
ComponentOne

InputPanel for WinForms

Copyright © 2012 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 InputPanel for WinForms Overview	1
Installing InputPanel for WinForms	1
InputPanel for WinForms Setup Files	1
System Requirements	2
Installing Demonstration Versions	2
Uninstalling InputPanel for WinForms	2
End-User License Agreement	3
Licensing FAQs	3
What is Licensing?	3
How does Licensing Work?	3
Common Scenarios	4
Using licensed components with automated testing products	5
Troubleshooting.....	6
Technical Support.....	7
Redistributable Files	8
About This Documentation	8
Namespaces.....	9
Creating a .NET Project.....	10
Adding the C1InputPanel Component to a Project	10
Key Features	11
InputPanel for WinForms Quick Start	14
Step 1 of 6: Create a Data Source for C1InputPanel	15
Step 2 of 6: Add the First Group of Input Components Using the Editor	16
Step 3 of 6: Add the Second Group of Input Components Using the Editor.....	17
Step 4 of 6: Add the Third Group of Input Components Using the Editor	18
Step 5 of 6: Add the Fourth Group of Input Components Using the Editor	18
Step 6 of 6: Run your Quick Start Application	18
Design-Time Support	19
C1InputPanel Smart Tag.....	19
C1InputPanel Context Menus	21

Check Keyboard Accelerators Designer	22
C1InputPanel Designer	23
ErrorText Editor	25
Office Tab.....	26
Html Tab	27
Properties Tab.....	28
ToolTip Editor	30
Office Tab.....	31
Html Tab.....	32
Properties Tab.....	33
InputPanel Fundamentals	34
InputPanel Keyboard Navigation and Validation.....	34
InputPanel Components.....	35
InputButton	36
InputCheckBox	37
InputRadioButton.....	37
InputComboBox	37
InputDatePicker.....	38
InputTimePicker	39
InputGroupHeader	39
InputImage	40
InputLabel	40
InputHtmlLabel	40
InputMenu	40
InputProgressBar	41
InputTrackBar.....	41
InputMaskedTextBox.....	42
InputNumericBox.....	42
Input TextBox	42
InputDataNavigator	43
InputControlHost	44
InputPanel Components Layout.....	45
InputPanel for WinForms Samples.....	46
InputPanel for WinForms Task-Based Help	49
Adding a Background Image to C1InputPanel	49

Adding Input Components to C1InputPanel.....	50
Arranging C1InputPanel's Input Components	51
Arranging Input Components Using the HorizontalAlign Property	52
Arranging Input Components Using the VerticalAlign Property	52
Arranging Input Components Using Padding.....	53
Arranging the Input Components into Columns	57
Arranging the Input Components into Groups	60
Creating a Spacing Scheme for Input Components	62
Moving Items in a C1InputPanel	64
Removing Input Components from C1InputPanel	65
Creating a Collapsible Input Group	65
Creating Rich ToolTips for C1InputPanel Items	66
Exporting and Importing XML Layout Files	68
Formatting C1InputPanel's Error Frame	70
Using C1InputPanel's InputSeparator.....	73
Working with InputDataNavigator Items	76
Adding Custom Images to InputDataNavigator Items	76
Choosing Items for InputDataNavigator	78
Adding the Reload, Save, and Edit Buttons	80

ComponentOne InputPanel for WinForms Overview

ComponentOne InputPanel for WinForms is the new paradigm to create and maintain data forms. It is a container control that can be placed on a Windows Form to help you create powerful data input applications. You can choose from any combination of the nineteen native data input controls provided or any third party .NET control simply by selecting the Add New Item combobox from InputPanel's SmartTag. InputPanel for WinForms provides automatic binding, layout, and styling. When you bind the C1InputPanel to a data source object at design time, the control is automatically populated with components used to view and edit the data in the data source, including keyboard accelerators.

InputPanel for WinForms provides a keyboard accelerator designer used automatically to add or fix duplicate key accelerators. Make changes to your form without worrying about restructuring the entire form with **InputPanel for WinForms** automatic alignment and tab order. All of the nineteen native input components share a common set of styles, properties, methods, and events which makes programming the **C1InputPanel** easy and consistent. This simple setup paired with low maintenance yields increased productivity. Stop spending hours building data forms when you can generate polished, interactive data forms in minutes with **InputPanel for WinForms**.

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



Getting Started

- [InputPanel for WinForms Quick Start](#) (page 14)
- [Design-Time Support](#) (page 19)
- [InputPanel Fundamentals](#) (page 34)
- [InputPanel Components](#) (page 35)

Installing InputPanel for WinForms

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

InputPanel for WinForms Setup Files

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

- | | |
|---------------------|---|
| bin | Contains copies of all ComponentOne binaries (DLLs, EXEs). |
| C1InputPanel | Contains files related to the InputPanel for WinForms product. |

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 folder:

- | | |
|-------------------|---|
| HelpViewer | Contains Microsoft Help Viewer Visual Studio 2010 integrated documentation for all Studio components. |
|-------------------|---|

Samples

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

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

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

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

Common	Contains support and data files that are used by many of the demo programs.
C1InputPanel	Contains samples and tutorials for InputPanel 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 | InputPanel Samples**.

System Requirements

System requirements include the following:

Operating Systems:	Windows 2000 Windows 2003 Server Windows 2008 Server Windows XP SP2 Windows Vista Windows 7
Web Server:	Microsoft Internet Information Services (IIS) 6.x
Environments:	.NET Framework 2.0 or later C# .NET Visual Basic .NET
Disc Drive:	CD or DVD-ROM drive if installing from CD

Installing Demonstration Versions

If you wish to try **ComponentOne InputPanel 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 InputPanel for WinForms

To uninstall **ComponentOne Studio for WinForms**:

1. Open Control Panel and select the **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.

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 dialog boxes. This is the expected behavior since the test application typically does not contain the necessary licensing information, and there is no easy way to add it.

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

For example:

```
#if AUTOMATED_TESTING
    [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")]
#endif
```

```
#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.
8. Paste the line from step 5 into this file (replace the old licensing information with the new).
9. 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 InputPanel 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.C1InputPanel.2.dll
- C1.Win.C1InputPanel.4.dll
- C1.Win.C1InputPanel.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, 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®](#).

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 general namespace for ComponentOne Web-based products is **C1.Web**. The namespace for the C1InputPanel control is **C1.Win.C1InputPanel**. The following code fragment shows how to declare a C1InputPanel control using the fully qualified name for this class:

- Visual Basic

```
Dim inputpanel As C1.Win.C1InputPanel.C1InputPanel
```

- C#

```
C1.Win.C1InputPanel.C1InputPanel inputpanel;
```

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 a new class named C1InputPanel is created, it can be used inside the project without qualification. However, the C1InputPanel assembly also implements a class called C1InputPanel. So, if the C1InputPanel class will be used in the same project, a fully qualified reference must be used to make the reference unique. If the reference is not unique, Visual Studio .NET produces an error stating that the name is ambiguous.

Fully qualified names are object references that are prefixed with the name of the namespace where the object is defined. Objects defined in other projects can be used if a reference to the class is created (by choosing Add Reference from the Project menu) and then using the fully qualified name for the object in the 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 MyInputPanel = C1.Win.C1InputPanel  
Dim inputpanel As MyInputPanel
```

- C#

```
using C1.Win.C1InputPanel = MyInputPanel;  
MyInputPanel inputpanel;
```

If the **Imports** statement is used without an alias, use all the names in that namespace without qualification provided they are unique to the project.

As a warning, unless specified explicitly in the code, it is taken for granted that the following statement has been specified:

- Visual Basic
`Imports C1.Win.C1InputPanel`
- C#
`using C1.Win.C1InputPanel;`

This applies only to small code samples. Tutorials and other longer samples will specify complete qualifiers.

Creating a .NET Project

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

1. From the **File** menu in Microsoft Visual Studio .NET, select **New Project**. The **New Project** dialog box opens.
2. Under **Project Types**, choose either **Visual Basic** or **Visual C#**. Note that one of these options may be located under **Other Languages**.
3. Select **Windows Application** from the list of **Templates** in the right pane.
4. Enter or browse for a location for your application in the **Location** field and click **OK**.
A new Microsoft Visual Studio .NET project is created in the specified location. In addition, a new Form1 is displayed in the Designer view.
5. Double-click the C1InputPanel component from the Toolbox to add it to Form1. For information on adding a component to the Toolbox, see [Adding the C1InputPanel Component to a Project](#) (page 10).

Adding the C1InputPanel 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 InputPanel for WinForms provides the following control:

- C1InputPanel

To use C1InputPanel, add one of these controls to the form or add a reference to the C1.Win.C1InputPanel.2 or assembly in your project.

Manually Adding C1InputPanel to the Toolbox

When you install C1InputPanel, the following C1InputPanel components will appear in the Visual Studio Toolbox customization dialog box:

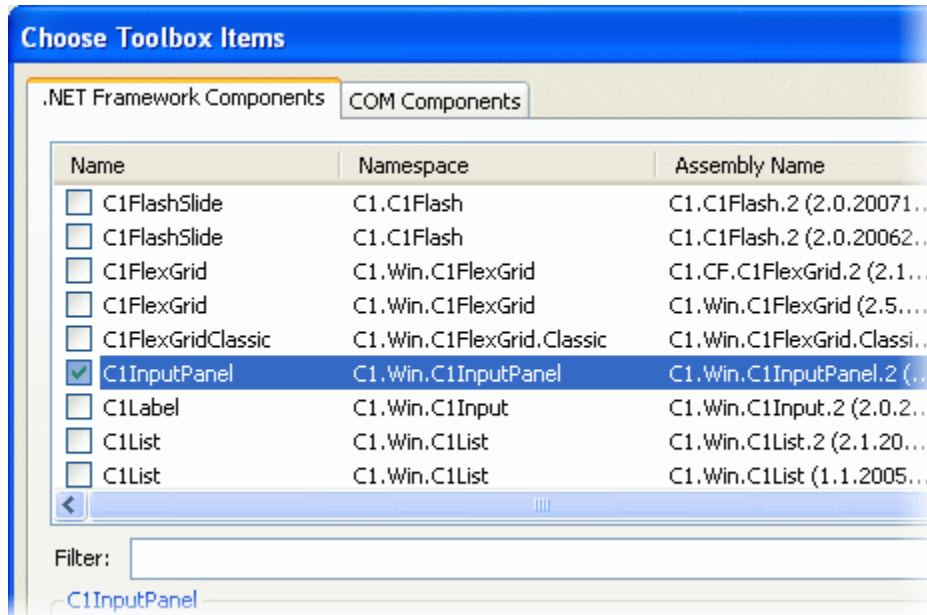
- C1InputPanel

To manually add the **C1InputPanel** control 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 the Toolbox to open the context menu.
2. To make the C1InputPanel component appear on their own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, C1InputPanel, for example.
3. Right-click the tab where the components are to appear, and select **Choose Items** from the context menu.

The **Choose Toolbox Items** dialog box opens.

4. In the **Choose Toolbox Items** dialog box, go to the **.NET Framework Components** tab. Sort the list by Namespace (click the *Namespace* column header) and check the check box for the component belonging to namespace C1.Win.C1InputPanel.



Adding C1InputPanel to the Form

To add **C1InputPanel** to the form:

1. Add the C1InputPanel control to the Visual Studio Toolbox.
2. Double-click the C1InputPanel control or drag it onto your form.

Adding a Reference to the C1InputPanel Assembly

To add a reference to the C1InputPanel assembly:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the most recent version of the **C1InputPanel** assembly from the list on the **.NET** tab or browse to find the C1.Win.C1InputPanel.2.dll file and click **OK**.
3. Select the **Code** menu item from the **View** menu of your project to open the Code Editor. At the top of the file, add the following **Imports** statement (**using** in C#):

```
Imports C1.Win.C1InputPanel
```

Note: This makes objects defined in the C1InputPanel assembly visible to the project. See [Namespaces](#) (page 9) for more information.

Key Features

ComponentOne InputPanel for WinForms offers many rich features such as the following:

- **Easy Setup**

Drop the C1InputPanel component on your form, connect to a data source, select the record set and an input component for each field is automatically created for you. All fields and corresponding labels are consistently aligned for you. A navigation bar is also added to the form for end-users to navigate the record set at run time.

- **Universal .NET Data Binding**

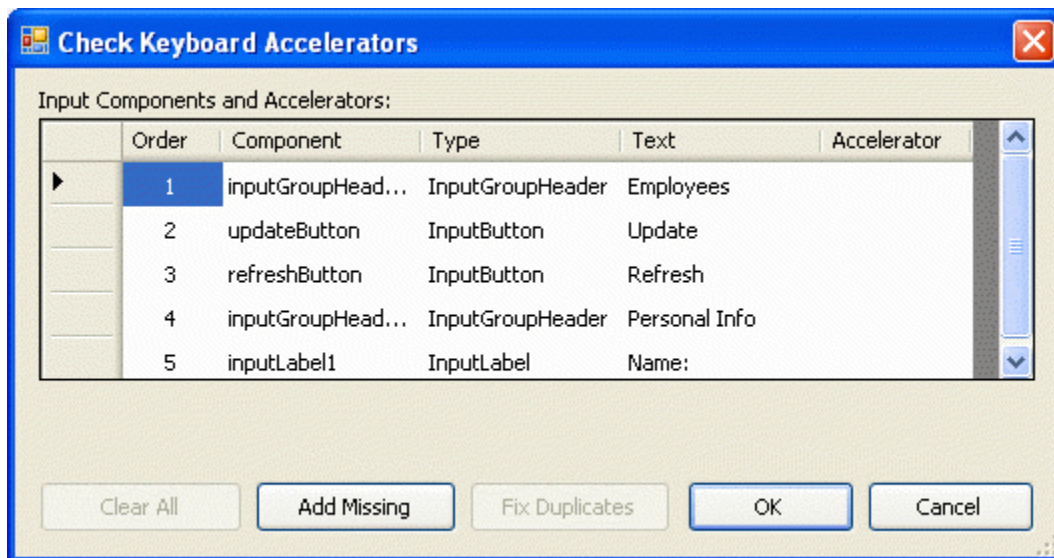
InputPanel for WinForms can universally bind to any .NET data source and with little or no code at all, allowing you to create a fully-navigational database browser in seconds.

- **Automatic Tab order**

Make a change to your data form without worrying about restructuring the entire form – add components and move them around on the form without throwing off the form’s structure. InputPanel for WinForms’ automatic tab order makes this possible. At design time, InputPanel for WinForms automatically establishes the tab order. So at run time, end-users can tab from entry to entry for fast and friendly editing experience.

- **Accelerator Keys**

When the form is populated with fields from a data source, accelerator keys are automatically generated. InputPanel provides a key accelerator editor to manage any newly added accelerators to fix any duplicates.



