the control in the licenses.licx file (see <http://helpcentral.componentone.com/PrintableView.aspx?ID=1886> for more information). Also if this is a website, as opposed to an ASP.NET web application, please try right-clicking the licenses.licx file and selecting "Build Runtime Licenses" from the context menu.

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

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

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

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

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

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

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

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

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

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

#### **Option 2 – Continue to use the components you have.**

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

## Technical Support

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

Some methods for obtaining technical support include:

- **[Online Resources](#)**  
ComponentOne provides customers with a comprehensive set of technical resources in the form of FAQs, samples and videos, Version Release History, searchable Knowledge base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**  
This online support service provides you with direct access to our Technical Support staff via an [online](#)

[incident submission form](#). When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.

- **Product Forums**

ComponentOne's [product forums](#) are available for users to share information, tips, and techniques regarding ComponentOne products. ComponentOne developers will be available on the forums to share insider tips and technique and answer users' questions. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.

- **Installation Issues**

Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the [online incident submission form](#) or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.

- **Documentation**

Microsoft integrated ComponentOne documentation can be installed with each of our products, and documentation is also available online. If you have suggestions on how we can improve our documentation, please email the [Documentation team](#). Please note that e-mail sent to the [Documentation team](#) is for documentation feedback only. [Technical Support](#) and [Sales](#) issues should be sent directly to their respective departments.

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

## Redistributable Files

**ComponentOne Sizer 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.C1Sizer.2.dll
- C1.Win.C1Sizer.4.dll
- C1.Win.C1Sizer.4.Design.dll

Site licenses are available for groups of multiple developers. Please contact [Sales@ComponentOne.com](mailto:Sales@ComponentOne.com) for details.

## About This Documentation

### Acknowledgements

*Microsoft, Visual Studio, Visual Basic, Windows, and Windows Vista 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  
3<sup>rd</sup> 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®](#).

# Namespaces

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

The general namespace for ComponentOne Windows products is **C1.Win**. The namespace for the **C1Sizer** control is **C1.Win.C1Sizer**. The following code fragment shows how to declare a **C1Sizer** control using the fully qualified name for this class:

- Visual Basic

```
Dim Column As C1.Win.C1Sizer.Column
```

- C#

```
C1.Win.C1Sizer.Column Column;
```

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 **Column**, you can use it inside your project without qualification. However, the **C1Sizer** assembly also implements a class called **Column**. So, if you want to use the **C1Sizer** 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 Column object
```

```
Dim MyColumn as Column
```

```
' Define a new C1Sizer.Column object
```

```
Dim NewColumn as C1.Win.C1Sizer.Column
```

- C#

```
// Define a new Column object;
```

```
Column MyColumn;
```

```
// Define a new C1Sizer.Column object;
```

```
C1.Win.C1Sizer.Column NewColumn;
```

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 Column = C1.Win.C1Sizer.Column
Imports MyColumn = C1.Win.C1Sizer.Column
Dim C1 As Column
Dim C2 As My Column
```

- **C#**

```
using Column = C1.Win.C1Sizer.Column;
using MyColumn = C1.Win.C1Sizer.Column;
Column C1;
My C2 Column;
```

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.

**Note:** As a warning about the code examples of this documentation, it is taken for granted that the following statement has been specified. This applies only to small code samples. Tutorials and other longer samples will specify complete qualifiers.

- **Visual Basic**

```
Imports C1.Win.C1Sizer
```

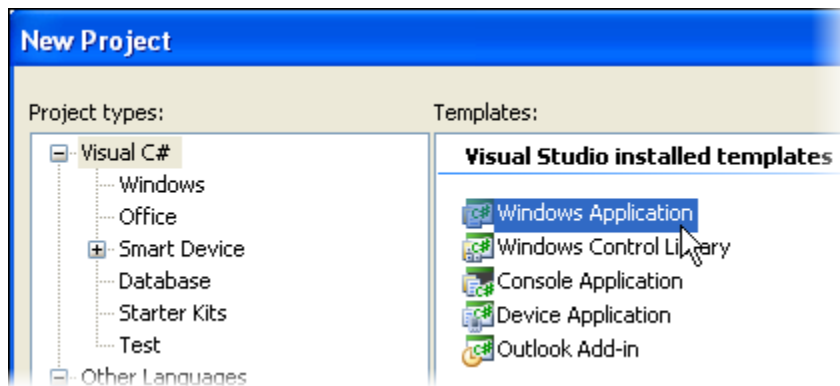
- **C#**

```
using C1.Win.C1Sizer;
```

## Creating a .NET 2.0 Project

To create a .NET 2.0 project, complete the following steps:

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



4. Enter a name for your new application in the **Name** field or you can use the default name, **WindowsApplication1**. Click on the **Browse** button to the right of the **Location** field to specify the location of your new project 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.

## Adding the C1Sizer Components to a Project

When you install ComponentOne Studio for WinForms, the **Create a ComponentOne Visual Studio 2008/2005 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 2008/2005 Toolbox Tab** checkbox during installation, you can manually add ComponentOne controls to the Toolbox at a later time.

**ComponentOne Sizer for .NET 2.0** provides the following controls:

- C1Sizer
- C1SizerLight

To use **C1Sizer**, add these controls to the form or add a reference to the C1.Win.C1Sizer assembly in your project.

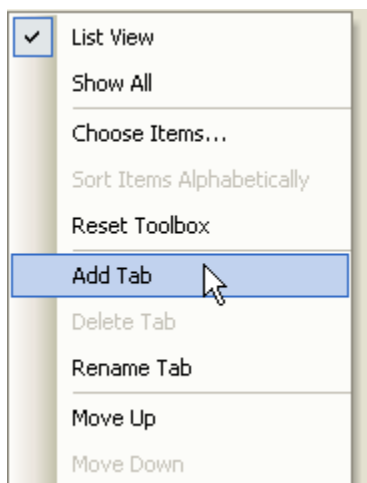
### Manually Adding C1Sizer to the Toolbox

When you install **C1Sizer**, the following **C1Sizer** controls will appear in the Visual Studio Toolbox customization dialog box:

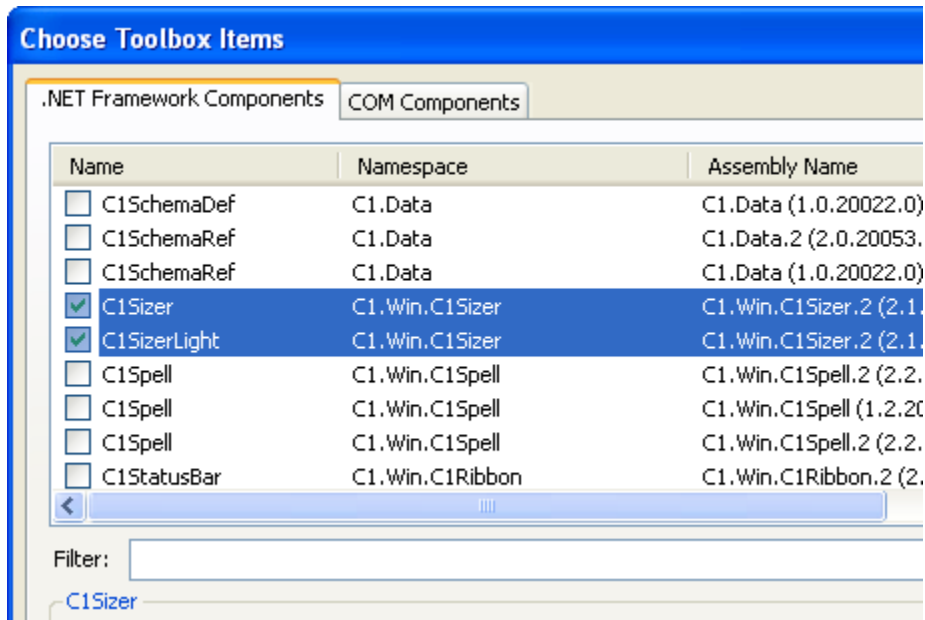
- C1Sizer
- C1SizerLight

To manually add the **C1Sizer** 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 **C1Sizer** controls appear on their own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1Sizer**, 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 select the check boxes for all components belonging to namespace C1.Win.C1Sizer. Note that there may be more than one component for each namespace.



### Adding C1Sizer to the Form

To add **C1Sizer** to a form:

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

### Adding a Reference to the Assembly

To add a reference to the **C1Sizer** assembly:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne C1Sizer** assembly from the list on the **.NET** tab or browse to find the C1.Win.C1Sizer.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.C1Sizer
```

**Note:** This makes the objects defined in the **C1Sizer** assembly visible to the project. See [Namespaces](#) (page 9) for more information.

## Migrating a C1Sizer Project to Visual Studio 2005

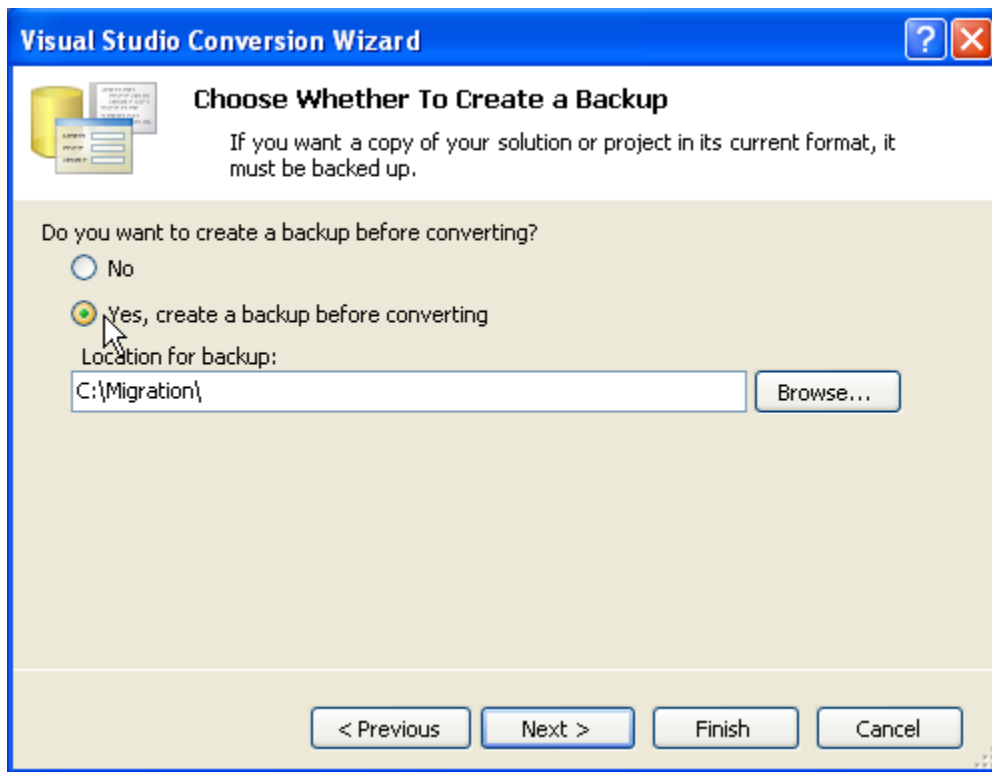
To migrate a project using ComponentOne components to Visual Studio 2005, there are two main steps that must be performed. First, you must convert your project to Visual Studio 2005, which includes removing any references to a previous assembly and adding a reference to the new assembly. Secondly, the .licx file, or licensing file, must be updated in order for the project to run correctly.

### To convert the project:

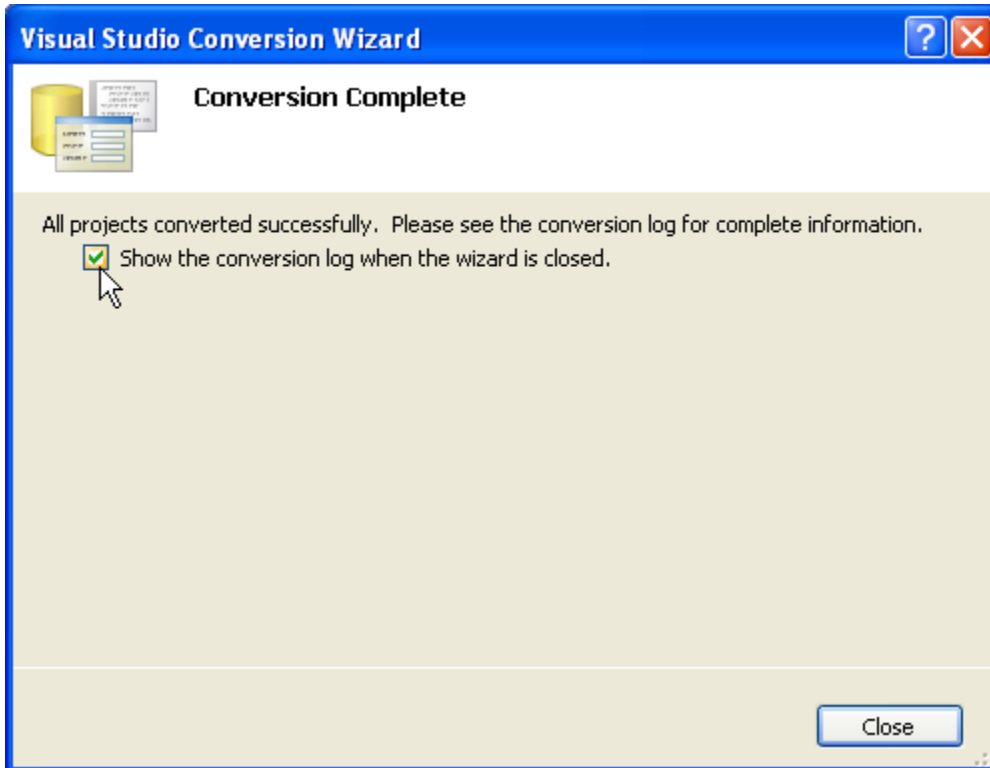
1. Open Visual Studio 2005 and select **File, Open Project**.
2. Locate the **.sln** file for the project that you wish to convert to Visual Studio 2005. Select it and click **Open**. The **Visual Studio Conversion Wizard** appears.



3. Click **Next**.
4. Select **Yes, create a backup before converting** to create a backup of your current project and click **Next**.

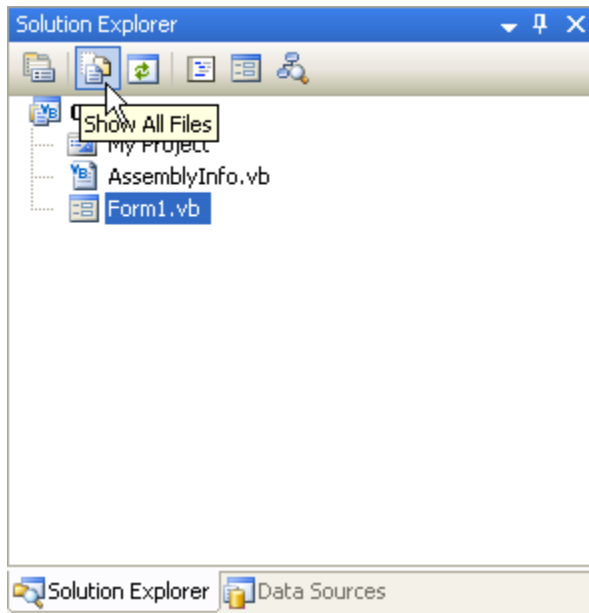


5. Click **Finish** to convert your project to Visual Studio 2005. The **Conversion Complete** window appears.
6. Click **Show the conversion log when the wizard is closed** if you want to view the conversion log.



7. Click **Close**. The project opens. Now you must remove references to any of the previous ComponentOne .dlls and add references to the new ones.
8. Go to the Solution Explorer (**View | Solution Explorer**), select the project, and click the **Show All Files** button.

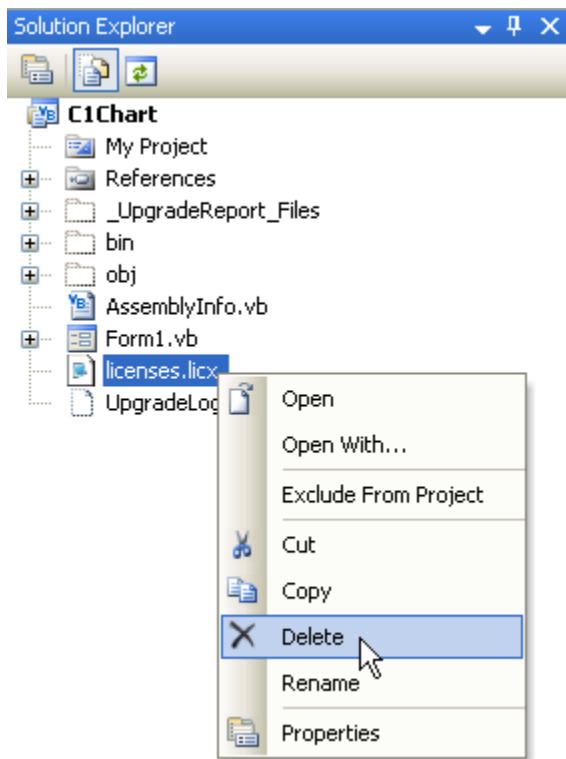
**Note:** The Show All Files button does not appear in the Solution Explorer toolbar if the Solution project node is selected.



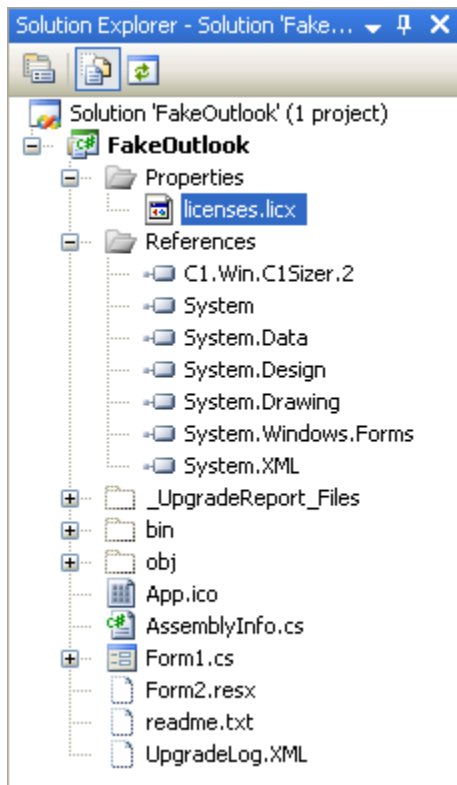
9. Expand the **References** node, right-click C1.Win.C1Sizer and select **Remove**.
10. Right-click the **References** node and select **Add Reference**.
11. Locate and select **C1.Win.C1Sizer.2.dll**. Click **OK** to add it to the project.

**To update the .licx file:**

1. In the Solution Explorer, right-click the **licenses.licx** file and select **Delete**.



2. Click **OK** to permanently delete **licenses.licx**. The project must be rebuilt to create a new, updated version of the .licx file.
3. Click the **Start Debugging** button to compile and run the project. The new .licx file may not be visible in the Solution Explorer.
4. Select **File | Close** to close the form and then double-click the **Form.vb** or **Form.cs** file in the Solution Explorer to reopen it. The new **licenses.licx** file appears in the list of files.



The migration process is complete.

## Key Features

**ComponentOne Sizer for WinForms** includes the following features:

- **Create a Gradient Background**  
With the easy-to-use C1Sizer Gradient Editor that is accessible from the "Edit Gradient" Smart Tag, you can quickly add a gradient to the sizer panel and customize the gradient settings.
- **Update your Form with Rounded Corners**  
The new Border property gives you the flexibility to add rounded corners to the sizer and modify the color, thickness, and more.
- **Flexibility when Adding Images to the Background**

With the new image properties you have more power and flexibility for using images as part of the background. You can now control the alignment and scale of the images.

- **Arrange Controls on the Form with Ease**

The C1Sizer control act like a piece of graph paper where each control will occupy one or more of the grid boxes on the form. The graph allows you to neatly layout the controls on the form. You have the flexibility to set up the grid before or after you create the child controls.

- **Resize all Contained Controls Proportionally**

The C1SizerLight component is a non-visual component that keeps track of a form's size and position. When the form is resized, the C1SizerLight component resizes all contained controls proportionally, so the form retains its appearance at any resolution. C1SizerLight can also resize the fonts on all or some of the contained controls.

- **Easy-to-use Grid Editor**

Efficiently set up your form with the Grid Editor. Simply add one or more C1Sizer controls to the form, and set their Dock property to fill the form. Then use the Grid Editor to set up a grid layout, and add controls which will snap to the grid. You can also set the number and dimension of the grid's bands (rows and columns), and also specify which bands should act as splitters, and which should have fixed dimensions.

## Design-Time Support

C1Sizer provides customized context menus, smart tags, and a designer that offers rich design-time support and simplifies working with the object model.

The following sections describe how to use **C1Sizer's** design-time environment to configure **C1Sizer's** controls/components.

### C1Sizer Smart Tag

In Visual Studio, the **C1Sizer** control includes a smart tag. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in each component/command.

The **C1Sizer** control provides quick and easy access to the **C1Sizer Grid Editor** and common properties through its smart tag.

To access the **C1Sizer Tasks** menu, click on the smart tag (▾) in the upper-right corner of the **C1Sizer** control. This will open the **C1Sizer Tasks** menu.



The **C1Sizer Tasks** menu operates as follows:

#### About C1Sizer

Clicking on the **About** item displays the **About ComponentOne C1Sizer** dialog box, which is helpful in finding the version number of **C1Sizer** and online resources.

## Edit Gradient

Clicking on the **Edit Gradient** item opens the **C1Sizer Grid Editor**. For more information about the C1Sizer Gradient Editor's elements, see [C1Sizer Gradient Editor](#) (page 22).

## Edit Grid

Clicking on the **Edit Grid** item opens the **C1Sizer Grid Editor**. For more information about the C1Sizer Grid Editor's elements, see [C1Sizer Grid Editor](#) (page 20).

## Auto Grid

Clicking on the **Auto Grid** item clears all unused bands in the grid that don't have any controls attached to them.

## Clear Grid

Clicking on the **Clear Grid** item removes the bands from the grid.

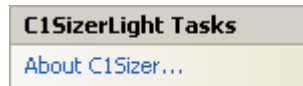
## Dock in parent container

Clicking on the **Dock in parent container** link docks the **C1Sizer** control inside its parent container.

# C1SizerLight Smart Tag

The **C1SizerLight** component provides quick and easy access to the **About ComponentOne C1Sizer** dialog box through its smart tag.

To access the **C1SizerLight Tasks** menu, click on the smart tag (▾) in the upper right corner of the **C1SizerLight** component. This will open the **C1SizerLight Tasks** menu.



The **C1SizerLight Tasks** menu operates as follows:

### About C1Sizer

Clicking on the **About** item displays the **About ComponentOne C1Sizer** dialog box, which is helpful in finding the version number of **C1Sizer** and online resources.