- **Adjust the Layout with One Property**

You can effortlessly adjust the layout. By default, the data automatically flows from row to row. You can break the flow by setting just one property. For example, you can break the row to a two-column layout. By just modifying one property, the new layout shows two columns and the components remain aligned. Without the flow management functionality provided in InputPanel for WinForms, you would have to move each component one at a time, and then realign the components.

- **Enhance your UI with Microsoft Office 2007 Styling**

InputPanel for WinForms supports Visual Styles that mimic the styles available in Office 2007 including Blue, Silver, and Black.

- **Interactive Components with Graphical Effects**

The appearance of each field at run time is customized when you move the mouse over the component, press it, or disable it. No extra steps are required for these graphical effects.

- **Scrollbars Automatically Added if Needed**

InputPanel for WinForms automatically adds horizontal and vertical scroll bars when the form is resized, if needed. This feature is available at both design time and run time.

- **Validation and Error Handling**

When the end-user enters invalid input, a customizable visual alert is automatically provided. A red frame appears around the input component. You can also add custom alerts such as ToolTips. For example, you can allow your end-users to only enter alpha characters, and show an error indicator when any other character is entered.

- **Manage your Tasks in One Editor**

InputPanel for WinForms enables you to re-arrange the items in the panel, add new items, and format the items. To efficiently manage these tasks, there is an [InputPanel Collection Editor](#) (page 23). The collection editor is easily accessible from the context menu or smart tag.

Designer enables you to select the data source for the panel, create all the input components, and edit individual components (add/remove/set properties).

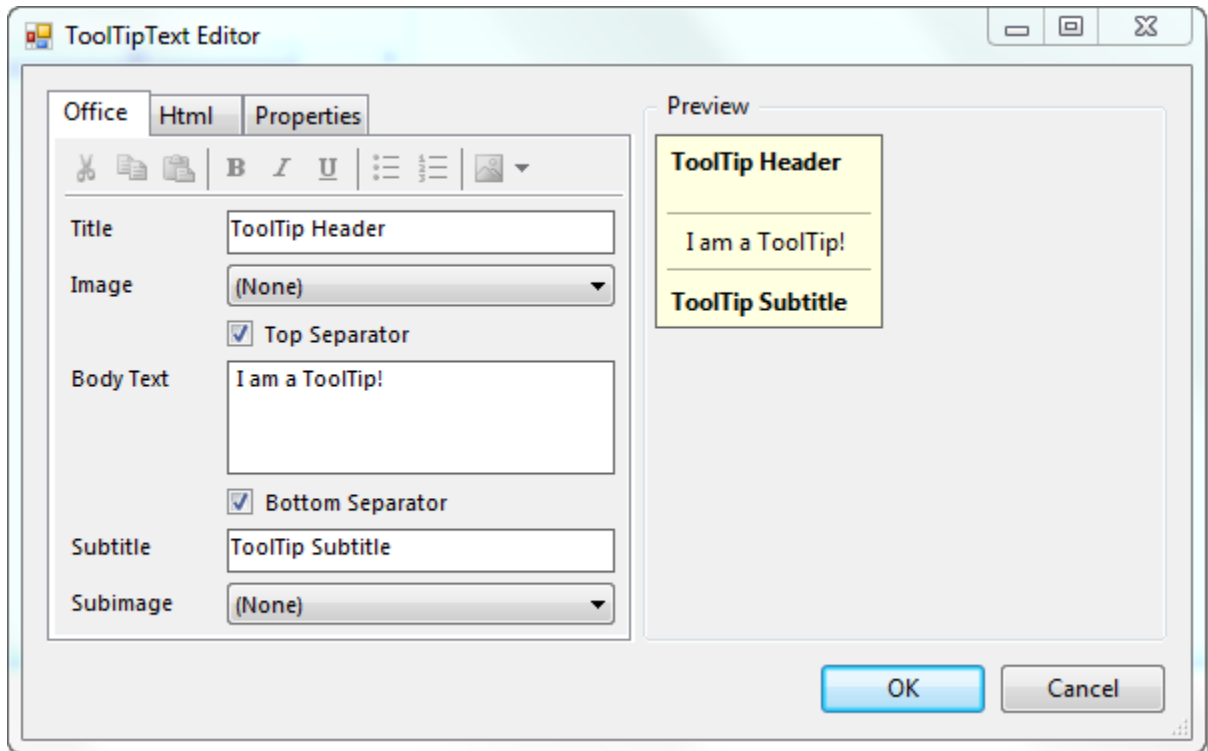
For more information about the designer see, [C#InputPanel Designer](#) (page 23).

- **Spatially Arrange Components on the Form**

To spatially arrange your components on the form, simply adjust the component's padding. You have the freedom to increase and decrease the spacing between components while the components remain neatly aligned. You can also use the HorizontalAlign or VerticalAlign properties for this purpose.

- **Add ToolTips for an Elevated User Experience**

You can easily add graphical elements such as ToolTips at design time. The ToolTipText property has an editor where you can enter your ToolTip text and add images as well as HTML formatting.



- **Simple data binding at design-time**

C1InputPanel control has a DataSource property that can be set at design-time to automatically generate the built-in components used for data-entry.

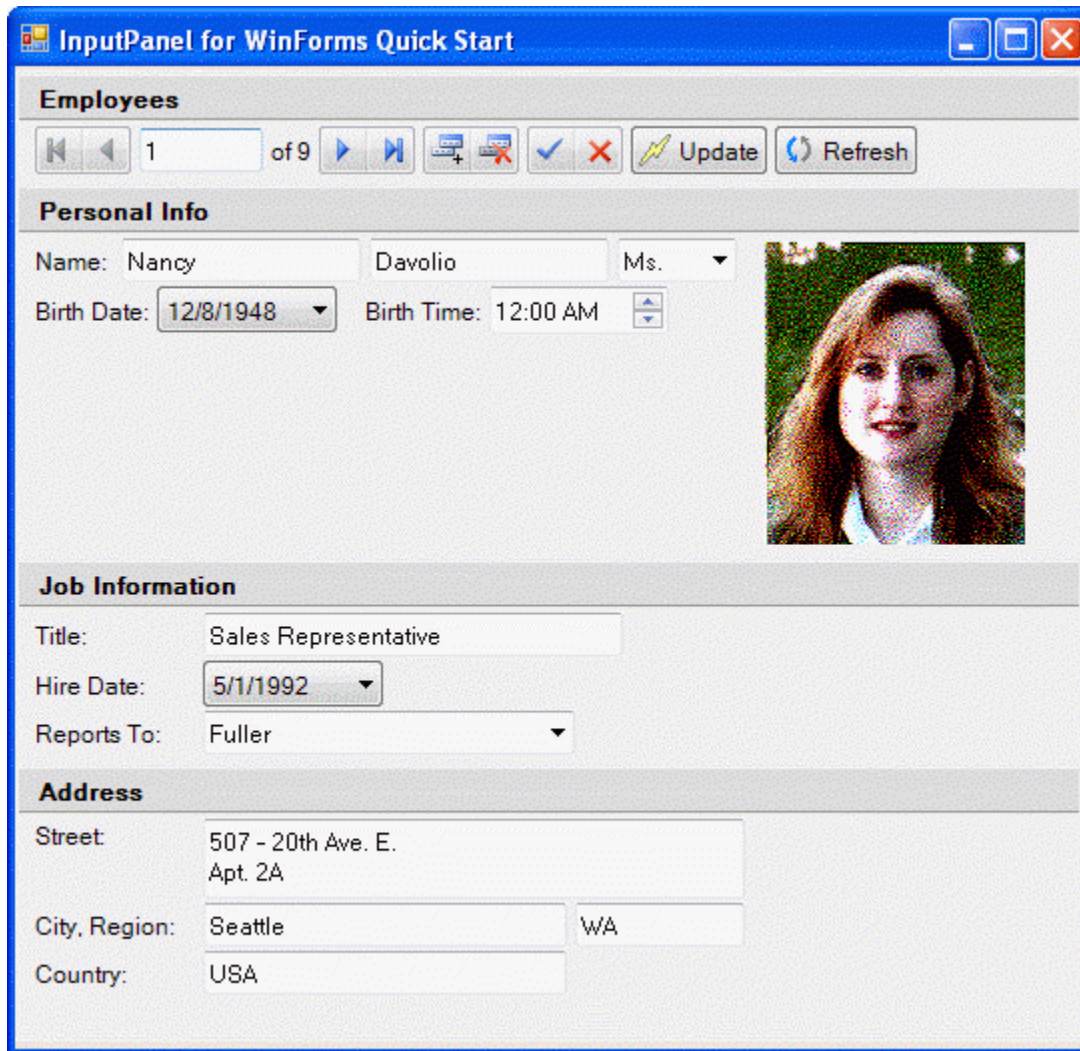
- **Simple, fast, and flexible layout logic to flow individual input components within the panel**

C1InputPanel provides layout logic to flow the individual input components within the panel. The basic mechanism is a flow layout with optional dividers. Within the layout, the panel will control the focus/tab order, etc.

InputPanel for WinForms Quick Start

This section provides step-by-step instructions for building an application that provides input fields for each employee's first and last name, title, title of courtesy, birth data, hire date, address, city, region, postal code, country, home phone, extension, photo, notes and a reports to field. The application uses group headers to divide the employees' information into three categories: Name and birth date of employee, title and job responsibility employee, and address of employee.

Completing this quick start will produce an input application that resembles the following illustration:



Step 1 of 6: Create a Data Source for C1InputPanel

In this step you will create a data source that you can later bind the InputPanel's elements using the **C1InputPanel Item Collection Editor** designer. Create a .NET project (see [Creating a .NET Project](#) (page 10)), and complete the following steps:

Add a new data source

1. In the project toolbar, from the **Data** menu select **Add New Data Source**. The **Data Source Configuration Wizard** dialog box appears.
2. Select **Database** and then click **Next**.
3. Click **New Connection**.
4. In the **Add Connection** dialog box, click **Browse**.
5. In the **Select Microsoft Access Database File** dialog box, locate the **Nwind.mdb** (*In Vista/7: C:\Users\<username>\Documents\ComponentOne Samples\Common, In XP: C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Common*), click **Open**, and then click **OK**.

6. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Since it is not necessary to copy the database to your project, click **No**.
7. Verify the **Yes, save the connection as** check box is checked and click **Next** to continue.
8. Expand the **Tables** node and select the **Employees** object.
9. Click **Finish**.
A DataSet and connection string are added to your project.
10. Expand the **Data** group in the Visual Studio Toolbox and double-click on the BindingSource to add it to your component tray.
11. Select **bindingSource1** from the Properties window, set its **DataSource** property to **nwindDataSet** and its **DataMember** property to **Employees**.

An **employeesTableAdapter** is added to the component tray as well as the following code to the **Form_Load** event:

- Visual Basic

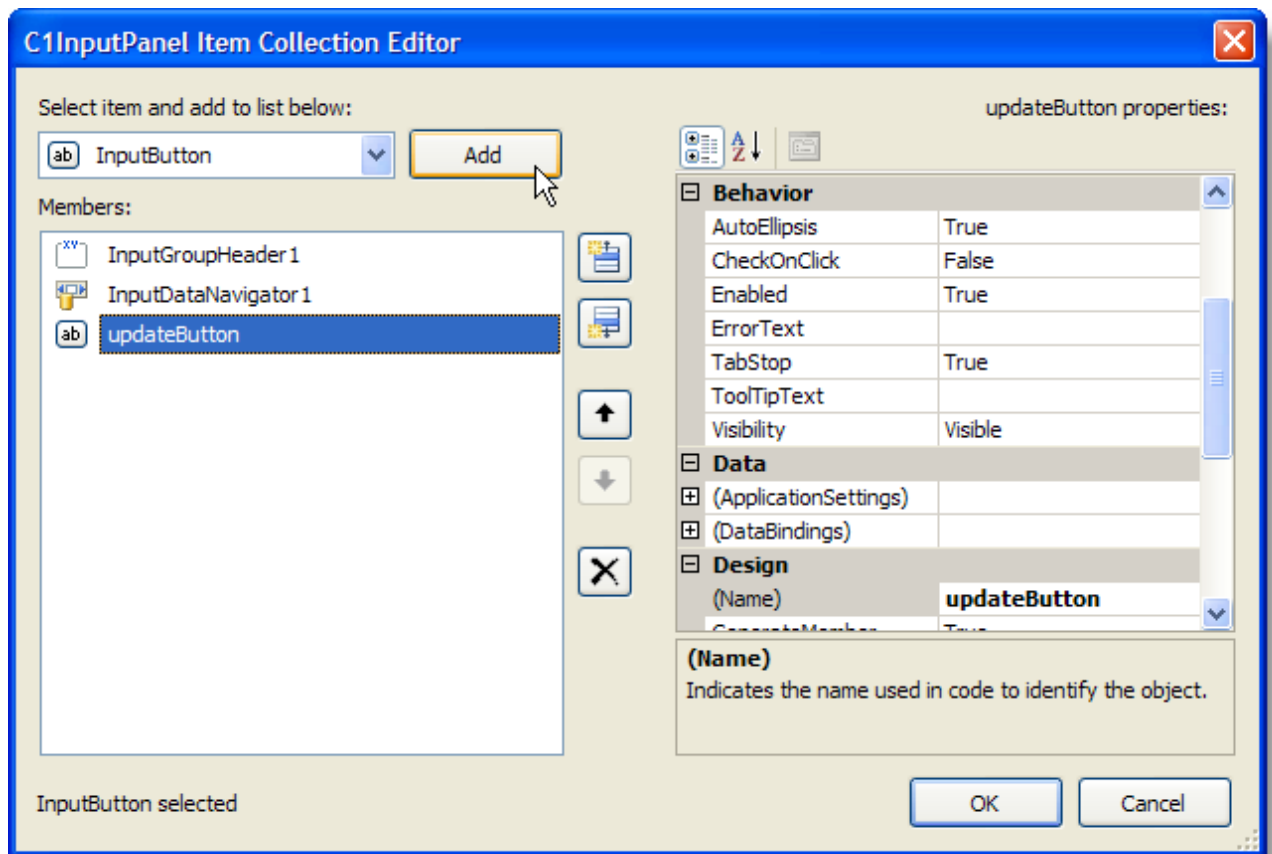
```
Me.EmployeesTableAdapter.Fill(Me.NwindDataSet.Employees)
```
- C#

```
this.EmployeesTableAdapter.Fill(this.NwindDataSet.Employees);
```

Step 2 of 6: Add the First Group of Input Components Using the Editor

To add and format input components using the **C1InputPanel Item Collection Editor**, complete the following tasks:

1. Resize **Form1** to 534x466.
2. Add the C1InputPanel control to the form and set its **Size** property to 534x466 and **SmoothScaling** to True.
For more information on adding the C1InputPanel control to the form, see [Adding the C1InputPanel Component to a Project](#) (page 10).
3. Right-click on the C1InputPanel control and select **Edit Items** from the menu. The **C1InputPanel Item Collection Editor** appears.
4. Click on **Add** to add the [InputGroupHeader](#) component to C1InputPanel and set its **Text** property to "Employees".
5. From the drop-down list box in the **C1InputPanel Item Collection Editor** select **InputDataNavigator** and click **Add**.
6. Set its **DataSource** property to "bindingSource1" and its **InputComponent.Break** property to **None** so the next element that's added will appear after it.



7. Select [InputButton](#) from the drop-down list box and set its **Name** to updateButton, **Break** to None so the next input element will appear next to it, and its **Text** to "Update".
8. Click on the [InputButton](#)'s **Image** property to browse for the image you wish to add to it.
9. Select [InputButton](#) from the drop-down list box and set its **Name** to refreshButton and its **Text** to "Refresh".

Step 3 of 6: Add the Second Group of Input Components Using the Editor

To add the second group of input components to the **InputPanel** using the **C1InputPanel Item Collection Editor**, complete the following:

1. Select [InputGroupHeader](#) from the drop-down list box and set the **Text** property to "Personal Info".
2. Select [InputLabel](#) from the drop-down list box and set the **Text** property to "Name:".
3. Select [InputTextBox](#) from the drop-down list box and click **Add**.
4. Set **InputTextBox1's Break** property to **None** so the next input element appears next to it. Expand its **DataBindings** node and set its **Text** to "bindingSource1-FirstName".
5. Select [InputTextBox](#) from the drop-down list box and click **Add**. Expand its **DataBindings** node and set its **Text** property to bindingSource1-LastName.
6. Select [InputComboBox](#) from the drop-down list box and click **Add**. Expand its **DataBindings** and set its **Text** property to "bindingsource-TitleOfCourtesy".
7. Select [InputLabel](#) from the drop-down list box and click **Add**. Set its **Text** property to "Birth Date:".

8. Select **InputDatePicker** from the drop-down list box and click **Add**. Set its **Break** property to **None** and expand its **DataBindings** node, and then set its **Text** property to "bindingSource1-BirthDate".
9. Select **InputLabel** from the drop-down list box and click **Add**. Set its **Text** property to "Birth Time:".
10. Select **InputTimePicker** from the drop-down list box and click **Add**. Expand its **DataBindings** node and set its **BoundValue** property to "bindingSource1-BirthDate" and its **Break** property to **Column**.
11. Select **InputImage** from the drop-down list box and click **Add**. Expand its **DataBindings** node and set its **Image** property to "bindingSource1-Photo".

Step 4 of 6: Add the Third Group of Input Components Using the Editor

To add the third group of input components to the InputPanel using the **C1InputPanel Item Collection Editor**, complete the following:

1. Select [InputGroupHeader](#) from the drop-down list box and click **Add**. Set its **Text** property to "Job Information".
2. Select [InputLabel](#) from the drop-down list box and click **Add**. Set its **Text** property to "Title:".
3. Select **InputTextBox** from the drop-down list box and click **Add**. Expand its **DataBindings** node and set its **Text** property to "bindingSource1-Title".
4. Select [InputLabel](#) from the drop-down list box and click **Add**. Set its **Text** property to "HireDate:".
5. Select [InputDatePicker](#) from the drop-down list box and click **Add**. Expand its **DataBindings** node and set its **Value** property to "bindingSource1-HireDate". Set its **ValueIsNull** property to **False**.
6. Select [InputLabel](#) from the drop-down list box and click **Add**. Set its **Text** property to "Reports To:".
7. Select [InputComboBox](#) from the drop-down list box and click **Add**. Expand its **DataBindings** node and set its **SelectedValue** property to **bindingSource1-ReportsTo**, **DataSource** property to **BindingSource1**, **DisplayMember** property to **LastName**, **FormattingEnabled** to **True**, and **ValueMember** to **EmployeeID**.

Step 5 of 6: Add the Fourth Group of Input Components Using the Editor

To add the fourth group of input components to C1InputPanel using the **C1InputPanel Item Collection Editor**, complete the following:

1. Select [InputGroupHeader](#) from the drop-down list box and click **Add**. Set its **Text** property to "Address".
2. Select [InputLabel](#) from the drop-down list box and click **Add**. Set its **Text** property to "Street".
3. Select **InputTextBox** from the drop-down list box and click **Add**. Set its **Multiline** property to **True**. Expand its **DataBindings** node and set its **Text** to "bindingSource1-Address".
4. Select [InputLabel](#) from the drop-down list box and click **Add**. Set its **Text** property to "City,Region".
5. Select **InputTextBox** from the drop-down list box and click **Add**. Expand its **DataBindings** node and set its **Text** to "bindingSource1-City". Set its **Break** property to **None**.
6. Select **InputTextBox** from the drop-down list box and click **Add**. Expand its **DataBindings** node and set its **Text** to "bindingSource1-Region".
7. Select [InputLabel](#) from the drop-down list box and click **Add**. Set its **Text** property to "Country:".
8. Select **InputTextBox** from the drop-down list box and click **Add**. Expand its **DataBindings** node and set its **Text** to "bindingSource1-Country".

You have successfully bound all of **C1InputPanel**'s elements to the data source.

Step 6 of 6: Run your Quick Start Application

Press the F5 key to run your application.

To observe the changes, complete the following tasks:

- Click on the Move Next button in the [InputDataNavigator](#) (page 43). All of the input fields update.

Congratulations!

You have successfully created a data source that you used to bind the input components. Additionally, you learned how to use the **Break** property to control the layout for the input components.

Design-Time Support

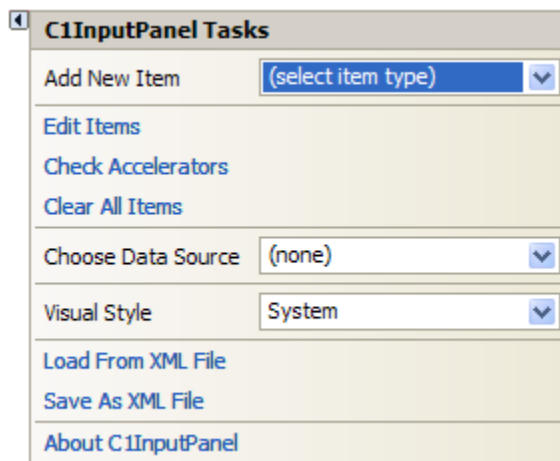
C1InputPanel provides customized context menus, smart tags, and designers that offer rich design-time support and simplify working with the object model.

The following topics describe how to use C1InputPanel's design-time environment to configure C1InputPanel's components.

C1InputPanel Smart Tag

In Visual Studio 2005, each control in **ComponentOne InputPanel for WinForms** includes a smart tag. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in each control.

To access the **C1InputPanel Tasks** menu, click on the smart tag (☰) in the upper right corner of the **C1InputPanel** control. This will open the **C1InputPanel Tasks** menu.



The **C1InputPanel Tasks** menu operates as follows:

Add New Item

Selecting the **Add New Item** drop-down box provides a list of input components to choose from.

Edit Items

Clicking on **Edit Items** opens the **C1InputPanel Items Collection Editor**.

Check Accelerators

Clicking on **Check Accelerators** opens the **Check Keyboard Accelerators** designer.

Clear All Items

Clicking on **Clear All Items** removes the items from C1InputPanel.

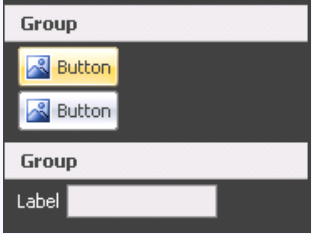
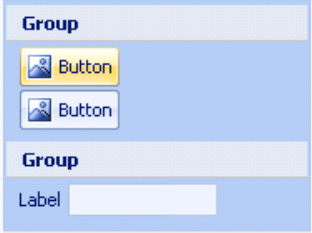
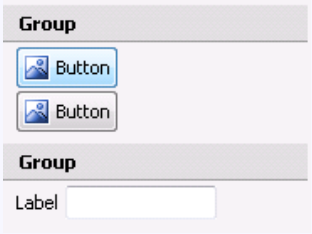
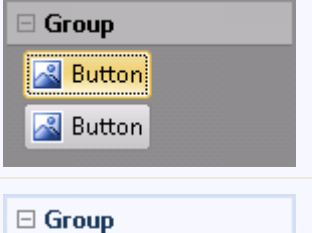
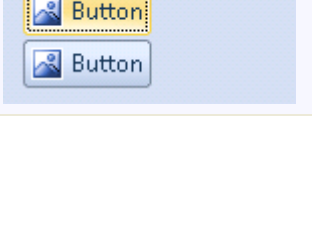
Choose Data Source

Selecting the Choose Data Source drop-down box, opens a list of available data sources and allows you to add a new data source. To add a new data source to the project, click **Add Project Data Source** to open the **Data Source Configuration Wizard**.

Visual Style

Selecting the **Visual Style** drop-down box opens a list box of several built-in inputpanel schemes for you to choose from. The default visual style is System.

C1InputPanel provides the following visual styles:

Visual Style Name	Image
Office2007Black	
Office2007Blue	
Office2007Silver	
Office2010Black	
Office2010Blue	



Load from XML file

Selecting **Load from XML** file option opens the **Load from XML** dialog box, which you can use to navigate to and select an XML file for the InputPanel layout.

Save as XML file

Selecting **Save as XML** file option opens the **Save as XML** dialog box, which you can use to navigate to and select an XML file for the InputPanel layout.

About C1InputPanel

Clicking on the **About** item displays the **About C1InputPanel** dialog box, which is helpful in finding the version number of C1InputPanel and online resources.

C1InputPanel Context Menus

C1InputPanel has designer verbs displayed in the shortcut menu or context menu associated with the C1InputPanel components.

C1InputPanelContext Menu

Right-click anywhere on the C1InputPanel control to display its context menu.

The following table provides a brief description of the custom commands added by C1InputPanel:

Command	Description
About ComponentOne C1InputPanel...	Displays the About ComponentOne C1InputPanel dialog box, which is helpful in finding the version number of C1InputPanel and online resources.
Edit Items	Clicking on Edit Items opens the C1InputPanel Items Collection Editor .
Check Accelerators	Clicking on Check Accelerators opens the Check Keyboard Accelerators designer.
Clear All Items	Clicking on Clear All Items removes the

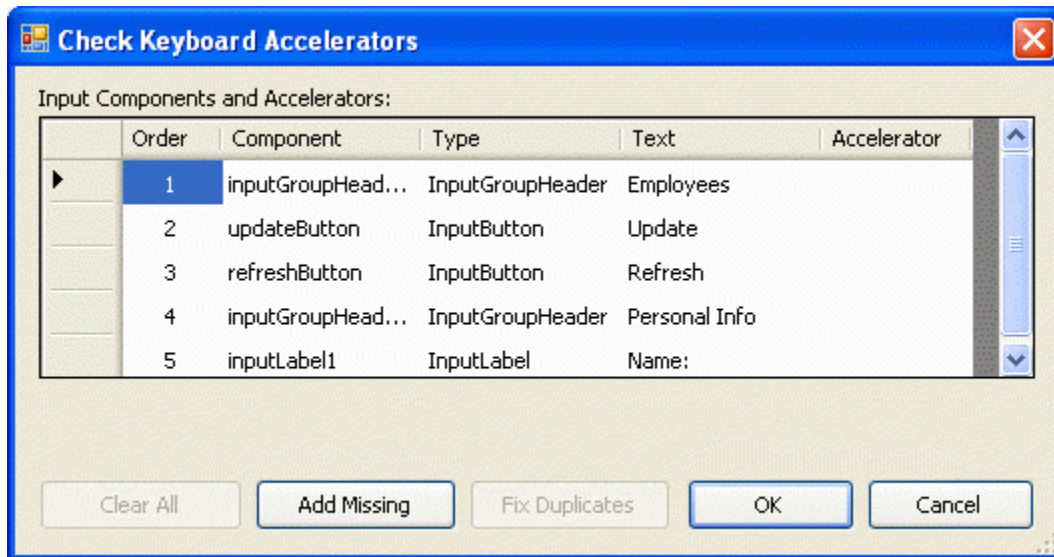
items from C1InputPanel.

Check Keyboard Accelerators Designer

When you add an input component to C1InputPanel, it is automatically added to the **Check Keyboard Accelerator** designer.

To access the Check Keyboard Accelerator

- Right-click on C1InputPanel and select **Check Accelerators**.



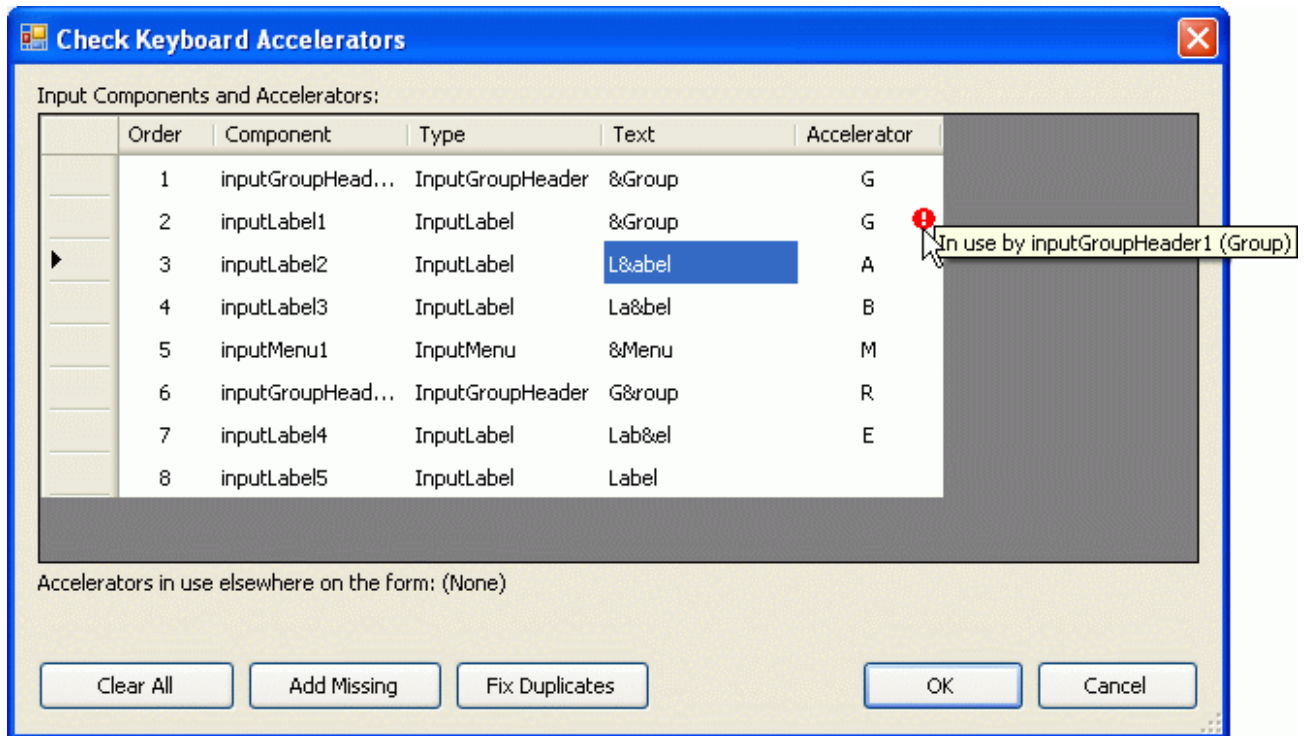
To automatically add keyboard accelerators for all input components

- Click the **Add Missing** button in the **Check Keyboard Accelerators** designer

Typically, the first character for each input component's **Text** property is used as the default key accelerator.