# C1Sizer Context Menus

**C1Sizer** has additional commands with each context menu that Visual Studio provides for all .NET controls.

### C1Sizer Context Menu

Right-click anywhere on the list to display the **C1Sizer** context menu.

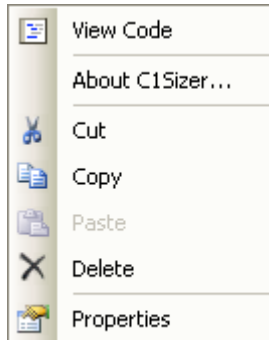
The following table provides a brief description of the custom commands added by **C1Sizer**:

Commands	Description
<b>About C1Sizer</b>	Displays the <b>About ComponentOne C1Sizer</b> dialog box, which is helpful in finding the version number of <b>C1Sizer</b> and online resources.
<b>Edit Grid</b>	Opens the <b>C1Sizer Grid Editor</b> .

<b>Edit Gradient</b>	Opens the C1Sizer Gradient Editor.
<b>Auto Grid</b>	Clears all unused bands in the grid that don't have any controls attached to them.
<b>Clear Grid</b>	Removes the bands from the grid.

### C1SizerLight Context Menu

Right-click anywhere on the **C1SizerLight** component to display the **C1SizerLight** context menu.



The following table provides a brief description of the custom commands added by **C1SizerLight**:

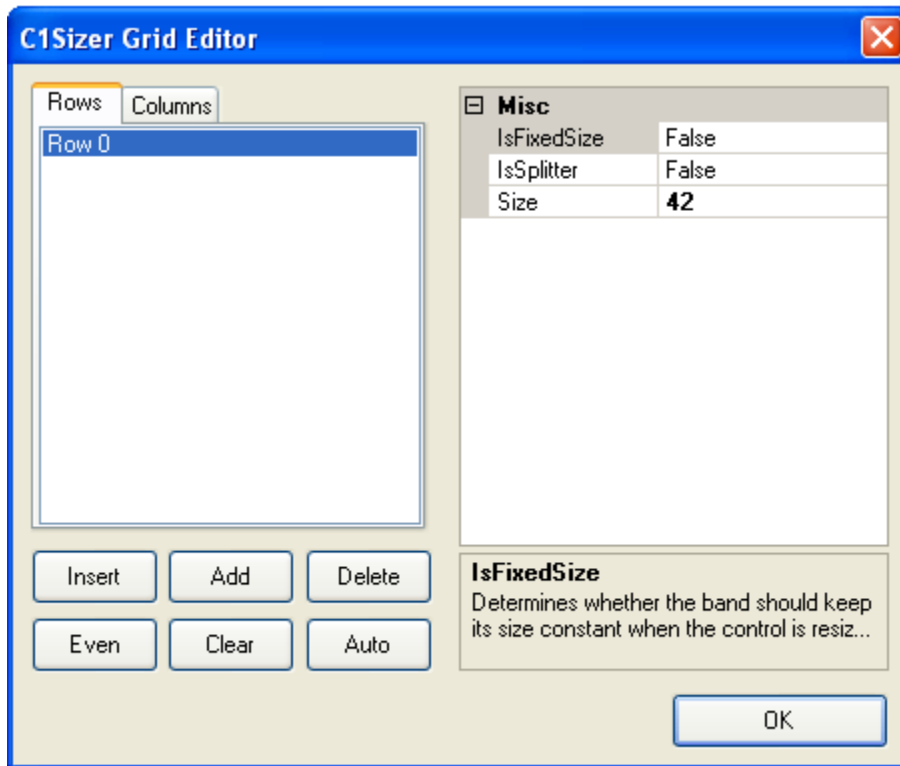
<b>Commands</b>	<b>Description</b>
<b>About C1Sizer</b>	Displays the <b>About ComponentOne C1Sizer</b> dialog box, which is helpful in finding the version number of <b>C1Sizer</b> and online resources.

## C1Sizer Grid Editor

**C1Sizer** has a **Grid Editor** to edit or add bands to rows and columns of the grid at design time.

### To access the C1Sizer Grid Editor:

Open the **C1Sizer Tasks** menu and click on the **Edit Grid** item or right-click the **C1Sizer** control and select **Edit Grid** from its context menu.



## Tabs

**C1Sizer Grid Editor** consist of two important tabs: **Rows** and **Columns**. Both tabs share the same command buttons and properties. Click the **Columns** tab to create or modify the bands in the *Columns* of the grid. Click the **Rows** tab to create or modify the bands in the *Rows* of the grid.

## Command Buttons

The following command buttons are available in both tabs:

Command	Description
<b>Insert</b>	Inserts a new band at the selection.
<b>Add</b>	Adds a new band to the collection.
<b>Delete</b>	Deletes the selected bands.
<b>Even</b>	Makes all bands the same size.
<b>Clear</b>	Removes all bands.
<b>Auto</b>	Creates bands based on child control.
<b>OK</b>	Updates the changes and closes the C1Sizer Grid Editor.

## Properties

The following properties are available in both tabs:

Property	Description
----------	-------------

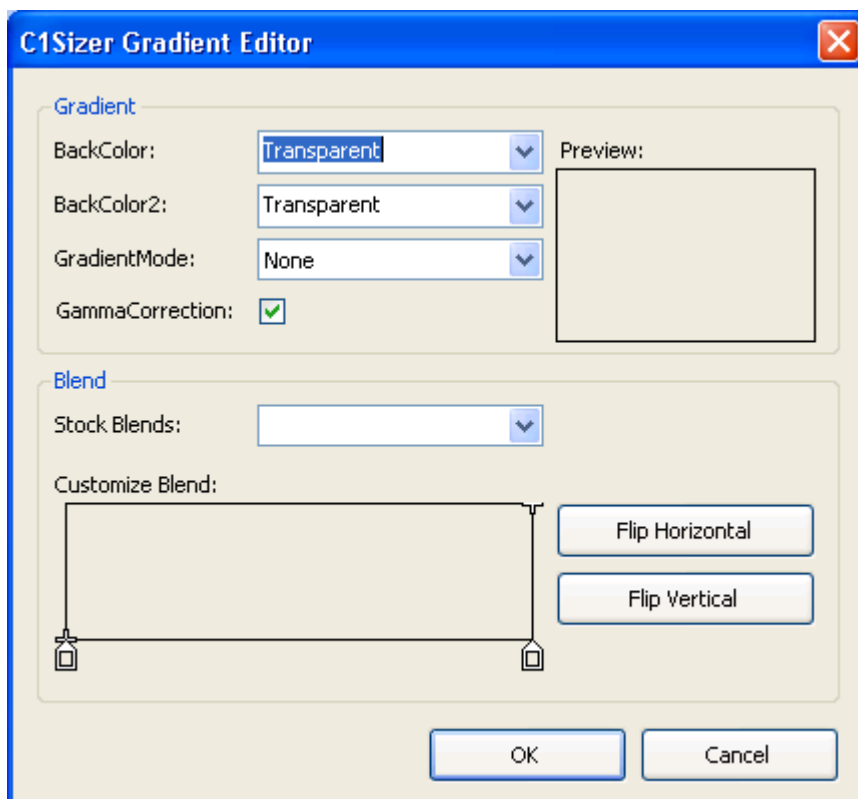
<b>IsFixedSize</b>	Determines whether the band should keep its size constant when the control is resized. The default value is False.
<b>IsSplitter</b>	Determines whether the band acts like a splitter (can be resized with the mouse at run time). The default value is False.
<b>Size</b>	Gets or sets the row height or column width in pixels.

## C1Sizer Gradient Editor

**C1Sizer** has a **Gradient Editor** to add gradients to the **C1Sizer** grid at design time.

To access the **C1Sizer Gradient Editor**:

Open the **C1Sizer Tasks** menu and click on the **Edit Gradient** item or right-click the **C1Sizer** control and select **Edit Gradient** from its context menu.



### Gradient Group

**C1Sizer Gradient Editor** consist of two important groups: **Gradient** and **Blend**. The Gradient group consist of BackColor and Gradient properties and a Preview window so you can preview the gradient updates you make to the **C1Sizer** control.

The Gradient group consists of the following properties:

Property	Description
C1Sizer.BackColor	Gets or sets the Drawing.Color used to paint the background.

Gradient.BackColor2	Gets or sets the secondary color used to build the background gradient.
GradientMode	Specifies the background gradient mode.
Gradient.GammaCorrection	Gets or sets whether to apply gamma correction to the background gradient.

## Blend Group

The Blend group enables you to choose from a stock blend and customize the blend by flipping the blend colors horizontally or vertically.

# Using the C1SizerLight Component

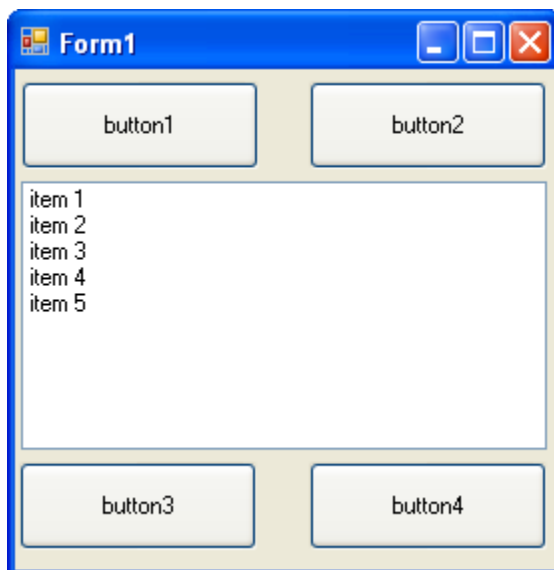
C1SizerLight is a non-visual component. When you add it to a form, it keeps track of the form's size and position. When the form is resized, **C1SizerLight** resizes all contained controls proportionally, so the form retains its appearance at any resolution. **C1SizerLight** can also resize the fonts on all or some of the contained controls.

Using the **C1SizerLight** component is extremely easy. Start by creating a Windows Form as usual (or open an existing one), and simply add a **C1SizerLight** component to it. Then run the project and resize the form. All controls on the form will be automatically resized with the form (including their fonts).

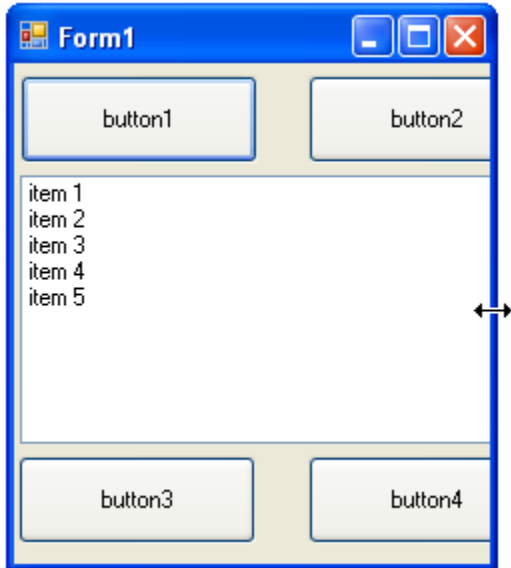
## Quick Start Tutorial

This Quick Start tutorial shows how the C1SizerLight component functions. It also demonstrates how the C1SizerLight.ResizeFonts property and C1SizerLight.ResizingFont event work.

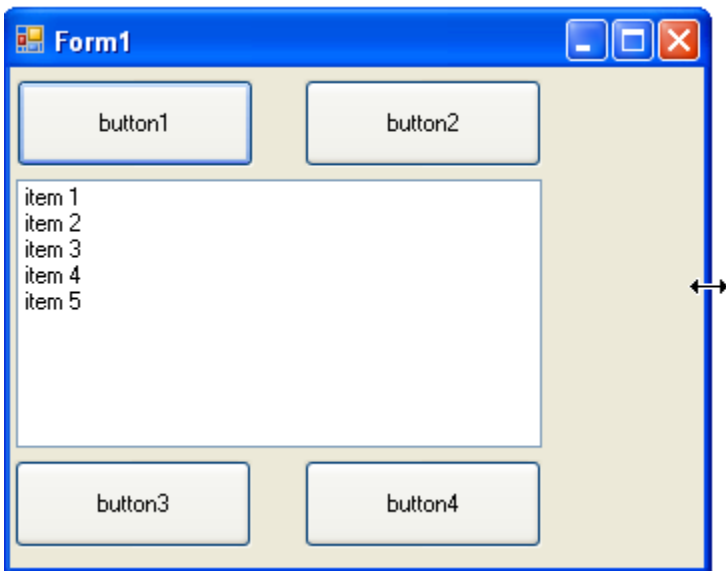
To experiment with the **C1SizerLight** component, open Visual Studio and create a new .NET project, add the **C1SizerLight** component to the form, then add the buttons and list box on the form so it looks like the following:



Now run the project and resize the form. If you make it smaller, parts of the controls will be clipped.

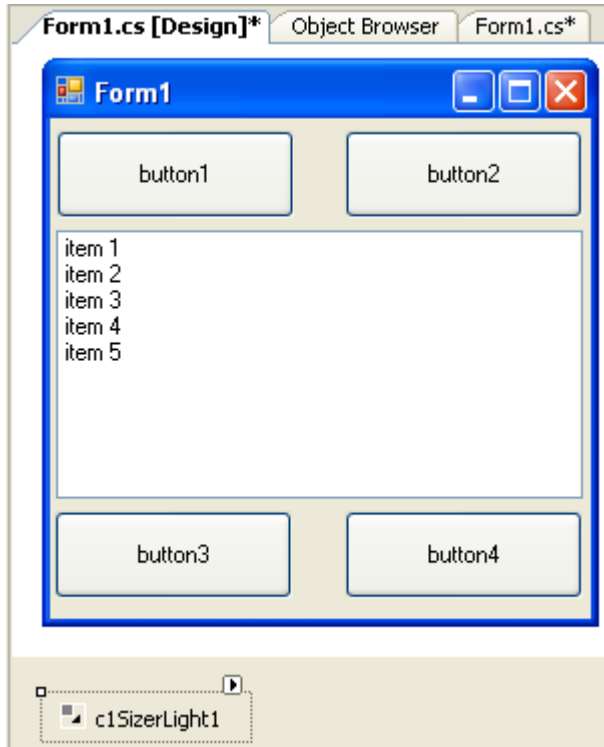


If you make it bigger, you will see an empty gray area.

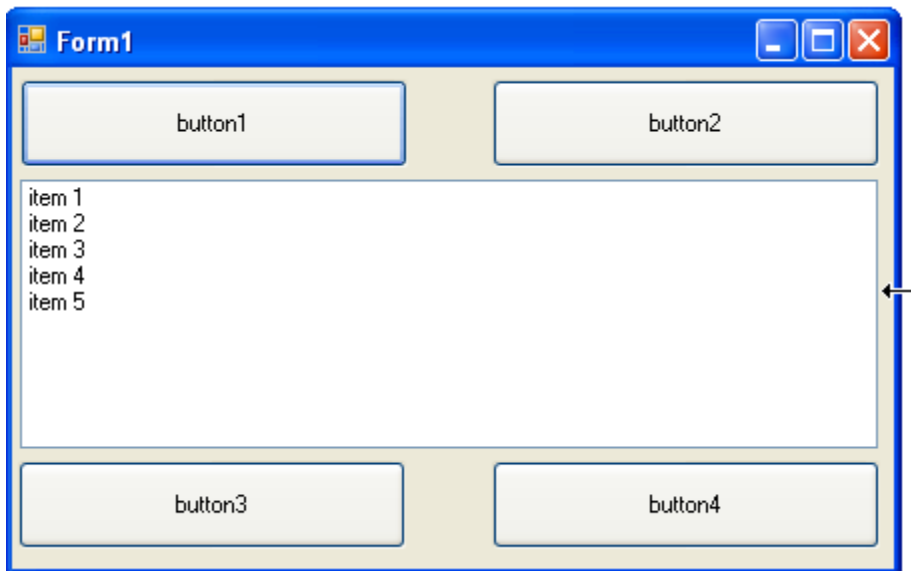


You could improve the controls' appearance on the form using the **Dock** and **Align** properties on the controls, but there is no combination of these two properties that will prevent the controls from overlapping each other or leaving blank areas on the form. Also, depending on your monitor's resolution, the fonts may appear very small when you enlarge the form.

Instead of making the form size fixed and giving up the resize feature, add a `C1SizerLight` component to the form. `C1SizerLight` is a non-visual component, so it will appear in the tray area below the form.



You do not have to set any properties or write any code, just run the project again and resize the form. All contained controls will automatically resize and the form will retain its aspect:

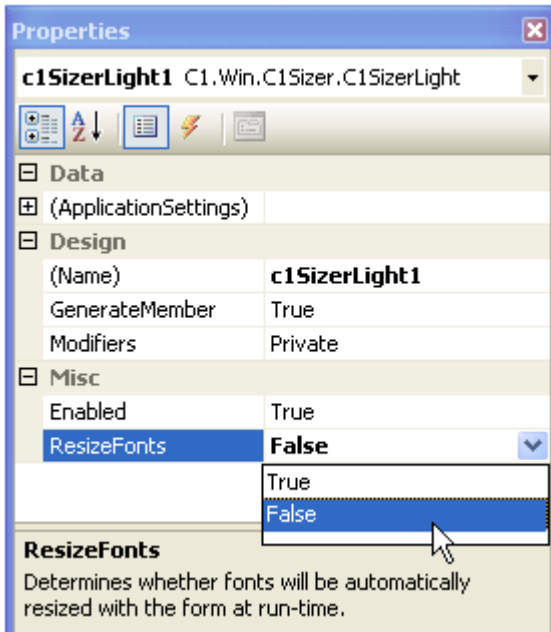


### Resizing Fonts

Notice that when you resized the form, the fonts in the button controls were resized to accommodate the new size. This is the default behavior provided by `C1SizerLight`. In some cases, you may want to prevent that and keep all the original font sizes. You can do this using the `C1SizerLight.ResizeFonts` property. Set `C1SizerLight.ResizeFonts` to **False** and the fonts will not resize.

To set the `C1SizerLight.ResizeFonts` to **False** at design-time use the following steps:

1. Select **C1SizerLight** in the **Properties** sheet.
2. Set the `C1SizerLight.ResizeFonts` to **False**.



You may also want to prevent font resizing in specific controls. For example, it would probably be a good idea to keep the font size constant for the list control on our sample form, and update the fonts on the buttons only. (In general, controls that can scroll their contents do not need the font resizing feature). To do this, you need to handle the `C1SizerLight.ResizingFont` event and cancel it for some controls. For example, the following event handler will cause the component to resize fonts in button controls only:

- Visual Basic