Note: If you have several Input components with all having the same text name, for their text property, then the automatic Add Missing key accelerator method will use a different letter from the Text name for each input component. It will not assign a key accelerator if every character from the Text name has been assigned. You will need to enter a unique key accelerator for it in the accelerator field of the designer.

An error image appears along with a warning message if you entered a duplicate key accelerator.



Clicking on the **Fix Duplicates** button automatically fixes all duplicate key accelerators by assigning unique key accelerators for each one.

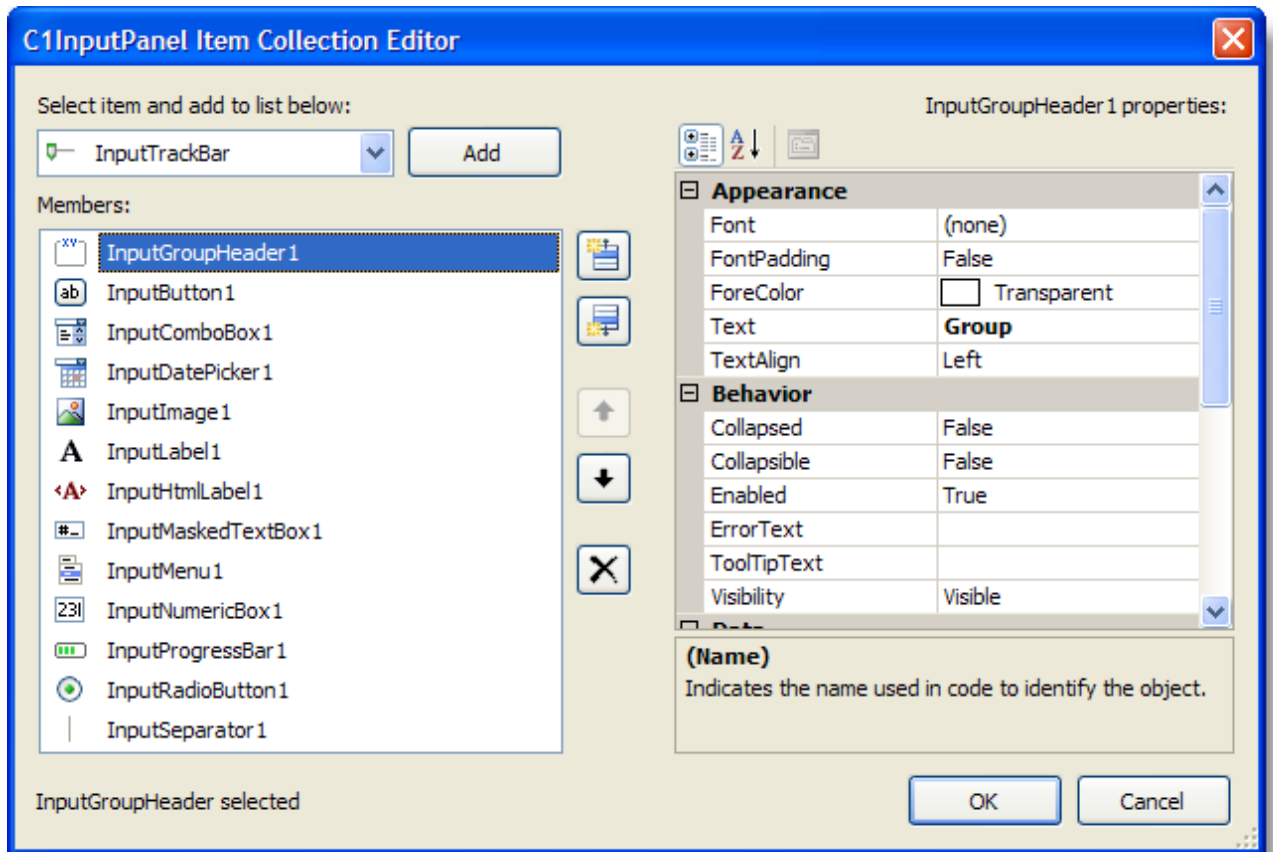
C1InputPanel Designer

C1InputPanel, has a powerful yet simple designer, **C1InputPanel Item Collection Editor**, for adding, removing, or editing input component items on the panel. The main part of the editor's application consists of a windows form, which conveniently allows the user to modify the input components' properties. The editor's command buttons enables the user to quickly add, insert items before or after the selected item, delete, or move the position of the input components just with one click. Each time you add an input component using the designer, the input component automatically appears on the C1InputPanel control on the form. When you add an input component, its properties appear in the group pane of the editor so you can conveniently modify the input component's properties.







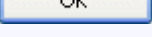
C1InputPanel's **WYSIWYG** (“What you see is what you get”) designer interface makes it simple to see how your elements appear on the InputPanel at design time without having to run the project to see it at run time.

To access the C1InputPanel Item Collection Editor

- Right-click on C1InputPanel and select **Edit Items**.
The **C1InputPanel Item Collection Editor** appears.



The following table provides a description for the functions of all of the command buttons in the **C1InputPanel Item Collection Editor**:


Command Button	Description
	Add the selected item in the combo box after the last input item in the C1InputPanel.
	Inserts the selected input item in the combo box <i>before</i> the selected item in the Members list box.
	Inserts the current input item in the combo box <i>after</i> the selected item in the Members List box.
	Moves the selected input item in the Members list box one position up.
	Moves the selected input item in the Members list box one position down.
	Deletes the selected input item in the Members list box from the C1InputPanel.
	Saves and updates the changes in the C1InputPanel and closes the C1InputPanel Item Collection Editor .


Cancel	Cancels the changes you made and closes the C1InputPanel Item Collection Editor .
--------	--

Adding Input Components



To add an input component, select an item from the drop-down list box and click on the **Add** button.

Adding an Input Component Before or After the Current Input Component

Click on the insert before, , button to insert the input component before the current/selected one.

Click on the insert after, , button to insert the input component after the current/selected one.

Changing Input Component's Position

To change an input component's position within C1InputPanel click the **MoveUp**, , and **MoveDown**, , buttons. The **MoveUp** button moves the selected item up one position in the C1InputPanel and the **MoveDown** button moves the selected item down one position in the C1InputPanel.

Deleting a Specific Input Component

Select the input item from the **Members** list box that you wish to delete and click the **Delete** button.

Editing Input Components

Once an input component is added to the **Members** list box in the editor, you can edit its properties in the property grid.


To edit a specific input component, select the input component from the **Members** list box and edit its properties in the property grid.

ErrorText Editor

The **ErrorText Editor** is similar to the ToolTip editor, only it is used for creating HTML-based message boxes that will appear when users input something other than acceptable data into a C1InputPanel item.

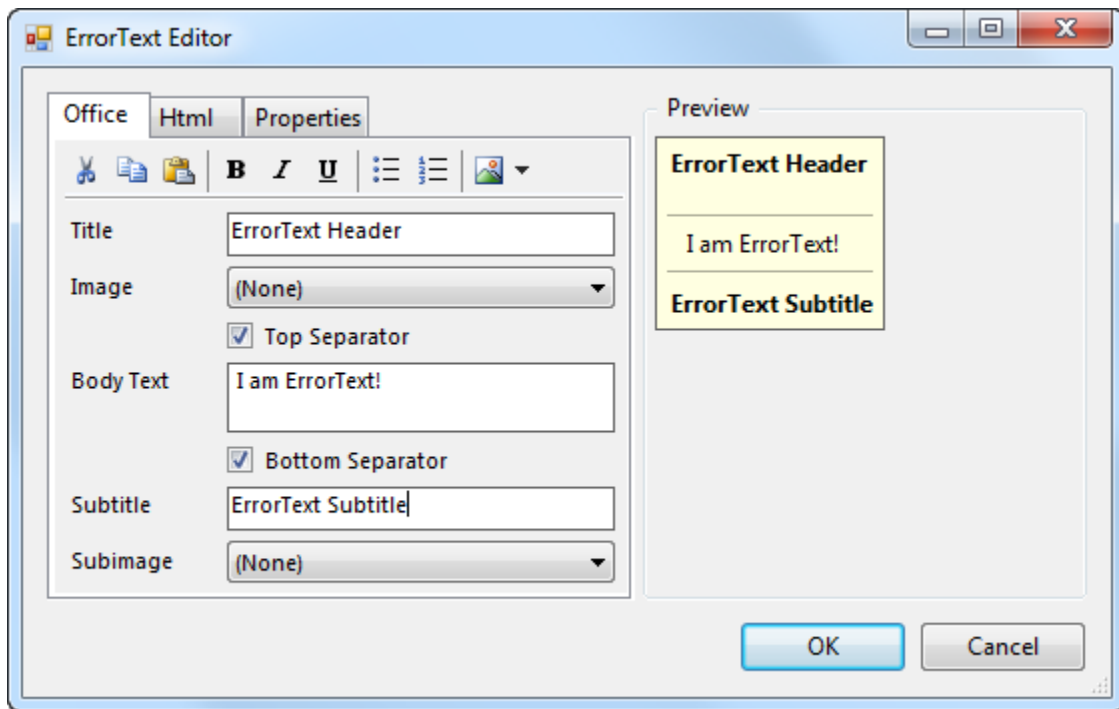
Accessing the ErrorText Editor

To open the **ErrorText Editor**, follow these steps:

1. Click C1InputPanel's smart tag to open the **C1Input Tasks** menu and then select **Edit Items**.
2. In the Members pane, select a component. A list of that component's properties will appear in the Properties grid.
3. In the Properties grid, locate the **ErrorTextText** property and press its ellipsis button . The **ErrorText Editor** appears.

Layout of the ErrorText Editor

When the **ErrorText Editor** opens, you will see three tabs - **Office**, **HTML**, and **Properties** - on the left side of the dialog box. Each tab page features an editor where you can alter the content, source, or properties of the error message. On the right side of the editor is the **Preview** pane, which provides a real-time preview of the final error message.

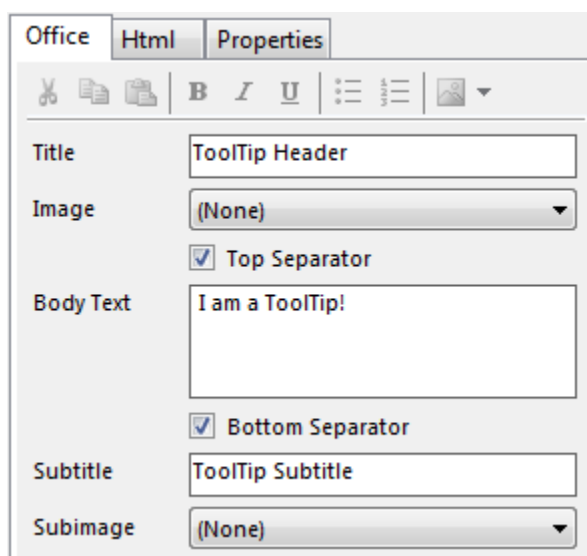


The following topics will detail the **Office**, **HTML**, and **Properties** tab pages.

Office Tab

The **Office** tab page is used to add content, graphics, and separator lines to your Error message. The editor automatically creates all of the HTML code behind the error message, saving you time and effort. Any additions or changes you make to your error message under the **Office** tab will be reflected in the Preview pane.







The **Office** tab page also has a toolbar of command buttons. You can use the command buttons to add HTML elements to your project.



The following table describes the elements under the **Office** tab:

Element	Description
Title	The title element appears at the top of the error message; it is bold by default. You can alter the HTML of this title.
Image	This image will appear over the body text of the error message. By default, it will appear on the left-hand side.
Top Separator	Places a horizontal rule beneath the title and/or image. The horizontal rule is a dark grey.
Body Text	The body text holds the main content of the error message; it is where the description and/or instructions for the tool appear. This will appear beneath the title and above the subtitle.
Bottom Separator	Places a horizontal rule beneath the the body text. The horizontal rule is a dark grey.
Subtitle	The subtitle element appears at the bottom of the error message.
Subimage	This image will appear beneath the body text of the error message. By default, it will appear on the left-hand side.

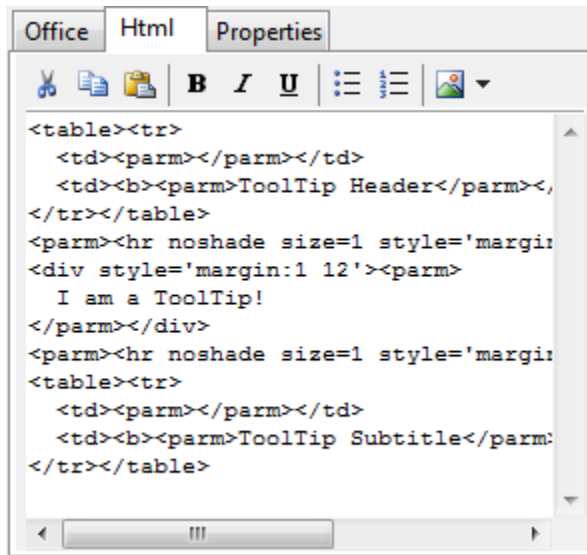
The following table provides descriptions for all of the command buttons under the **Office** tab:

Button	Name	Description
	Cut	Removes selected content from the HTML editor window and places it on the clipboard.
	Copy	Copies selected content from the HTML editor window to the clipboard.
	Paste	Pastes clipboard content to the HTML editor window.
B	Bold	Adds HTML tags for bold text () around selected text.
<i>I</i>	Italics	Adds HTML tags for italicized text (<i></i>) around selected text.
<u>U</u>	Underline	Adds HTML tags for underlined text (<u></u>) around selected text.
	Bulleted List	Adds tags for a bulleted list () to the HTML editor.
	Numbered List	Adds tags for numbered list () to the HTML editor.
	Insert Image	Inserts an image into the tooltip using HTML tags ().







Html Tab

The **Html** tab page is used to add or modify the content of your error message. When creating an error message under the **Html** tab, you have more control over the design of the error message. Any changes you make to your error message under the **Html** tab will be reflected in the Preview pane.

The **Html** tab page consists of a toolbar of command buttons and an HTML editor window. You can use the command buttons to add markup tags to your project, or you can code by hand in the HTML editor window.

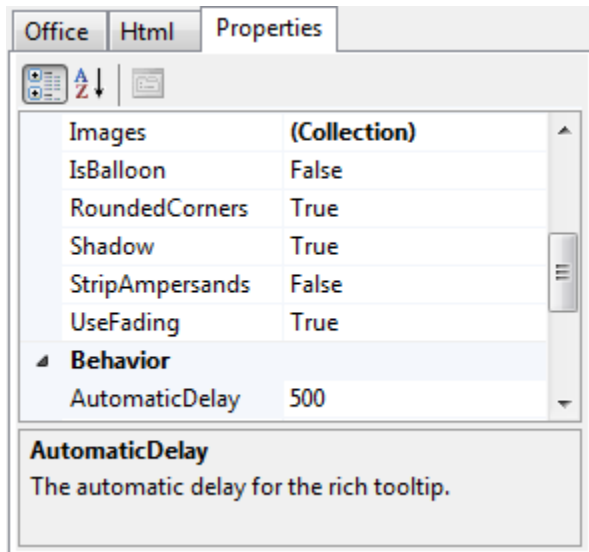


The following table provides descriptions for all of the command buttons under the **Html** tab:

Button	Name	Description
	Cut	Removes selected content from the HTML editor window and places it on the clipboard.
	Copy	Copies selected content from the HTML editor window to the clipboard.
	Paste	Pastes clipboard content to the HTML editor window.
B	Bold	Adds HTML tags for bold text () around selected text.
<i>I</i>	Italics	Adds HTML tags for italicized text (<i></i>) around selected text.
<u>U</u>	Underline	Adds HTML tags for underlined text (<u></u>) around selected text.
	Bulleted List	Adds tags for a bulleted list () to the HTML editor.
	Numbered List	Adds tags for numbered list () to the HTML editor.
	Insert Image	Inserts an image into the tooltip using HTML tags ().

Properties Tab

Within the **Properties** tab page is the Properties grid, which is used to customize the appearance and behavior of your error messages. Many of the changes that you make to the error message properties will be displayed in the Preview pane.



The following table provides a list of properties that affect the appearance of the error message:

Property	Description
BackColor	Sets background color of the error message window.
BackgroundImage	Sets the background image displayed in the error message window.
BackgroundImageLayout	Sets the background image layout to None (default), Tile, Center, Stretch, Zoom, or TileStretch.
Border	Sets whether the error message window should display a solid border.
BorderColor	Sets the color of the error message window border.
Font	Sets the default font used within the error message.
ForeColor	Sets the foreground color of the error message window.
Images	Gets a collection of images that may be shown in the error messages.
IsBalloon	Sets whether or not the error message is shown in a balloon shape. If this is set to True, the error message will resemble a dialogue bubble in a comic strip.
RoundedCorners	Sets whether or not the error message window will have rounded corners.
Shadow	Sets whether or not the error message window will display a drop shadow.
StripAmpersands	Sets whether ampersands in the text should be displayed or hidden.
UseFading	Sets whether a fade effect should be used when displaying a tooltip.

The following table provides a list of properties that affect the behavior of the error message:

Property	Description
Active	Sets a value indicating whether or not the error message is currently active.
AutomaticDelay	Sets the automatic delay for the rich error message.
AutoPopDelay	Sets the period of time (in milliseconds) that the a error message will remain visible if the mouse pointer is stationary.
HitTestVisible	Specifies whether the error message is visible to the mouse.


InitialDelay	Sets the time that passes before the error message
MaximumWidth	Sets the maximum width of the error message.
Opacity	Gets or sets the opacity of the error messages.
ReshowDelay	Sets the length of time that must transpire before subsequent error messages appear as the mouse pointer moves from one component to another.
ShowAlways	Sets a value indicating whether a error message window is displayed even when its parent component is not active.

ToolTip Editor

The **ToolTip Editor** is used to create rich ToolTips for your C1InputPanel components.

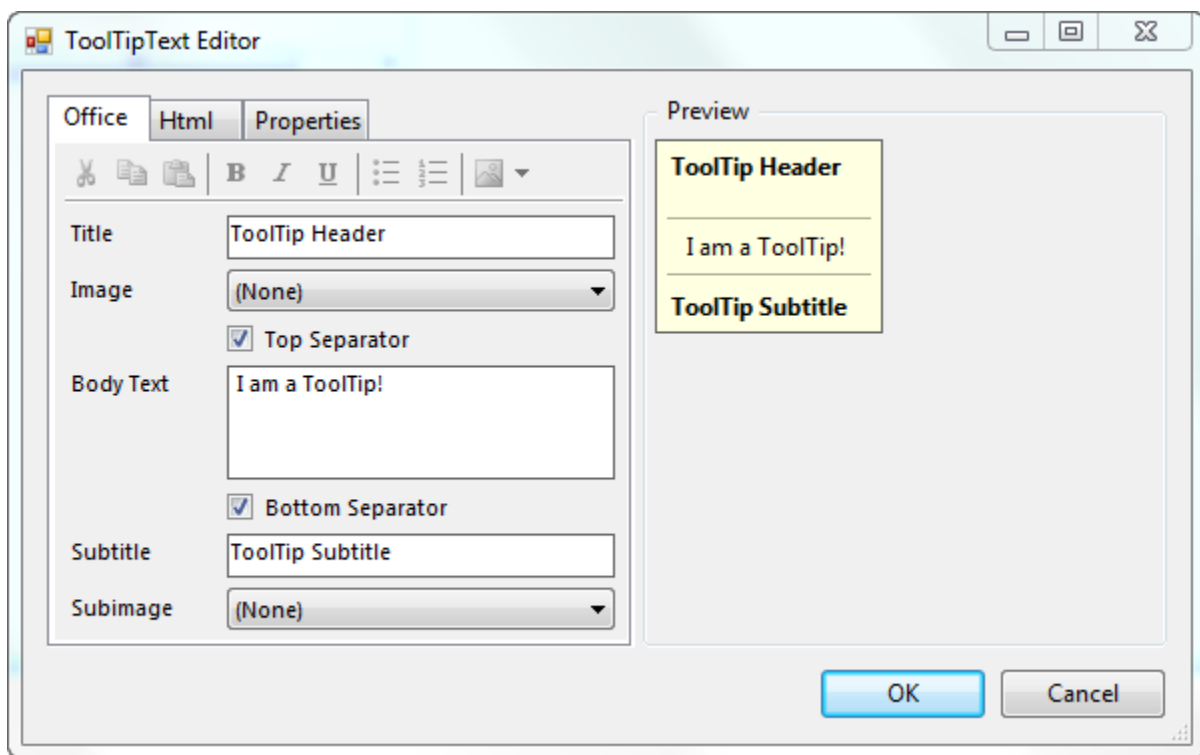
Accessing the ToolTip Editor

To open the **ToolTip Editor**, follow these steps:

1. Click C1InputPanel's smart tag to open the **C1Input Tasks** menu and then select **Edit Items**.
2. In the Members pane, select a component. A list of that component's properties will appear in the Properties grid.
3. In the Properties grid, locate the **ToolTipText** property and press its ellipsis button . The **ToolTip Editor** appears.

Layout of the ToolTip Editor

When the **ToolTip Editor** opens, you will see three tabs - **Office**, **HTML**, and **Properties** - on the left side of the dialog box. Each tab page features an editor where you can alter the content, source, or properties of a ToolTip. On the right side of the editor is the Preview pane, which provides a real-time preview of the final ToolTip product.

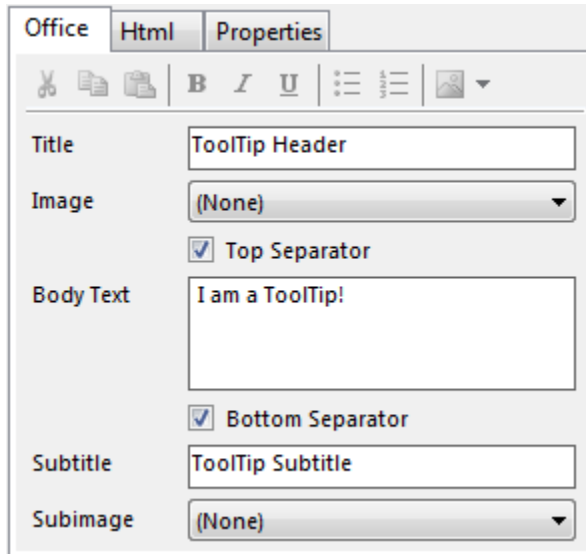


The following topics will detail the **Office**, **HTML**, and **Properties** tab pages.

Office Tab

The **Office** tab page is used to add content, graphics, and separator lines to your ToolTip. The editor automatically creates all of the HTML code behind the ToolTip, saving you time and effort. Any additions or changes you make to your ToolTip under the **Office** tab will be reflected in the **Preview** pane.

The **Office** tab page also has a toolbar of command buttons. You can use the command buttons to add HTML elements to your project.









The following table describes the elements under the **Office** tab:

Element	Description
Title	The title element appears at the top of the ToolTip; it is bold by default. You can alter the HTML of this title.
Image	This image will appear over the body text of the ToolTip. By default, it will appear on the left-hand side.
Top Separator	Places a horizontal rule beneath the title and/or image. The horizontal rule is a dark grey.
Body Text	The body text holds the main content of the ToolTip; it is where the description and/or instructions for the tool appear. This will appear beneath the title and above the subtitle.
Bottom Separator	Places a horizontal rule beneath the the body text. The horizontal rule is a dark grey.
Subtitle	The subtitle element appears at the bottom of the ToolTip.
Subimage	This image will appear beneath the body text of the ToolTip. By default, it will appear on the left-hand side.

The following table provides descriptions for all of the command buttons under the **Office** tab:

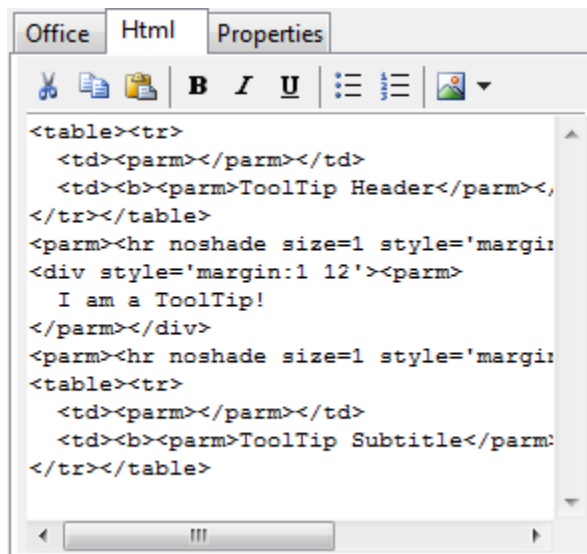
Button	Name	Description
--------	------	-------------

	Cut	Removes selected content from the HTML editor window and places it on the clipboard.
	Copy	Copies selected content from the HTML editor window to the clipboard.
	Paste	Pastes clipboard content to the HTML editor window.
B	Bold	Adds HTML tags for bold text () around selected text.
<i>I</i>	Italics	Adds HTML tags for italicized text (<i></i>) around selected text.
<u>U</u>	Underline	Adds HTML tags for underlined text (<u></u>) around selected text.
	Bulleted List	Adds tags for a bulleted list () to the HTML editor.
	Numbered List	Adds tags for numbered list () to the HTML editor.
	Insert Image	Inserts an image into the tooltip using HTML tags ().

Html Tab







The **Html** tab page is used to add or modify the content of your Tooltip. When creating a Tooltip under the **Html** tab, you have more control over the design of the Tooltip. Any changes you make to your Tooltip under the **Html** tab will be reflected in the **Preview** pane.

The **Html** tab page consists of a toolbar of command buttons and an HTML editor window. You can use the command buttons to add markup tags to your project, or you can code by hand in the HTML editor window.



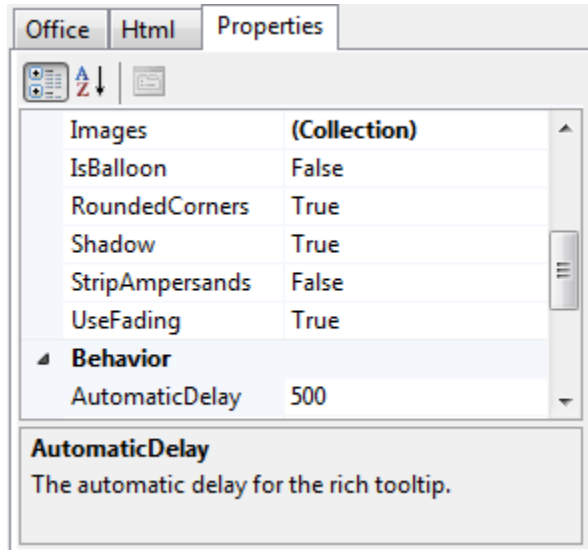
The following table provides descriptions for all of the command buttons under the **Html** tab:

Button	Name	Description
--------	------	-------------

	Cut	Removes selected content from the HTML editor window and places it on the clipboard.
	Copy	Copies selected content from the HTML editor window to the clipboard.
	Paste	Pastes clipboard content to the HTML editor window.
B	Bold	Adds HTML tags for bold text () around selected text.
<i>I</i>	Italics	Adds HTML tags for italicized text (<i></i>) around selected text.
<u>U</u>	Underline	Adds HTML tags for underlined text (<u></u>) around selected text.
	Bulleted List	Adds tags for a bulleted list () to the HTML editor.
	Numbered List	Adds tags for numbered list () to the HTML editor.
	Insert Image	Inserts an image into the tooltip using HTML tags ().

Properties Tab

Within the **Properties** tab page is the Properties grid, which is used to customize the appearance and behavior of your ToolTips. Many of the changes that you make to the Tooltip properties will be displayed in the **Preview** pane.



The following table provides a list of properties that affect the appearance of the Tooltip:

Property	Description
BackColor	Sets background color of the Tooltip window.
BackgroundImage	Sets the background image displayed in the Tooltip window.