```
Private Sub C1SizerLight1_ResizingFont(ByVal sender As Object, _
    ByVal e As C1.Win.C1Sizer.C1SizerLightEventArgs) Handles _
    C1SizerLight1.ResizingFont

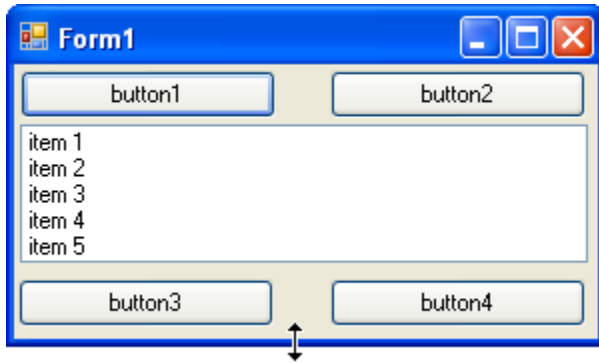
    If Not (e.Control.GetType() Is GetType(Button)) Then
        e.Cancel = True
    End If

End Sub
```

- C#

```
private void c1SizerLight1_ResizingFont(object sender,
    C1.Win.C1Sizer.C1SizerLightEventArgs e)
{
    if (!(e.Control is Button))
        e.Cancel = true;
}
```

If you run the project again, you will notice that the **ListBox** control retains the original font when the form is resized.



### Docked and Nested Controls

Although **C1SizerLight** provides easy-to-use power, you may still want to use the native layout capabilities in the .NET framework. For example, if you add a **ToolBar** or **StatusBar** control to the form, you probably want those controls docked to the top or bottom of the form, and **C1SizerLight** should not change their dimensions.

That is why **C1SizerLight** will not resize any docked controls. It will take into account the fact that they are docked and will reduce the form's client area accordingly, so all non-docked controls will still be resized correctly.

This makes it easy to use **C1SizerLight** and still takes advantage of the layout capabilities built into the .NET framework.

### MDI Forms

**C1SizerLight** also works with MDI child forms. Simply add a **C1SizerLight** component to each child form and they will be resized proportionally, just like any regular form.

## Using the C1Sizer Control

**C1Sizer** is a container control with a powerful grid layout manager that extends the basic layout capabilities provided by the .NET framework (**Dock** and **Anchor** properties). The **C1Sizer** control allows you to define a grid made up of bands, then add controls that snap to these bands. When the **C1Sizer** control is resized, the bands are automatically recalculated, and contained controls move automatically to their new positions. You can set up bands at design time and configure them to act as splitters, or to keep their size constant when the control is resized.

Think of **C1Sizer** as a piece of graph paper. Each component will occupy one or more of the little boxes on the paper. This is similar to tables in word processors such as Microsoft Word™, where you can merge adjacent cells, and to Java's Grid Bag Layout, which works in a similar way.

There are two basic approaches to using the **C1Sizer** control in your projects: You can set up the grid before or after you create the child controls. For more information how to set up the grid before you create the child controls, see [Tutorial 1: Set up the grid, then add the controls](#) (page 28) or how to set up the grid after you create the controls, see [Tutorial 2: Add the controls, then set up the grid](#) (page 36).

The following table lists the controls that require an additional property other than the **Band.Bounds** property:

Control	Property Setting
<b>TextBox</b>	The <b>TextBox</b> control allows you to set its height only if <b>MultiLine = True</b> . Otherwise the height is set automatically based on the <b>Font</b> .
<b>ComboBox</b>	The <b>ComboBox</b> control allows you to set its height only if <b>DrawMode = OwnerDraw</b> .
<b>ListBox</b>	The <b>ListBox</b> control allows you to set its height (exactly) only if <b>IntegralHeight = False</b> .

The next section includes two tutorials that illustrate some of the main features in the **C1Sizer** control. The tutorials walk you through the creation of several simple projects, describing each step in detail. The distribution CD contains more sophisticated samples that can be used as a reference.

Here is a brief overview of the tutorials that follow:

Name	Description
Tutorial 1	This tutorial provides a thorough introduction to the primary features of the <b>C1Sizer</b> control. It shows how to set up the grid by inserting columns and rows, adding labels and data entry controls to the grid, cleaning up unused bands, setting up the splitters, and creating fix-sized bands.
Tutorial 2	This tutorial also provides an introduction to the typical usage of the <b>C1Sizer</b> control. It shows how to add the controls and then set up the grid.

## Tutorial 1: Set up the grid, then add the controls

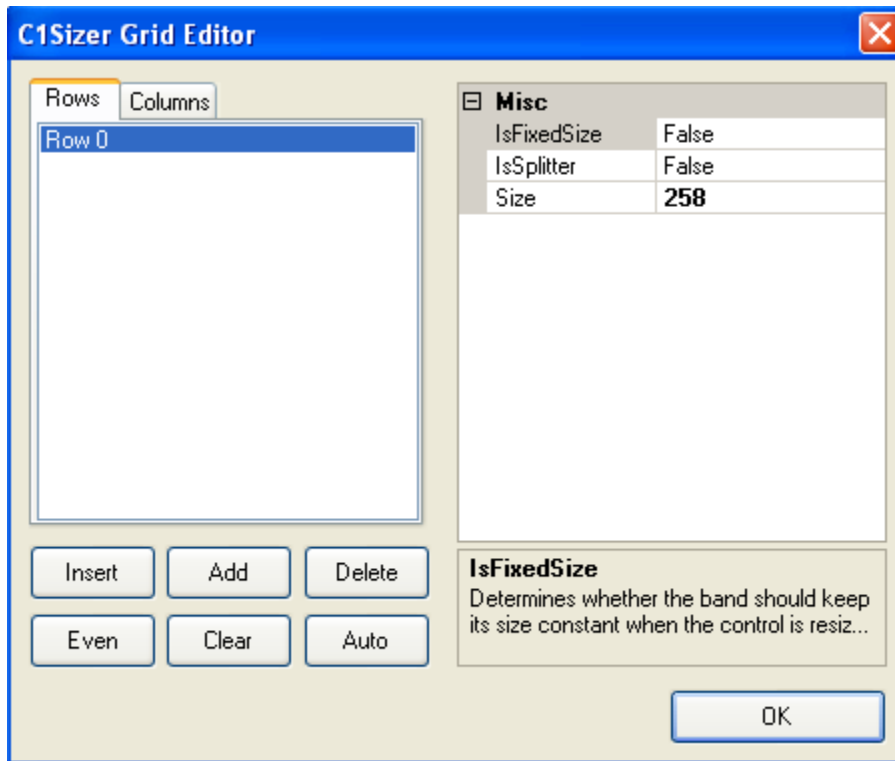
In this tutorial you'll learn how to use elements and properties in the **C1Sizer Grid Editor** to set up a grid. This tutorial demonstrates the following **C1Sizer** features:

- Inserting/deleting rows and columns
- Arranging controls on the grid
- Setting the **C1Sizer.SplitterWidth** and **C1Sizer.BorderWidth** properties
- Applying **AutoGrid** to clean up unused bands
- Setting the **Band.IsFixedSize** and **Band.IsSplitter** properties

To set up the grid for **C1Sizer**, complete the following steps:

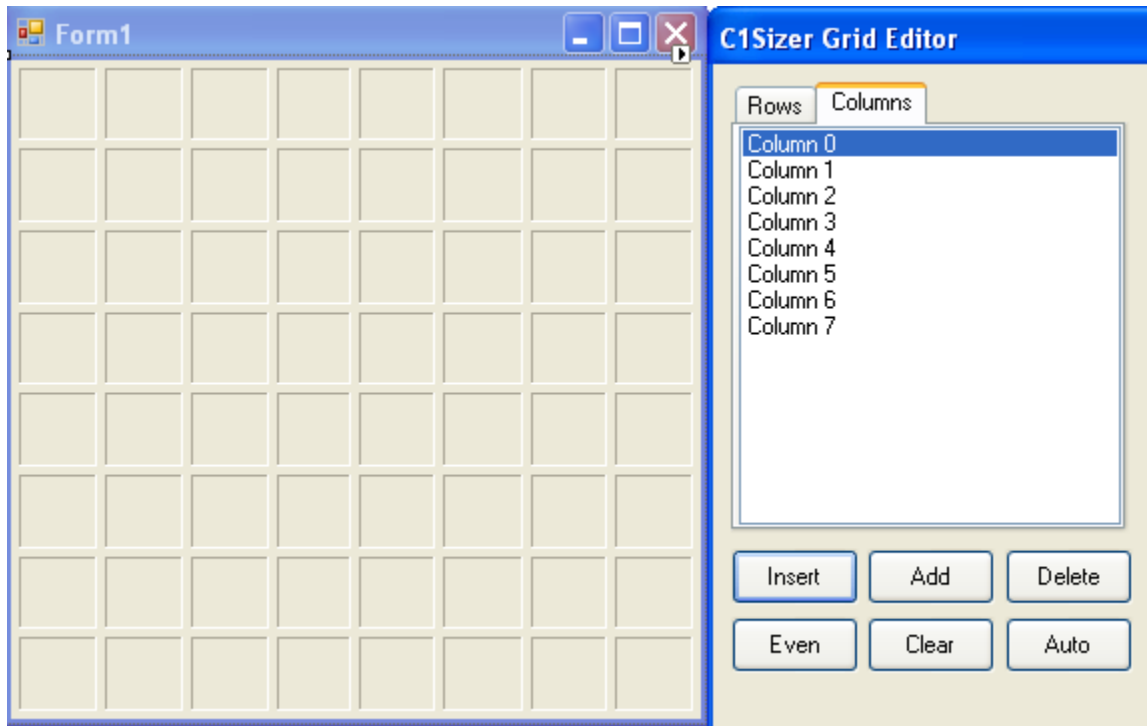
1. Open Visual Studio and create a new project, then add a **C1Sizer** control to the form and set its **Dock** property to **DockStyle.Fill** so it takes up the whole form. At this point, the **C1Sizer** does not have a grid layout set up yet, or any child controls, so it will display a message with brief instructions on how to use the control, but this can be ignored for now.
2. Now, right-click the **C1Sizer** control and select the **Edit Grid** option from the menu. This will bring up the **C1Sizer Grid Editor** dialog box so you can set up the grid layout.

The **C1Sizer Grid Editor** appears like the following:



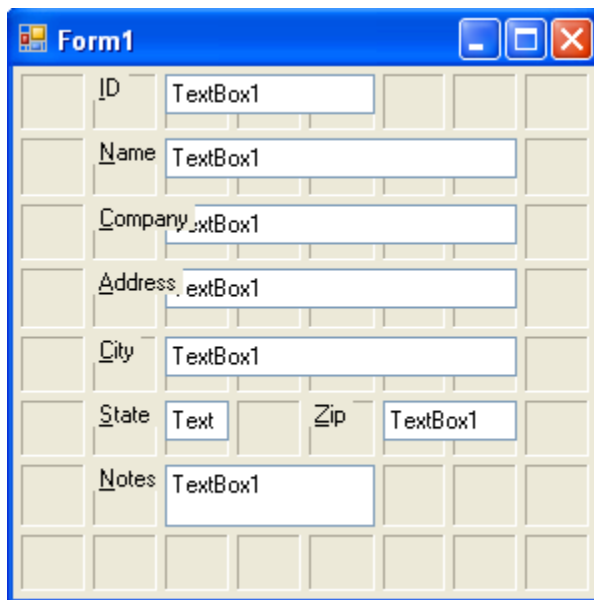
3. Position the editor next to the grid so you can see the effect of the changes on the control. Next, click the **Insert** button until the grid has eight rows. Then, click the **Columns** tab and click **Insert** again until the grid has eight columns.

This is what the control will look like at design time:



The grid is displayed at design time as a series of indented rectangles. Click **OK** to dismiss the editor, and add some controls to the form. Notice how the controls snap to the grid layout. You can resize any control and make it span any number of rows and columns.

- To follow the tutorial, add eight text boxes and eight label controls, and arrange them so the form looks like the following:

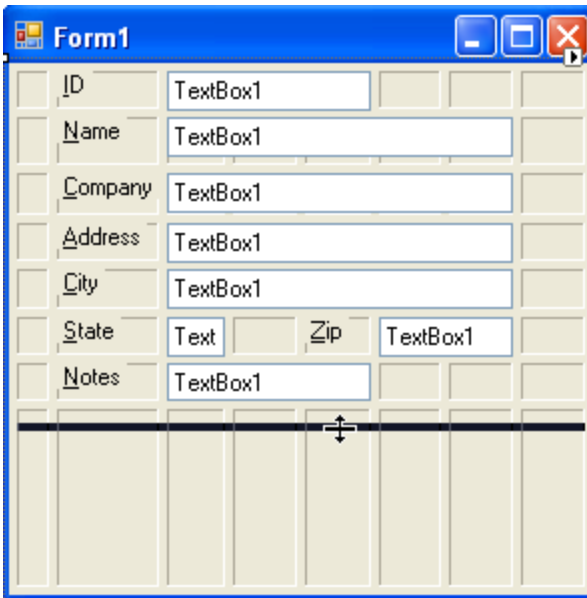


**Note:** Remember to set the **MultiLine** property of the textbox controls to True, or they will not resize vertically.

Also, when you place a text box control on the form it automatically expands to three columns.

5. Select the cursor over the second column and drag it to the left so there is enough space for the labels, then select the cursor over each row and drag it upward to make the height of each row smaller.

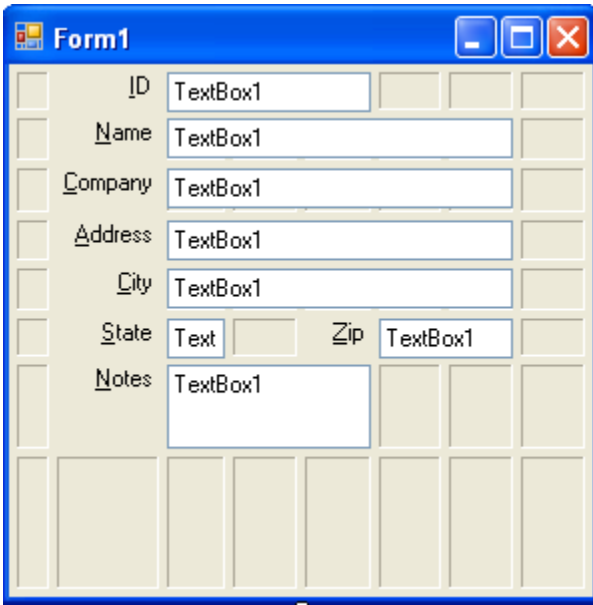
The following picture illustrates this effect:



The image shows a Windows form titled "Form1" with a grid layout. The grid has 8 rows and 5 columns. The labels in the first column are "ID", "Name", "Company", "Address", "City", "State", "Zip", and "Notes". The text boxes in the second column are labeled "TextBox1". The "State" label is followed by a "Text" box, and the "Zip" label is followed by a "TextBox1" box. A vertical double-headed arrow is positioned over the second column, indicating a resize operation.

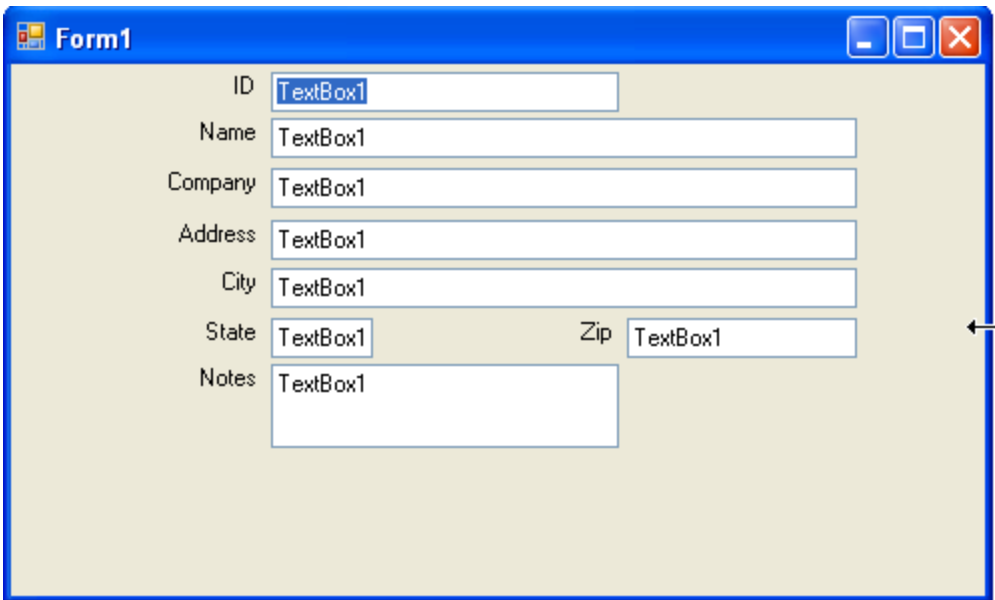
6. Change each label's **AutoSize** property to **False**, then change each label's **TextAlign** property to **TopRight**.

The labels will appear like the following:



If you think the controls are too close to each other or too far apart, change the `C1Sizer.SplitterWidth` and `C1Sizer.BorderWidth` properties to achieve the effect you want.

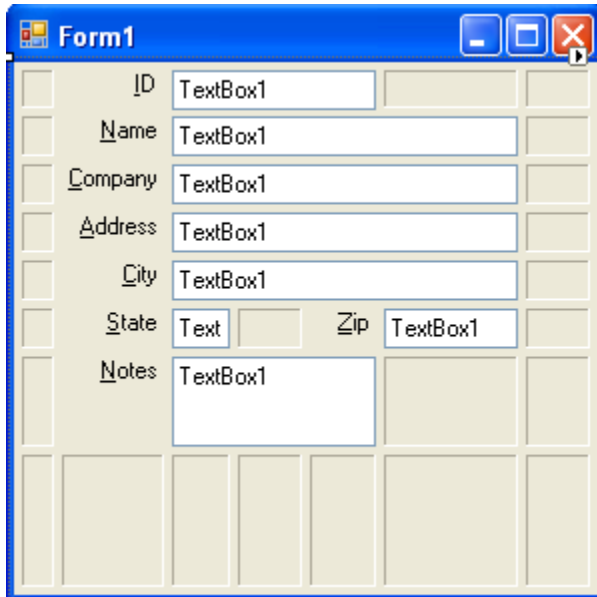
If you run the project now, you will see that when you resize the form, the text boxes and labels are automatically resized.



### Cleaning up the grid

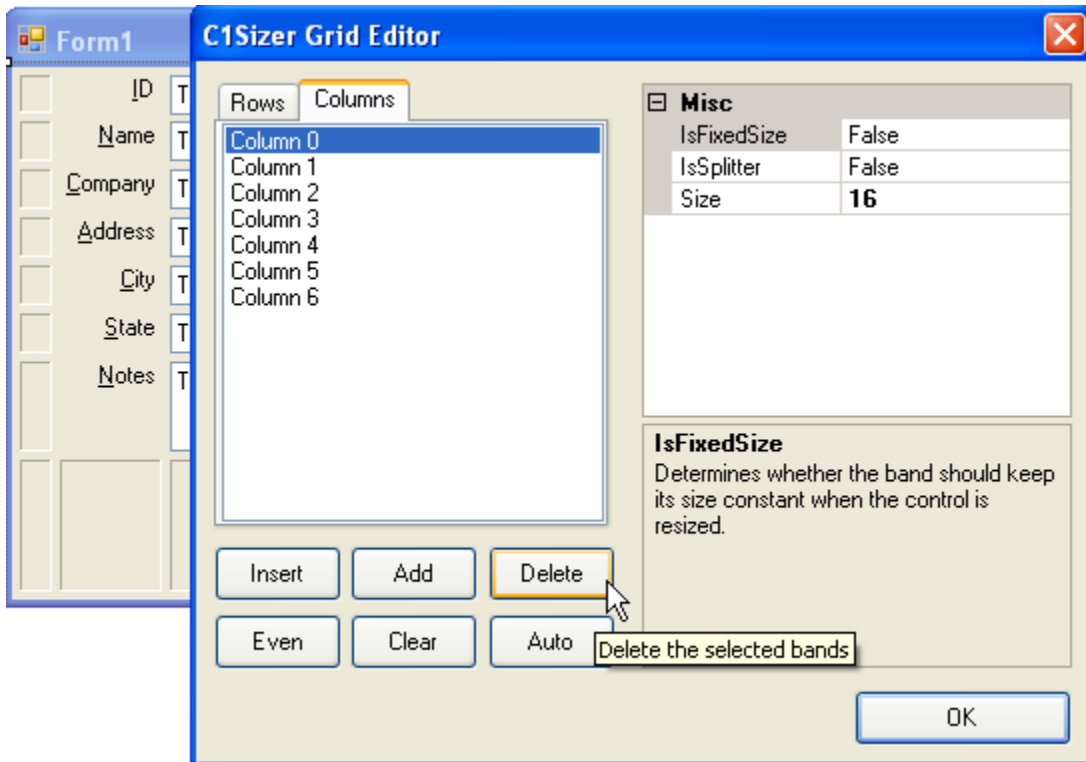