BackgroundImageLayout	Sets the background image layout to None (default), Tile, Center, Stretch, Zoom, or TileStretch.
Border	Sets whether the ToolTip window should display a solid border.
BorderColor	Sets the color of the ToolTip window border.
Font	Sets the default font used within the ToolTip.
ForeColor	Sets the foreground color of the ToolTip window.
Images	Gets a collection of images that may be shown in the ToolTips.
IsBalloon	Sets whether or not the ToolTip is shown in a balloon shape. If this is set to True, the ToolTip will resemble a dialogue bubble in a comic strip.
RoundedCorners	Sets whether or not the ToolTip window will have rounded corners.
Shadow	Sets whether or not the ToolTip window will display a drop shadow.
StripAmpersands	Sets whether ampersands in the text should be displayed or hidden.
UseFading	Sets whether a fade effect should be used when displaying a tooltip.

The following table provides a list of properties that affect the behavior of the ToolTip:

Property	Description
Active	Sets a value indicating whether or not the ToolTip is currently active.
AutomaticDelay	Sets the automatic delay for the rich ToolTip.
AutoPopDelay	Sets the period of time (in milliseconds) that the a ToolTip will remain visible if the mouse pointer is stationary.
HitTestVisible	Specifies whether the ToolTip is visible to the mouse.
InitialDelay	Sets the time that passes before the ToolTip
MaximumWidth	Sets the maximum width of the ToolTip.
Opacity	Gets or sets the opacity of the ToolTips.
ReshowDelay	Sets the length of time that must transpire before subsequent ToolTips appear as the mouse pointer moves from one component to another.
ShowAlways	Sets a value indicating whether a ToolTip window is displayed even when its parent component is not active.

InputPanel Fundamentals

The following image shows how the default C1InputPanel appears once it is added to the Windows Form:



InputPanel Keyboard Navigation and Validation

C1InputPanel supports arrow-key navigation and validation.

Keyboard Navigation

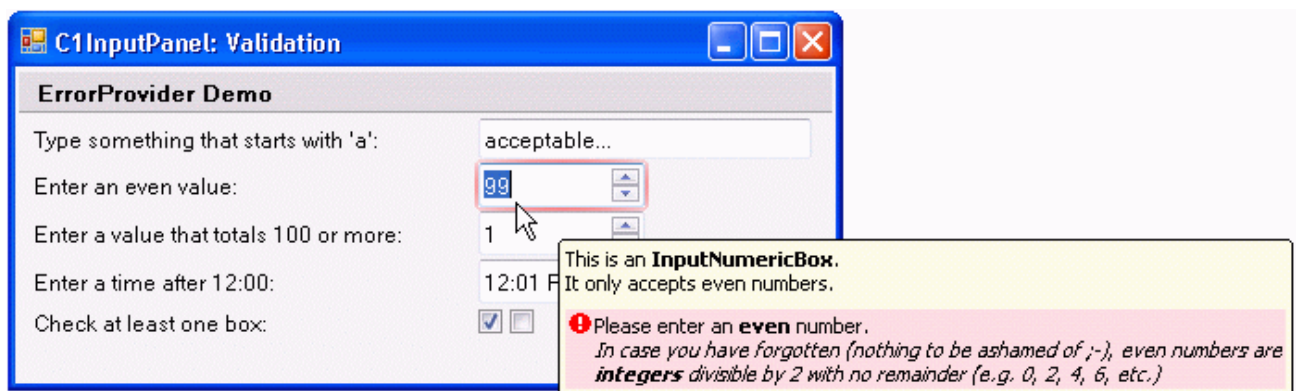
You can navigate through each input component in the C1InputPanel control by pressing the **Tab** key. If an input field contains a custom dialog box you can open the dialog box using the **F4** shortcut key. If the dialog box has several tab forms like a font dialog box does, you can use the TAB key to navigate through each tab form.

Validation

A default red frame automatically appears around an input component to act as a visual alert to let the user know that he or she entered invalid data. The default red frame's appearance can be customized using the C1InputPanel.ErrorFrameColor and C1InputPanel.ErrorFrameThickness properties.

In addition to the visual alert, you can also provide a text alert for invalid input. C1InputPanel has an InputComponent.ErrorText property that displays the string you enter for the property as a ToolTip when you move the mouse over the input component.

The following image shows the visual, and textual validation alerts being used:



InputPanel Components

C1InputPanel has an Items property to provide access to the collection of InputComponent objects. These objects can be added using the designer, **C1InputPanel Item Collection Editor** or programmatically

This section provides a visual and descriptive overview of the elements that comprise the C1InputPanel control. Each topic contains a brief overview about the element and how it is used in the C1InputPanel control.

Each input component has the following main properties:

- Text
- Enabled
- AutoSize
- Width
- Height
- TabStop
- ToolTipText

The following table lists the components and describes their functions:

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

InputGroupHeader	Group header in a C1InputPanel control.
InputButton	Represents a button element in a C1InputPanel control.
InputCheckBox	Represents a check box element in a C1InputPanel control.
InputComboBox	Represents a combo box element in a C1InputPanel control.
InputDataNavigator	Represents a data navigator element in a C1InputPanel control.
InputDatePicker	Represents a date-picker element in a C1InputPanel control. Allows the user to select a date.
InputImage	Image in a C1InputPanel control.
InputLabel	Label in a C1InputPanel control.
InputHtmlLabel	HTML Label in a C1InputPanel control.
InputMaskedTextBox	Represents a masked text box element in a C1InputPanel control.
InputMenu	Represents a menu element in a C1InputPanel control.
InputNumericBox	Represents a numeric box element in a C1InputPanel control.
InputProgressBar	Represents a progress bar element in a C1InputPanel control.
InputRadioButton	Represents a radio button element in a C1InputPanel control.
InputSeparator	Represents a line used to separate items in a C1InputPanel control.
InputSplitButton	Represents a split button element in a C1InputPanel control.
InputTextBox	Represents a text box element in a C1InputPanel control.
InputTimePicker	Represents a time picker element in a C1InputPanel control.
InputTrackBar	Represents a slider element in a C1InputPanel control.
InputControlHost	Represents a control host in a C1InputPanel control.

InputButton

The [InputButton](#) component, like the standard button type control, enables the user to click it to perform an action.



In addition to the standard **Button** properties, [InputButton](#) includes the following unique members:

- `InputComponent.Break` property – Gets or sets how the layout should break the flow after the component.
- `InputButton.CheckOnClick` property – Gets or sets a value that indicates whether the button should toggle its pressed state when it's clicked.
- `InputButton.Pressed` property – Gets a value that indicates whether the button is pressed.
- `InputComponent.ToolTipText` property – Gets or sets the text that appears as a `ToolTip` for this component.
- `InputButton.PressedChanged` event – Fires when the `Pressed` property is changed.

InputCheckBox

You can set the alignment of the checkbox to the left, center, or right of its text.

The [InputCheckBox](#) can have three states:

- Unchecked
- Checked
- Indeterminate

The default, **InputCheckBox** allows two check states, unchecked and checked, but you can allow all three check states by setting the `InputCheckBox.ThreeState` property to **True**.

You can specify whether the checkbox appears left, center, or right aligned using the `InputCheckBox.CheckAlign` property. The following table illustrates how the different settings for the `InputCheckBox.CheckAlign` property appear for the **InputCheckBox** component:

InputCheckBox.CheckAlign Setting	Appearance
Left	<input type="checkbox"/> CheckBox
Center	<input type="checkbox"/> CheckBox
Right	CheckBox <input type="checkbox"/>

InputRadioButton

The `InputRadioButton` is a button that, when checked, unchecks all others in the same input group/separation.

Radio Button

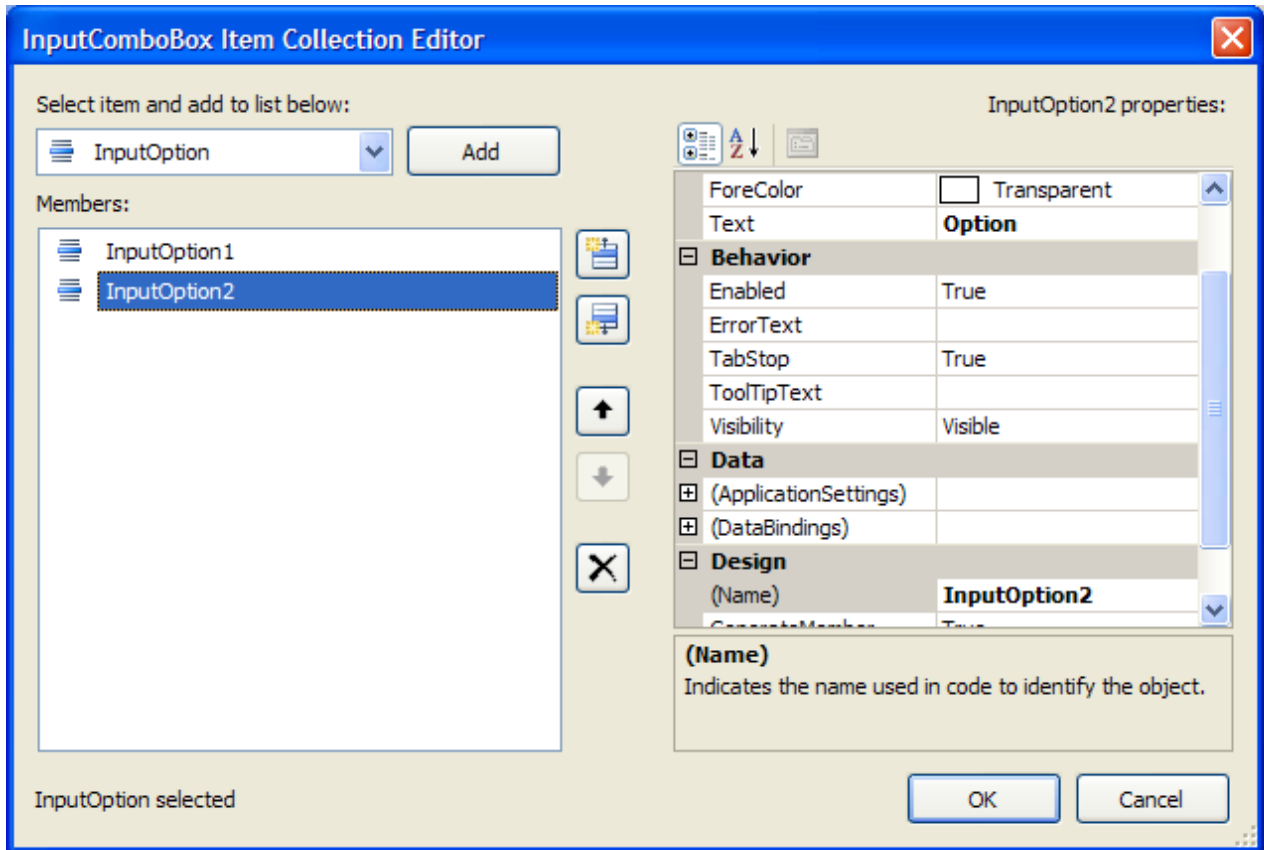
InputComboBox

The default [InputComboBox](#) has a drop-down arrow, but you can change the appearance of the drop-down style by setting its `DropDownStyle` property to a different value. [InputComboBox](#) has a drop-down arrow and a drop-down list.

You can also specify the maximum number of entries to display in the drop-down list using the `InputComboBox.MaxDropDownItems` property. The default is eight entries.

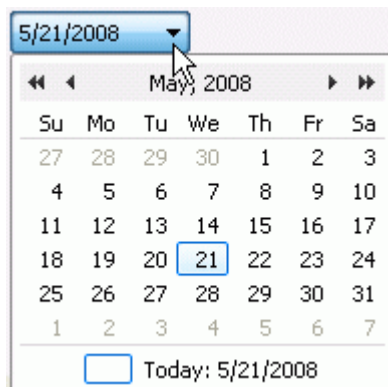
Two types of items can be added to the [InputComboBox](#): [InputOption](#) and [InputGroupHeader](#).

`C1InputPanel` provides an **InputComboBox Item Collection Editor** to add items to the [InputComboBox](#). The items are referred to as `InputOption`.



InputDatePicker

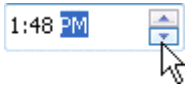
The `InputDatePicker` appears like the following at run time:



By clicking the down arrow key in the box a calendar appears with the current date at the bottom of the calendar. You can choose not to show the current date if you set the `InputDatePicker.ShowToday` to **False**.

InputTimePicker

The `InputTimePicker` component appears like the following at run time:



The [InputTimePicker](#) includes three fields for entering the time. The default [InputTimePicker](#) includes the following three fields: hour, minute, and whether it's AM or PM.

InputGroupHeader

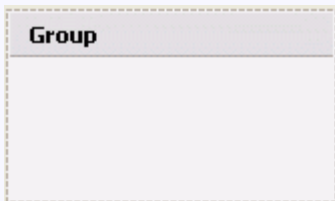
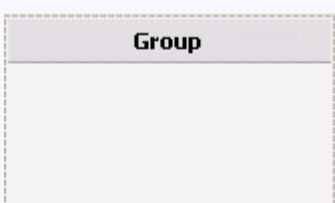
This component nests within the main control, `C1InputPanel`, to provide additional control over the panel's layout. It displays a border around the nested components with a collapsible title bar. The text for the title bar can be set using the `InputGroupHeader.Text` property. The default text, `Group`, appears for the title of the header.

The [InputGroupHeader](#) component can be collapsible when the `InputGroupHeader.Collapsible` property is set to **True**. See [Creating a Collapsible Input Group](#) (page 65) for more information.



If you want the [InputGroupHeader](#) to appear collapsed you can set its `InputGroupHeader.Collapsed` property to `true`.

The [InputGroupHeader](#)'s text can be aligned to the left, center, or right of the group by setting its `InputGroupHeader.TextAlign` property.

TextAlign Property Setting	Appearance
Left	
Center	



InputImage

The [InputImage](#) component is used to display images within C1InputPanel. You can determine how the image is sized to fit within the component by using the `InputImage.SizeMode` property.



InputLabel

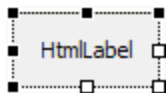
The InputLabel component provides descriptive text to inform users about the purpose of your form.

When you specify the [InputLabel](#)'s width, the label wraps and its height is adjusted to fit the text.