Right now the grid has eight rows and eight columns. Most of these bands are being used (controls are attached to them), but not all. To clean up the layout and get rid of unused bands, right click the `C1Sizer` control and select the `AutoGrid` option from the menu. This will clear any bands that have no controls attached to them.

The `C1Sizer` appears like the following:

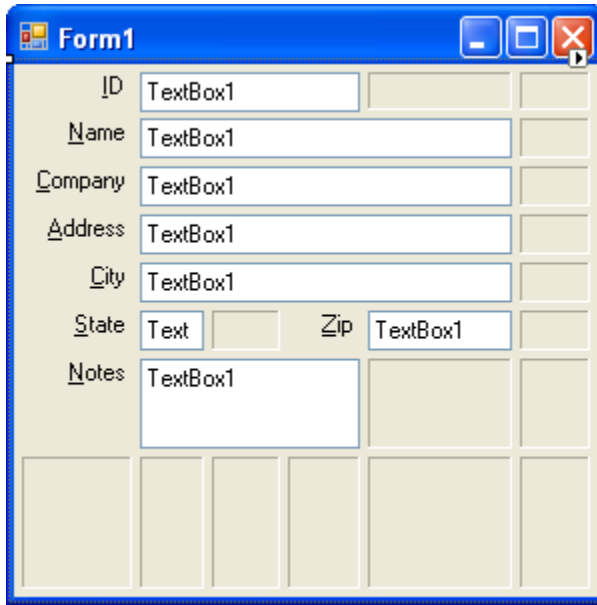


### Deleting columns and rows

To remove the first column, **Column 0**, from the **C1Sizer Grid Editor**, right-click the **C1Sizer** control and select the **Edit Grid** option from the menu. The **C1Sizer Grid Editor** appears. Select the **Columns** tab and click on the **Delete** button and click **OK**.

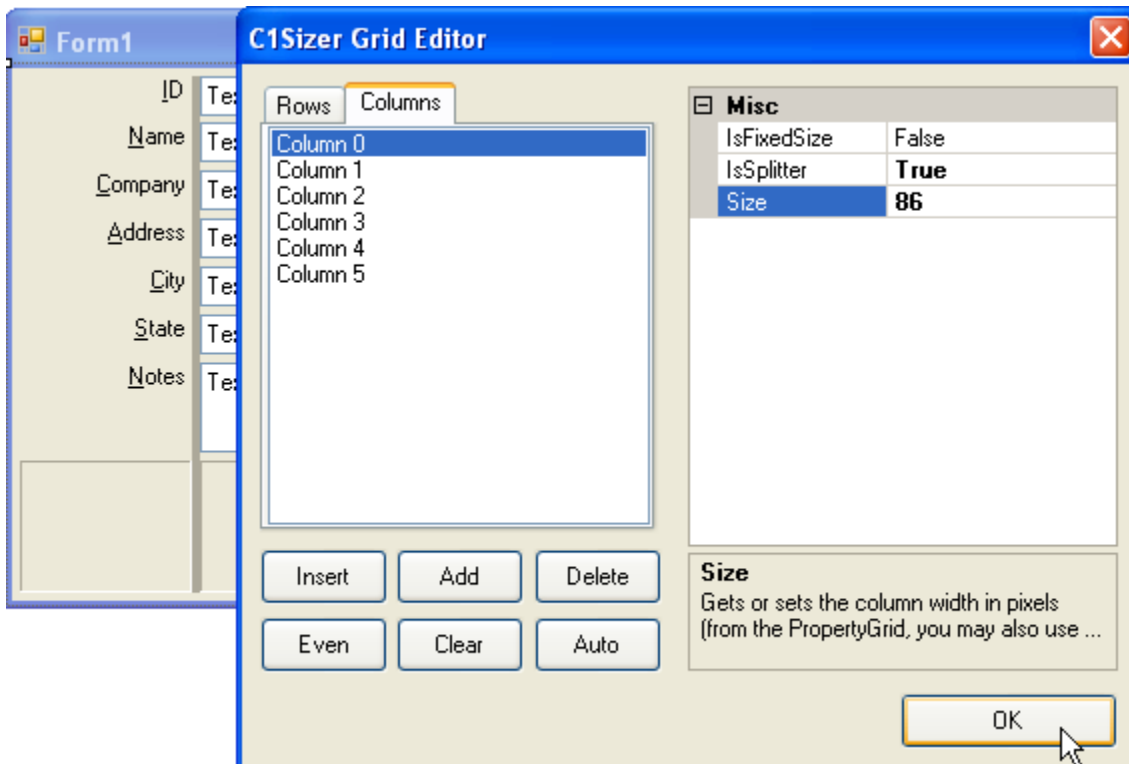


The **C1Sizer** grid appears like the following image at design time.



### Setting up a splitter

Each band (row or column) on the grid can be configured to act as a splitter. For example, if you wanted to allow users to control the size of the label area on the left, you would bring up the grid editor, select the first column and set the `Band.IsSplitter` property to **True**, as shown below:



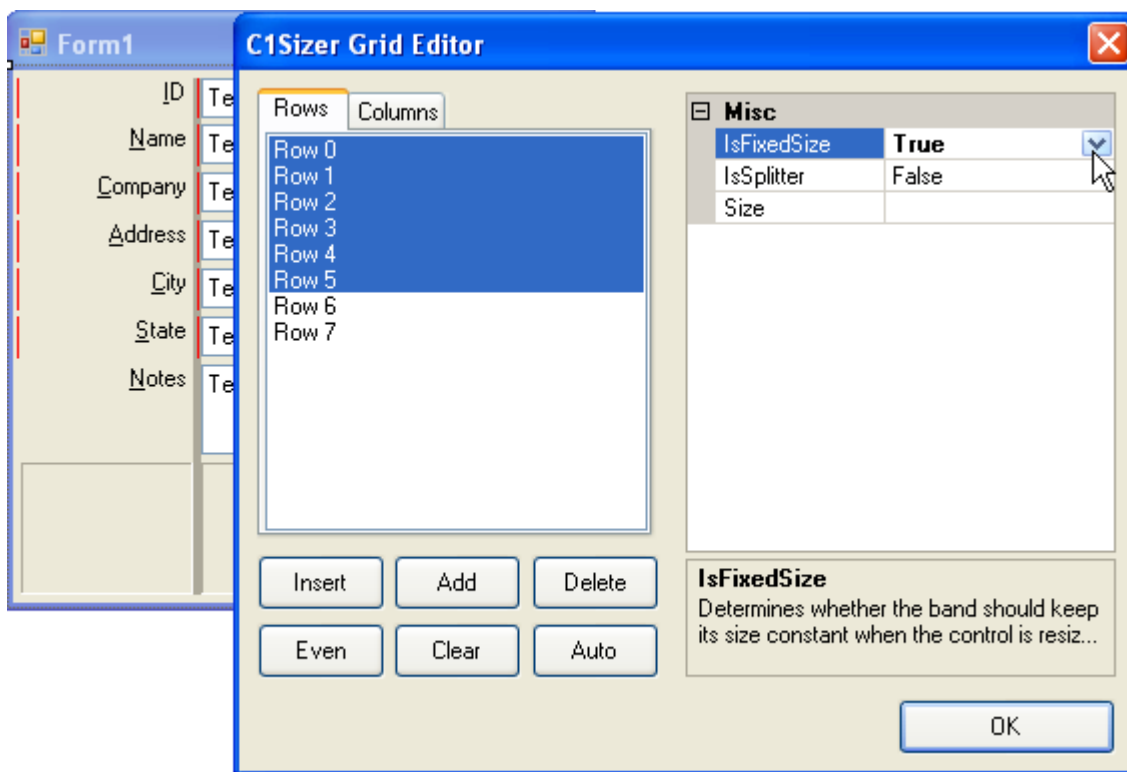
**Note:** A splitter can also be added by right-clicking the band and selecting **Splitter** from the context menu.

Notice the dark gray bar to the right of the first column. This indicates that the user will be able to drag that splitter at run time, automatically resizing the labels and text boxes on the first two columns of the grid.

### Setting up a fixed-size band

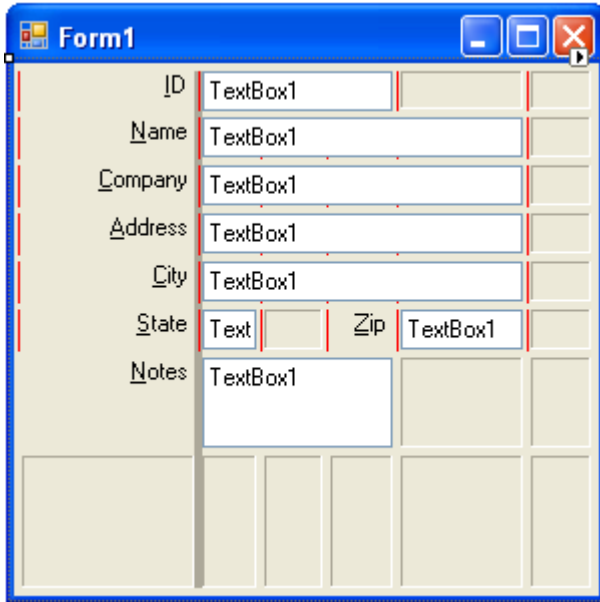
When the control is resized, the grid layout is adjusted to keep the proportion between the bands. You can designate certain bands as fixed size, and they will not be resized.

For example, we might want to allow only the bottom text box to be resized vertically, and keep the height of all others constant. To do this, bring up the grid editor, select the first six rows and set the `Band.IsFixedSize` property to **True**.



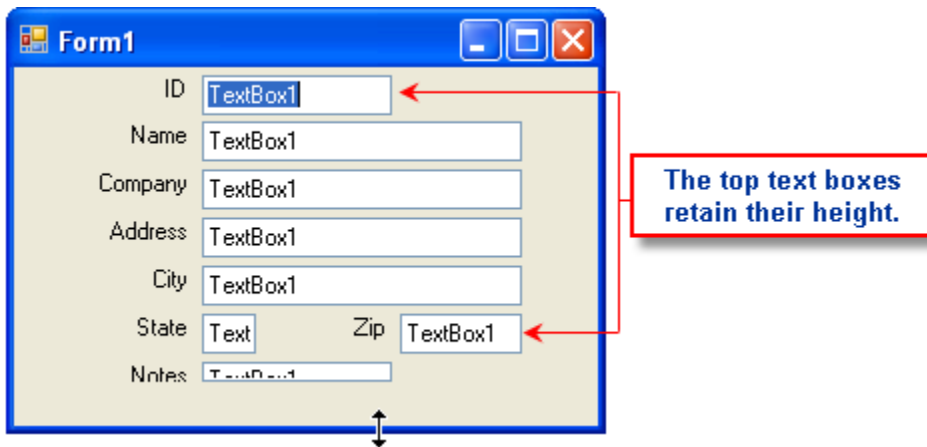
**Note:** A fixed-sized band can also be configured by right-clicking the band and selecting **Fixed Size** from the context menu.

Click the **OK** button. The **C1Sizer** appears with red markers on the first six rows. This indicates that those rows will not be resized with the form (the last two will).



**Run the program and observe the following:**

Use the splitter to adjust the size of the labels and text boxes, then resize the form and notice how the top text boxes retain their height.



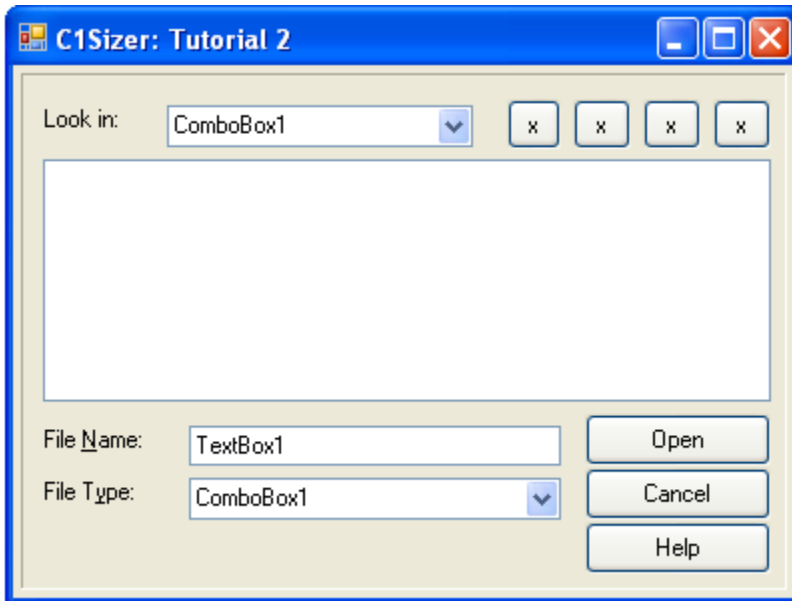
This concludes Tutorial 1. The following tutorial provides more detail about adding controls to the form and setting up the C1Sizer grid.

## Tutorial 2: Add the controls, then set up the grid

In the previous tutorial, we started with the grid and then added controls. You may prefer to work the other way around. Forget about the grid initially, and design your form the normal way. Then, when you are done, let **C1Sizer** create the grid for you automatically. (Of course, once the grid is created you may still modify it with the design time commands we used earlier to tweak the grid's appearance and behavior.)

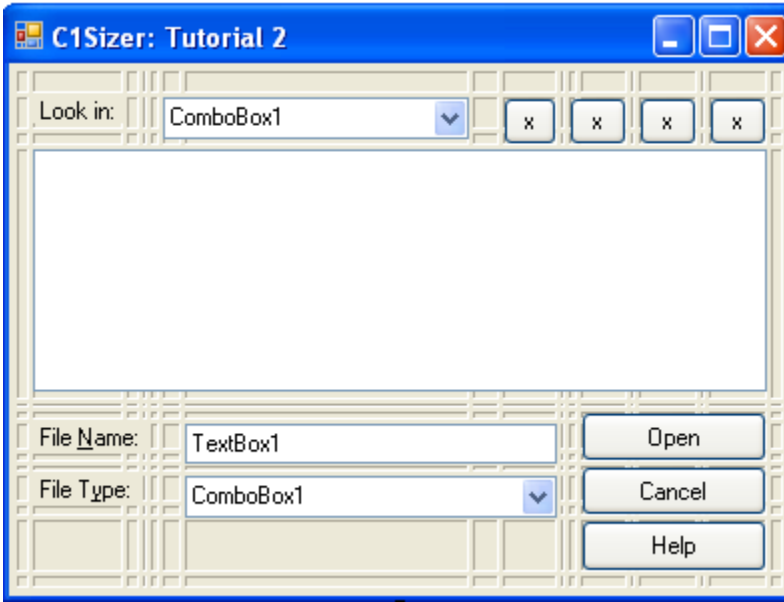
1. Open Visual Studio and create a new project, then add a **C1Sizer** control to the form and set its **Dock** property to **DockStyle.Fill** so it takes up the whole form.
2. Now add the following controls so the form looks like a FileOpen dialog box:

- 3 Labels
- 2 ComboBoxes
- 1 TextBox
- 7 Buttons
- 1 ListBox



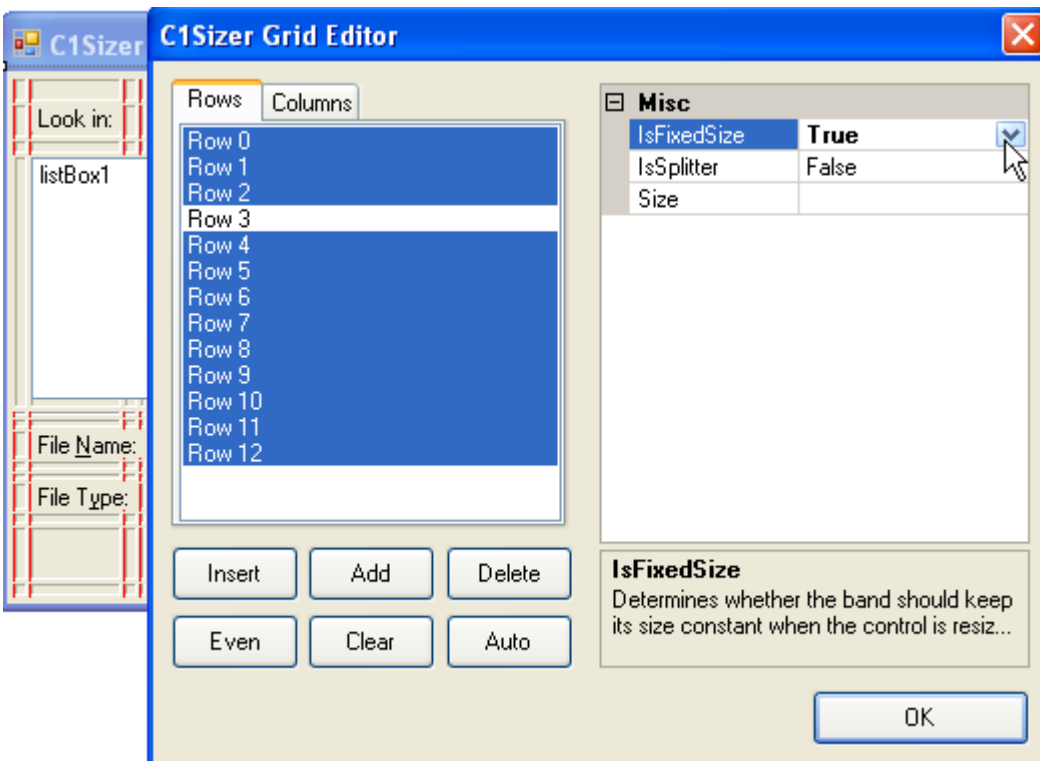
You can use all the layout commands available in Visual Studio to set up this new form. There is nothing new so far.

3. Next, change the `C1Sizer.SplitterWidth` property on the **C1Sizer** control to a small value, one or two pixels. This is recommended because the form has some buttons that are very close together, and we want them to stay that way when we create the grid.
4. To create the grid, right-click the **C1Sizer** control and select the **AutoGrid** option from the context menu. The control will create a grid based on the position of the child controls. The grid will look like this:



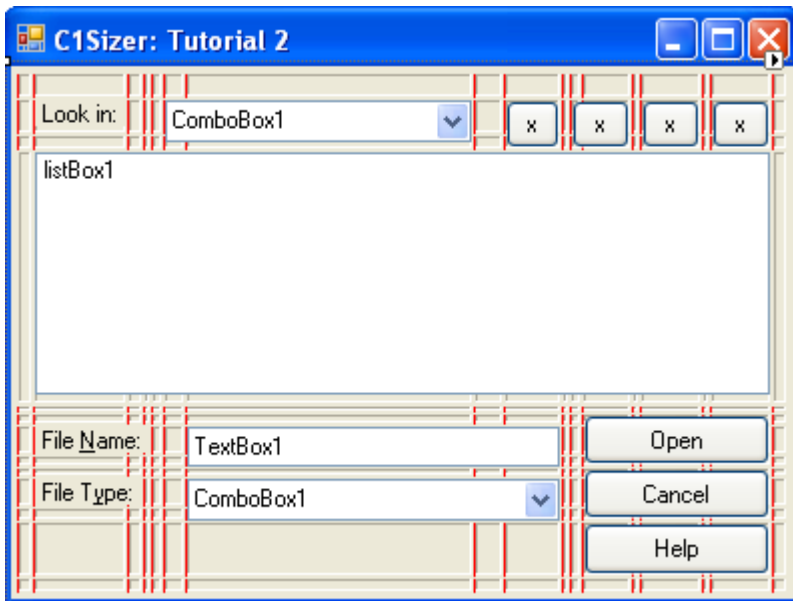
If you run the project now, you will see that when you resize the form, the contained controls are resized proportionally.

- In this case, however, you probably do not want the buttons and text boxes stretched vertically, only the list box. In this case, bring up the grid editor, select all rows except the one that contains the ListBox, and set the `Band.IsFixedSize` property to **True**, as shown below (note the fixed rows are highlighted in red in the designer):



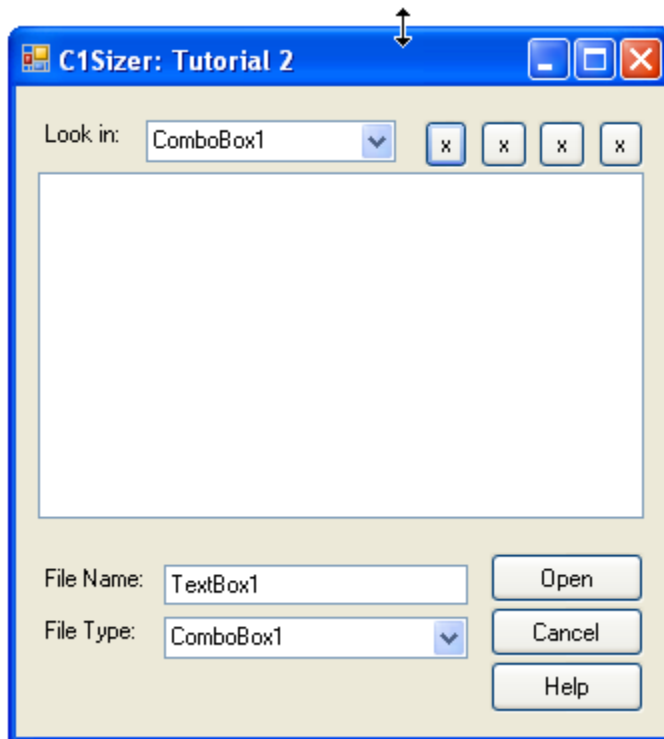
6. Select **OK**.

The designer highlights the fixed rows in red.



**Run the program and observe the following:**

Resize the form to see how the controls snap to the grid and adjust the layout automatically.



# Sizer 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 ComponentOne Studios.

You can access samples from the **ComponentOne Sample Explorer**. To view samples, click the **Start** button and then click **ComponentOne | Studio for WinForms | Samples | Sizer Samples**.

The following examples are included for the **C1Sizer** control:

- **Visual Basic Samples**

**C1Sizer** includes the following Visual Basic samples:

Sample	Description
SizerTutorial1	Demonstrates how to set up the grid and add controls. This sample uses the <b>C1Sizer</b> control.
SizerTutorial2	Demonstrates how to add controls and set up the grid. This sample uses the <b>C1Sizer</b> control.

- **C# Samples**

**C1Sizer** includes the following C# samples:

Sample	Description
AddControls	Shows how to add child controls to a <b>C1Sizer</b> using code. This sample uses the <b>C1Sizer</b> control.
CustomSplitters	Use the <b>OnPaint</b> event to customize the appearance of the splitter bars. This sample uses the <b>C1Sizer</b> control.
FakeOutlook	Create a user interface that looks like OutlookExpress. This sample uses the <b>C1Sizer</b> control.
FakeStudio	Create a user interface that looks like Visual Studio. This sample uses the <b>C1Sizer</b> control.
FindCell	Shows how to determine which cell is at a given point. This sample uses the <b>C1Sizer</b> control.
FindControl	Shows how to determine which control is at a given grid cell. This sample uses the <b>C1Sizer</b> control.
Light_MDI	Use the <b>C1SizerLight</b> with MDI forms. This sample uses the <b>C1SizerLight</b> component.
Light_Nested	Use <b>C1SizerLight</b> in forms with docked and nested controls. This sample uses the <b>C1SizerLight</b> component.
Light_Runtime	Control the <b>C1SizerLight</b> at run time. This sample uses the <b>C1SizerLight</b> component.
Light_Toolbar	Automatically resize controls with a wrapping toolbar. The samples shows how the <b>C1SizerLight</b> component works when the form has a wrapping toolbar docked to the top. This sample uses the <b>C1SizerLight</b> component.
RowHeaders	Demonstrates how to combine the .NET <b>Dock</b> property and panel controls to provide areas with headers. This sample uses the <b>C1Sizer</b> control.
SizerDesigner	Shows how to display the <b>C1Sizer</b> grid and implement the drag-and-drop operations between grid cells. This sample uses the <b>C1Sizer</b> control.





# Sizer for WinForms Task-Based Help

The task-based help section assumes that you are familiar with programming in the Visual Studio .NET environment, and know how to use the C1Sizer and C1SizerLight controls/components in general. If you are a novice to the **C1Sizer for .NET** product, please see [Using the C1Sizer Control](#) (page 27) first.

Each topic provides a solution for specific tasks using the ComponentOne Sizer for .NET product.

Each task-based help topic also assumes that you have created a new .NET project. For additional information on this topic, see [Creating a .NET 2.0 Project](#) (page 10).

## Add a C1SizerLight Component to a Form in code

To programmatically add a C1SizerLight component to a form, use the following code:

- Visual Basic