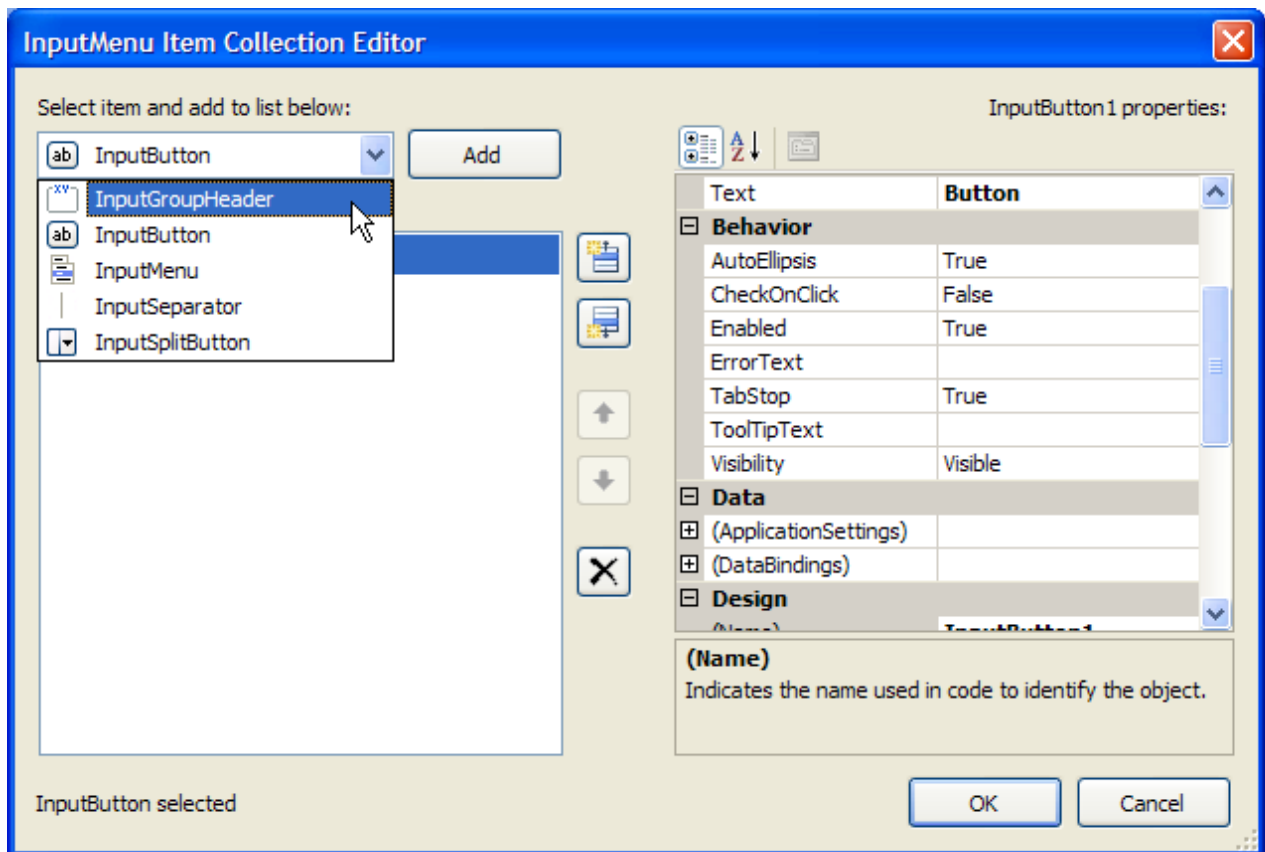
InputHtmlLabel

The InputHtmlLabel is a HTML Label component used to provide descriptive text to inform users about the purpose of your form.



InputMenu

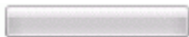
The InputMenu component is a menu. You can easily add menus and menu items through the **InputMenu Item Collection Editor**:



InputProgressBar

The `InputProgressBar` is a progress bar used to show how much of a task has been completed, for example when you download a file.

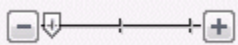
When you add [InputProgressBar](#) to `C1InputPanel`, it appears similar to the following:



InputTrackBar

The `InputTrackBar` represents a slider element in the `C1InputPanel` control.

When you add the [InputTrackBar](#) component to `C1InputPanel`, it appears similar to the following:



At run time you can move the down arrow to the left to decrease the frequency or to the right to increase the frequency of the object it is bound to. You can change the tick frequency by setting the `InputTrackBar.TickFrequency` property.

InputMaskedTextBox

The [InputMaskedTextBox](#) component is used to mask the input text. Each character position in the component maps to either a special placeholder or a literal character. Literal characters, or literals, can give visual cues about the type of data being used. For example, the parenthesis surrounding the area code of a telephone number and dashes are literals: (412)-123-2222. The edit mask prevents the user from entering invalid characters into the component and provides other enhancements to the user interface.

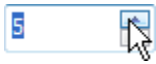


The [InputMaskedTextBox](#) is similar to the standard **MaskedTextBox** and is almost identical to **ComponentOne's WebInput for ASP.NET C1WebMaskEdit** control except for the following unique members:

- InputComponent.Break
- InputComponent.BoundValueChanged

InputNumericBox

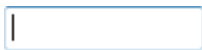
The InputNumericBox component appears similar to the following at run time:



The [InputNumericBox](#) includes one field for entering the number and two up and down arrows to the right of it. Clicking on the up arrow key or down arrow key once increases/decreases the current number by one value unless you specify a value other than 1 for the InputNumericBox.Increment property. You can customize its appearance by specifying the Font and ForeColor of the number. You can also align the number to the left, center, or right of the box by setting its InputTextBox.TextAlign property.

Input TextBox

The InputTextBox component appears similar to the following at run time:



The InputTextBox component is used to enter text. You can modify its behavior by changing any of the following behavior properties:

- InputTextBox.AcceptsReturn – Gets or sets a value that determines whether the InputTextBox should handle the Enter key.
- InputTextBox.AcceptsTab – Gets or sets a value that determines whether the InputTextBox should handle the Tab key.
- InputComponent.Enabled – Gets or sets a value indicating whether the component can respond to user interaction.
- InputComponent.ErrorText – Gets or sets an error description string associated with this component.
- InputTextBox.MaxLength – Gets or sets the maximum number of characters allowing in the component.

- `InputTextBox.Multiline` – Gets or sets a value that determines whether the user can type more than one line of text into the `InputTextBox`.
- `InputComponent.TabStop` – Gets or sets a value indicating whether the user can give the focus to this component using the TAB key.
- `InputComponent.ToolTipText` – Gets or sets the text that appears as a ToolTip for a component.
- `InputComponent.Visibility` – Gets or sets the display date of the component.
- `InputTextBox.WordWrap` – Gets or sets whether the text should be allowed to wrap within the `InputTextBox`.

You can customize the `InputTextBox` component's appearance by changing any of the following appearance properties:

- `InputComponent.Font` – Gets or sets the Font used by the `InputComponent`.
- `InputComponent.FontPadding` – Gets or sets whether the `InputComponent` should use padding to avoid text clipping.
- `InputComponent.ForeColor` – Gets or sets the Color used to display text in the `InputComponent`.
- `InputTextBox.Text` – Gets or sets the text associated with this component.
- `InputTextBox.TextAlign` – Gets or sets how text is aligned in the `InputTextBox`.

When you have text that spans multiple lines you need to set the `InputTextBox.Multiline` property to `True`.

InputDataNavigator

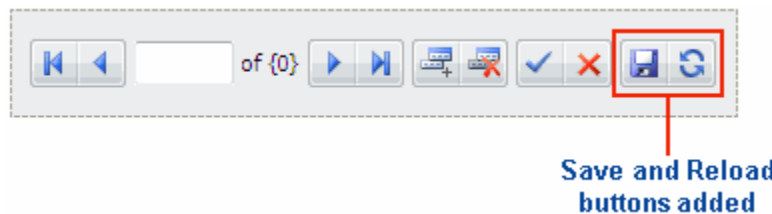
This is a data-bound component which provides buttons for convenient navigation over data source rows. It enables movement to the first, last, previous, and next record and common data actions, such as updating the data source and refreshing data.

When you add the `InputDataNavigator` to `CIInputPanel`, it appears similar to the following:













You can choose which items will be visible on `InputDataNavigator` by setting the `NavigatorItems` property. For more information on how to choose `InputDataNavigator` buttons, see [Choosing Items for InputDataNavigator](#) (page 78).

You can also add a **Save** button and a **Reload** button to the `InputDataNavigator` component (see [Adding the Reload, Save, and Edit Buttons](#) (page 80)). The image below depicts and `InputDataNavigator` component with its Save and Reload buttons added:



The following table provides information about the buttons of the [InputDataNavigator](#) component:

Image	Name	Description
	Move First	Moves the the first record.
	Move Previous	Moves to the previous record.
	Move Next	Moves to the next record.
	Move Last	Moves to the last record.
	Add New	Adds a new record.
	Delete	Deletes the current record.
	Apply Changes	Applies changes to the current record. Once the Apply Changes button is used, changes cannot be cancelled.
	Cancel Changes	Cancels changes made to the current record. Changes <i>cannot</i> be cancelled if the Apply Changes button was already used to apply modifications to the record.
	Save Data	Saves the current data to the external data source.
	Refresh Data	Retrieves updated data from the external data source.

InputControlHost

The InputControlHost makes it possible to embed arbitrary controls into the C1InputPanel. Complete the following steps to add the **MonthCalendar** control to the input panel:

1. Add the C1InputPanel control to the form.
2. We have to define the following class:

- Visual Basic

```
Public Class MonthCalendarHost
    Inherits C1.Win.C1InputPanel.InputControlHost

    Public Sub New()
        MyBase.New(New System.Windows.Forms.MonthCalendar())
    End Sub
End Class
```

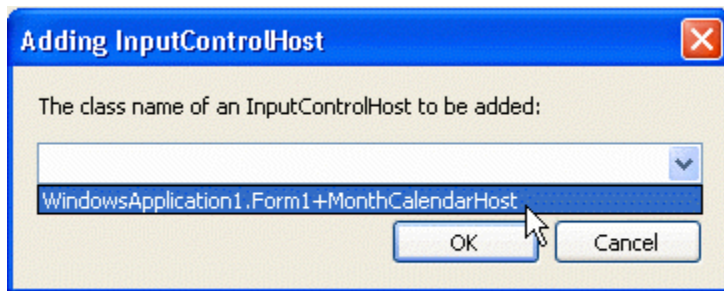
- C#

```
public class MonthCalendarHost : C1.Win.C1InputPanel.InputControlHost
{
    public MonthCalendarHost()
        : base(new System.Windows.Forms.MonthCalendar())
    {
    }
}
```

3. Build the project.
4. In the design view, select the C1InputPanel control and open its smart tag, and then select "InputControlHost" in the "Add New Item" combobox.
This will open a dialog box where you should type the class name of our control host, such as "WindowsApplication1.MonthCalendarHost".
5. Click **OK** in the **Adding InputControlHost** dialog box.

The control is added to C1InputPanel.

Next time, when you want to add the MonthCalendarHost component you don't have to type its name. The name of this class can be selected from the drop-down list in the "Adding InputControlHost" dialog box.



Note: It is highly recommended that you move all of the [InputControlHost](#)-based components to a separate .dll project that will not be rebuilt together with the main application. This allows you to avoid the following well-known issue: In VS2008 IDE, no property of the form or the C1InputPanel containing an [InputControlHost](#) can be changed when the application is debugged once.

InputPanel Components Layout

Input components, by default, are stacked one per line. You can change the layout flow for the input components using the input element's `InputComponent.Break` property to determine how the layout should flow after each input element. The `InputComponent.Break` property has the following four settings to choose from:

- **None** – The layout will not break the flow after the component.
- **Row** – The default setting. Sets each component to be placed in a new row within the flow.
- **Column** – Creates a new column after the component.
- **Group** – Creates a new row starting below the current input component, at the left edge of the current group.

C1InputPanel provides two ways to specify the breaks for the input components:

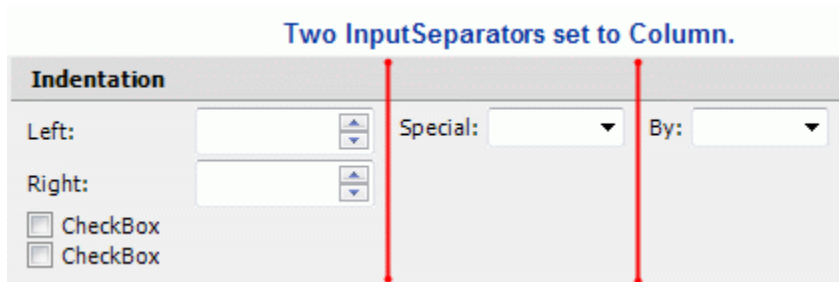
- If the **Break** property in the first component is set to **None**
- Or
- If an **InputSeparator** element exists between the input components

Indent sub-items within the InputGroup component

C1InputPanel also handles the layout for inner elements within the [InputGroupHeader](#) component. The [InputGroupHeader](#) components have a content area with padding, so inner elements are automatically indented.

Multiple Columns

Multiple columns can be created within a group using the InputSeparator component. By default, the **InputSeparator** is invisible when it's added to the C1InputPanel control, but can be made visible to create vertical or horizontal lines to break the controls into lines or columns. The following image illustrates the InputSeparator used to break the Label, Special, and the other Label, By, into columns:



Align elements within a group

Additionally, the [InputLabel](#) component's **Width** property can be used to align elements within a group. When the label width is specified, the label wrap and its height are adjusted to fit the text.



InputPanel 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.

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 | InputPanel Samples**. The following tables provide a short description of each sample.

Visual Basic Samples

Sample	Description
CustomEditors	Demonstrates custom editors used for editing C1InputPanel's visual style.
DataBinding	Demonstrates how to bind each type of input component to the data source using special properties.
ErrorProvider	Demonstrates how to use validating events for the components to validate them. It also shows how to use the ErrorText property to create an error message when the

	user enter incorrect input on the input component.
--	--

C# Samples

Sample	Description
CustomEditors	Demonstrates custom editors used for editing C1InputPanel's visual style.
DataBinding	Demonstrates how to bind each type of input component to the data source using special properties.
ErrorProvider	Demonstrates how to use validating events for the components to validate them. It also shows how to use the ErrorText property to create an error message when the user enter incorrect input on the input component.

InputPanel for WinForms Task-Based Help

The task-based help section assumes that you are familiar with programming in the Visual Studio.NET environment. If you are a novice to C1InputPanel, see the [InputPanel for WinForms Quick Start](#) (page 14) first.


Each task-based help topic provides a solution for specific tasks referencing the **C1.Win.C1InputPanel** namespace. For additional information, please see Adding a Reference to the C1InputPanel Assembly. Each topic also assumes that you have created a new .NET project.

Adding a Background Image to C1InputPanel

You can add a background image to the C1InputPanel control using the designer or code.