```
If Me.components Is Nothing Then
    Me.components = New System.ComponentModel.Container
End If
Dim szl As C1SizerLight = New C1SizerLight(components)
szl.SetAutoResize(Me, True)
```

- C#

```
if (this.components == null)
    this.components = new System.ComponentModel.Container();
C1SizerLight szl = new C1SizerLight(components);
szl.SetAutoResize(this, true);
```

To programmatically create a generic form start-up procedure which every form will call, use the following code:

- Visual Basic

```
Dim c As Container = New System.ComponentModel.Container
For Each f As Form In myFormList
    Dim szl As C1SizerLight = New C1SizerLight(c)
    szl.SetAutoResize(f, True)
Next
```

- C#

```
Container c = new System.ComponentModel.Container();
foreach (Form f in myFormList)
{
    C1SizerLight szl = new C1SizerLight(c);
    szl.SetAutoResize(f, true);
}
```

## Position Controls on the C1Sizer Grid at Run Time

To position controls on the C1Sizer grid at run time, simply move the control to the area where you want it to be located. The C1Sizer will snap the control to the nearest position on the grid, and will automatically resize the control when the form is resized. The control can be positioned over a single grid cell or it may span multiple cells.

1. Add the C1Sizer control to your form.

2. Right-click on the C1Sizer control and select **Edit Grid**.
3. In the **C1Sizer Grid Editor** add three rows and three columns and then click **OK**.

To use the **Band.Bounds** property to determine the position of a grid cell, and the **Control.Bounds** property to move the control, use the following code:

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs)
    Dim b1 As Button = New Button
    c1Sizer1.Controls.Add(b1)
    b1.Visible = True
    b1.Bounds = GetCellRectangle(0, 0)
    Dim rc As Rectangle = GetCellRectangle(1, 1)
    rc = Rectangle.Union(rc, GetCellRectangle(2, 2))
    Dim b2 As Button = New Button
    c1Sizer1.Controls.Add(b2)
    b2.Visible = True
    b2.Bounds = rc
End Sub

Private Function GetCellRectangle(ByVal row As Integer, ByVal col As
Integer) As Rectangle
    Return Rectangle.Intersect(c1Sizer1.Grid.Rows(row).Bounds,
c1Sizer1.Grid.Columns(col).Bounds)
End Function
```

- C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Create a button and position it over cell (0,0)
    Button b1 = new Button();
    c1Sizer1.Controls.Add(b1);
    b1.Visible = true;
    b1.Bounds = GetCellRectangle(0,0);

    // Calculate position of range (1,1)-(2,2)
    Rectangle rc = GetCellRectangle(1,1);
    rc = Rectangle.Union(rc, GetCellRectangle(2,2));

    // Create a button and position it over cell range (1,1)-(2,2)
    Button b2 = new Button();
    c1Sizer1.Controls.Add(b2);
    b2.Visible = true;
    b2.Bounds = rc;
}
private Rectangle GetCellRectangle(int row, int col)
{
    return Rectangle.Intersect(
        c1Sizer1.Grid.Rows[row].Bounds,
        c1Sizer1.Grid.Columns[col].Bounds);
}
```

## Create a Three Dimensional Border for the Rows and Columns

The following steps show you how to create a styled border for the rows and columns of the C1Sizer control by using the **C1Sizer.Paint** event to repaint the control's rows and columns and the **DrawBorder3D** method to draw an etched three dimensional border around the rows and columns of the C1Sizer control:

1. Add the C1Sizer control to your form and set its **Dock** property to **DockStyle.Fill** so it takes up the whole form.
2. Add some controls to **C1Sizer's** panel, for example two button controls.
3. Right-click the panel and select **Auto Grid** to create and activate the grid layout.
4. Add the following code to the **C1Sizer1\_Paint** event to repaint its rows and columns with a three dimensional border style:

- Visual Basic

```
Private Sub C1Sizer1_Paint(ByVal sender As Object, ByVal e As
PaintEventArgs)
    For Each row As Band In Me.c1Sizer1.Grid.Rows
        Dim rcrow As Rectangle = row.Bounds
        For Each col As Band In Me.c1Sizer1.Grid.Columns
            Dim rccol As Rectangle = col.Bounds
            Dim rccel As Rectangle = Rectangle.Intersect(rcrow, rccol)
            ControlPaint.DrawBorder3D(e.Graphics, rccel, Border3DStyle.Etched)
        Next
    Next
End Sub
```

- C#

```
private void c1Sizer1_Paint(object sender, PaintEventArgs e)
{
    // Paint sizer grid
    foreach (Band row in this.c1Sizer1.Grid.Rows)
    {
        Rectangle rcrow = row.Bounds;
        foreach (Band col in this.c1Sizer1.Grid.Columns)
        {
            Rectangle rccol = col.Bounds;
            Rectangle rccel = Rectangle.Intersect(rcrow, rccol);
            ControlPaint.DrawBorder3D(e.Graphics, rccel,
                Border3DStyle.Etched);
        }
    }
}
```

## Store Layout Information for the C1Sizer Control

To store layout information for the C1Sizer control, use the GridDefinition property that gets or sets a string containing the grid information. This property is non-browsable, but can be used as any other property.

This example assumes that you have imported the C1.Win.C1Sizer and System.Collections namespaces to your source code. To implement two buttons that save and restore the grid definition and control positions, use the following code:

- Visual Basic

```

Private _gridDefinition As String
Private _ctlPositions As ArrayList

Private Sub btnSaveGrid_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
    _gridDefinition = c1Sizer1.GridDefinition
    _ctlPositions = New ArrayList
    For Each ctl As Control In c1Sizer1.Controls
        _ctlPositions.Add(ctl.Bounds)
    Next
End Sub

Private Sub btnRestoreGrid_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
    If Not (_gridDefinition Is Nothing) Then
        c1Sizer1.GridDefinition = _gridDefinition
        Dim i As Integer = 0
        While i < c1Sizer1.Controls.Count
            c1Sizer1.Controls(i).Bounds = CType(_ctlPositions(i), Rectangle)
            System.Math.Min(System.Threading.Interlocked.Increment(i), i-1)
        End While
    End If
End Sub

```

- **C#**

```

string _gridDefinition;
ArrayList _ctlPositions;

private void btnSaveGrid_Click(object sender, System.EventArgs e)
{
    // Save grid definition
    _gridDefinition = c1Sizer1.GridDefinition;

    // Save control positions
    _ctlPositions = new ArrayList();
    foreach (Control ctl in c1Sizer1.Controls)
    {
        _ctlPositions.Add(ctl.Bounds);
    }
}

private void btnRestoreGrid_Click(object sender, System.EventArgs e)
{
    if (_gridDefinition != null)
    {
        // Restore grid definition
        c1Sizer1.GridDefinition = _gridDefinition;

        // Restore control positions
        for (int i = 0; i < c1Sizer1.Controls.Count; i++)
        {
            c1Sizer1.Controls[i].Bounds = (Rectangle)_ctlPositions[i];
        }
    }
}

```