Adding the Background Image Using the Designer

To add a background image to the panel, complete these steps:

1. With C1InputPanel selected, navigate to the Properties window.
2. Click the ellipsis button  next to the BackgroundImage property. The **Select Resource** dialog box opens.
3. Select the **Local Resource** radio button and click **Import**.
4. Once the **Open** dialog box appears, select an image and click **Open** to import it into the project.
5. Click **OK** to exit the **Select Resource** dialog box.
6. Navigate to the **BackgroundImageLayout** property and select a layout from its drop-down list. You can select one of the following options:

Layout Setting	Description
None	The image will be placed in the upper left-hand corner of the control and will retain its original size.
Tile (default)	The image will repeat horizontally and vertically to fill the entire control.
Center	The image will be placed in the center of the control and will retain its original size.
Stretch	The image will stretch to fill the entire control.
Zoom	The image will stretch as large as possible while maintaining its original aspect ratio.

Your **C1InputPanel** now has a background image that is laid out to your specifications.

Adding the Background Image Programmatically

Complete the following steps to add a background image and sets its layout to tile:

1. Import the following namespace:
 - Visual Basic
`Imports C1.Win.C1InputPanel`

- C#

```
using C1.Win.C1InputPanel;
```
2. Add the following code, which sets the `BackgroundImage` property to the path of your background image , to the **Form_Load** event:
 - Visual Basic

```
c1InputPanel1.BackgroundImage = Image.FromFile("C:\Backgrounds\YourCustomImage.gif")
```
 - C#

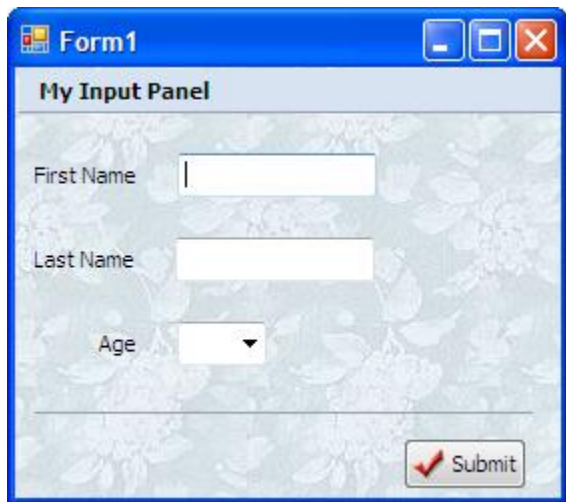
```
c1InputPanel1.BackgroundImage = Image.FromFile(@"C:\Backgrounds\YourCustomImage.gif");
```
 3. Add the following code, which sets the `BackgroundImageLayout` property, to the **Form_Load** event:
 - Visual Basic

```
c1InputPanel1.BackgroundImageLayout = ImageLayout.Tile
```
 - C#

```
c1InputPanel1.BackgroundImageLayout = ImageLayout.Tile;
```
 4. Run the program.

✔ **This Topic Illustrates the Following:**

The image below features a `C1InputPanel` control with a tiled background image:



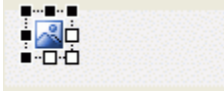
Adding Input Components to `C1InputPanel`

You can add input components to the `C1InputPanel` control using the designer or code.

Adding Input Components Using the Designer

To add an input component to the **`C1InputPanel`** using the designer:

1. Select the **`C1InputPanel`** control and click the smart tag (🔗) to open the **`C1InputPanel Tasks`** menu.
2. Click the **Add New Item** drop-down arrow and select an input component from the list of items. For this example, we will select an **`InputImage`** component.
3. The **`InputImage`** component automatically appears inside the **`C1InputPanel`** control.



Adding Input Components Using the Designer Programmatically

To add a **C1Input** component item to the **C1InputPanel** programmatically:

1. From the Solution Explorer, [add a reference to the C1InputPanel assembly](#) (page 10).
2. From the View menu, select **Code**.
3. Add the following import statement:

- Visual Basic

```
Imports C1.Win.C1InputPanel
```

- C#

```
using C1.Win.C1InputPanel;
```

4. To add the **C1InputPanel** to your form, add the following code to the **Form_Load** event.

- Visual Basic

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load  
    Dim NewPanel As New C1InputPanel  
    Controls.Add(NewPanel)  
End Sub
```

- C#

```
private void Form1_Load(object sender, System.EventArgs e) {  
    C1InputPanel NewPanel = new C1InputPanel();  
    Controls.Add(NewPanel);  
}
```

5. Using the Solution Explorer, add an existing item to the project. For this example, select an image of your choice.
6. To place the **InputImage** component on the **InputPanel** add the following code at the end of the **Form_Load** event:

- Visual Basic

```
Dim NewImage As InputImage = New InputImage  
NewImage.Image = Image.FromFile("[your image location/name]")  
X.Items.Add(NewImage)
```

- C#

```
InputImage NewImage = new InputImage();  
NewImage.Image = Image.FromFile("[your image location/name]");  
X.Items.Add(NewImage);
```

7. Run the program and observe that your image now appears in the **C1InputPanel**.

Arranging C1InputPanel's Input Components

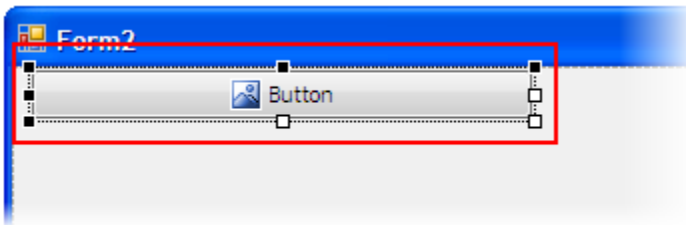
The following topics illustrate how to arrange input components on a **C1InputPanel** control.

Arranging Input Components Using the HorizontalAlign Property

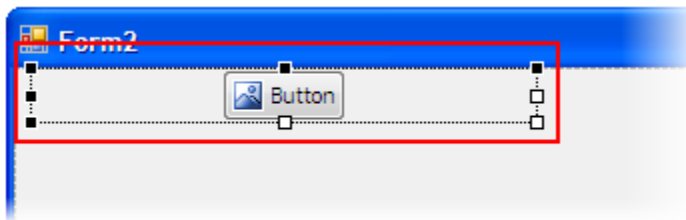
You can easily arrange components on a C1InputPanel control by setting the HorizontalAlign property of each component.

To arrange C1InputPanel components using the HorizontalAlign property, complete these steps:

1. Click C1InputPanel's smart tag (📌) to open the **C1Input Tasks** menu.
2. Click the **Add New Item** drop-down arrow and select **InputButton**.
3. Right click on the [InputButton](#) to open its context menu and select **Properties**.
The Properties window opens with [InputButton](#)'s properties in focus.
4. Set the Width property to "250". Observe that the button stretches horizontally to fill its container.



5. Click the drop-down arrow for the HorizontalAlign property and select a setting. For this example, select **Center**.
6. Press ENTER to apply the settings and observe that the [InputButton](#) is now centered in its container.

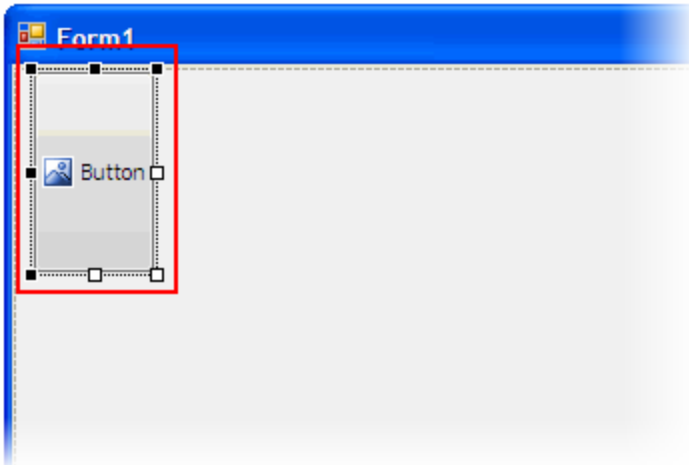


Arranging Input Components Using the VerticalAlign Property

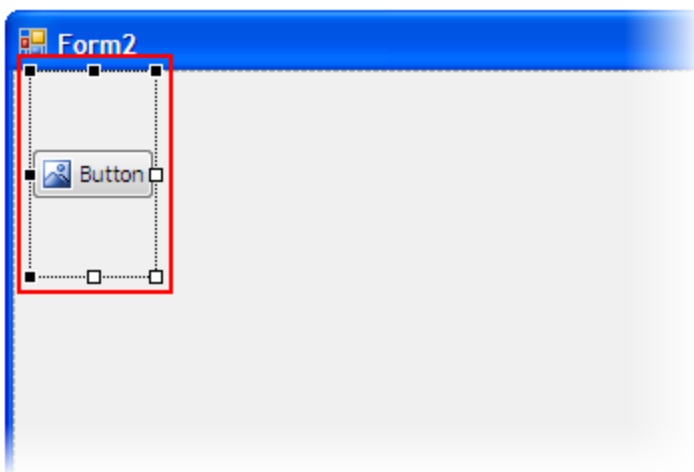
You can easily arrange components on a C1InputPanel control by setting the VerticalAlign property of each component.

To arrange C1InputPanel components using the VerticalAlign property, complete these steps:

1. Click C1InputPanel's smart tag (📌) to open the **C1Input Tasks** menu.
2. Click the **Add New Item** drop-down arrow and select **InputButton**.
3. Right click on the [InputButton](#) to open its context menu and select **Properties**.
The Properties window opens with [InputButton](#)'s properties in focus.
4. Set the Height property to "100". Observe that the button stretches vertically to fill its container.



5. Click the drop-down arrow for the `VerticalAlign` property and select a setting. For this example, select **Center**.
6. Press ENTER to apply the settings and observe that the `InputButton` is now vertically centered in its container.



Arranging Input Components Using Padding

You can easily arrange components on a `C1InputPanel` control by setting the **Padding** property of each component. In this topic, you will create a form with six components and manipulate them to create an aesthetically pleasing layout.



Arranging Components Using the Designer

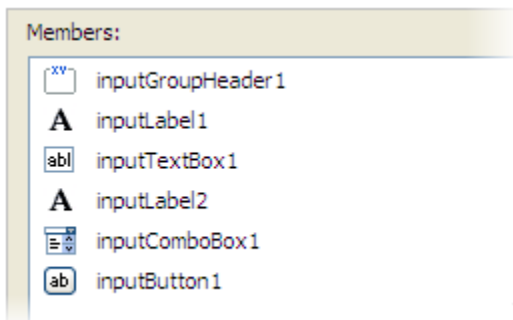
To arrange `C1InputPanel` components using the designer, follow these steps:

1. Click `C1InputPanel`'s smart tag (☑) to open the **C1Input Tasks** menu and then select **Edit Items**.
The **C1InputPanel Item Collection Editor** opens.
2. To add components to the panel, select them from the list box and then click **Add**. For this topic, we will add the following components to the panel:
 - Group Header

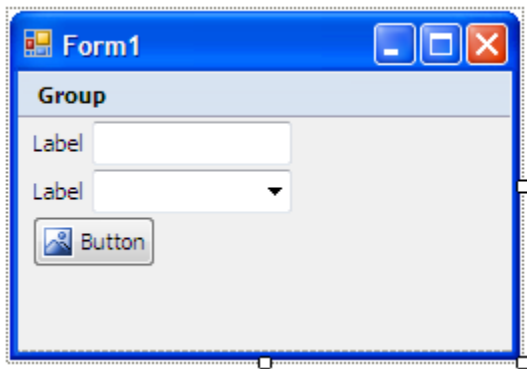
- InputLabels (2)
- InputTextBox
- InputComboBox
- InputButton

Observe that the components appear in the Members pane *and* on the **C1InputPanel** control as they are added.

3. Use the arrow buttons ( ) to rearrange the components so that they are in the same order as the components in the following image:



After reorganizing the components, your **C1InputPanel** will resemble the following image:



4. By default, the padding settings for each component are set to '0', meaning that there is little space between the components. To increase the padding of the components, complete the following steps:
- In the Members pane, select **inputLabel1** to reveal its list of properties in the Property grid. Expand the node next to the **Padding** property and set it as follows:

Padding Area	Setting
Left	5
Top	20
Right	20
Bottom	10

- b. Select **inputTextBox1** from the Members pane, click the node next to the **Padding** property, and set the padding of the component as follows:

Padding Area	Setting
Left	0
Top	10
Right	0
Bottom	0

- c. Select **inputLabel2** from the Members pane, click the node next to the **Padding** property, and set the padding of the component as follows:

Padding Area	Setting
Left	5
Top	0
Right	20
Bottom	0

- d. Select **inputButton1** from the Members pane, click the node next to the **Padding** property, and set the padding of the component as follows:

Padding Area	Setting
Left	120
Top	22
Right	0
Bottom	0

5. Click **OK** to exit the **C1InputPanel Collection Editor**.

Arranging Components Programmatically



To arrange **C1InputPanel** components programmatically, complete the following steps:

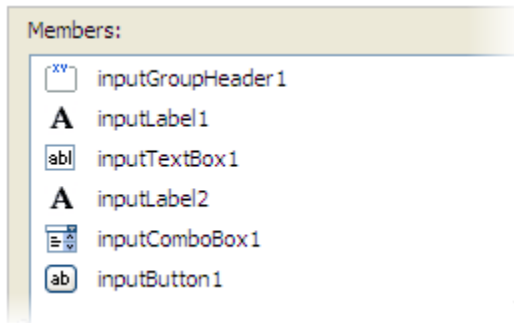
1. Click **C1InputPanel**'s smart tag (**Ⓛ**) to open the **C1Input Tasks** menu. Select **Edit Items**.

The **C1InputPanel Item Collection Editor** opens.

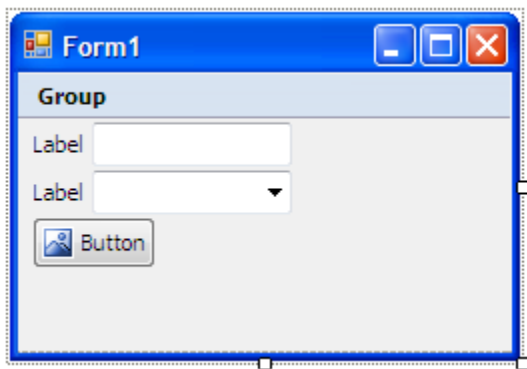
2. To add components to the panel, select them from the list box and then click **Add**. For this topic, we will add the following components to the panel:
 - Group Header
 - InputLabels (2)

- InputTextBox
- InputComboBox
- InputButton

3. Use the arrow buttons ( ) to rearrange the components so that they are in the same order as the components in the following image:



4. Click **OK** to close the **C1InputPanel Item Collection Editor**. Observe that your **C1InputPanel** resembles the following image:



5. Double-click the form. This will open the code editor and add a **Form_Load** event to the project.

6. Make sure that the following namespaces are imported into your project:

- Visual Basic


```
Imports System.Windows.Forms
Imports C1.Win.C1InputPanel
```

- C#


```
using System.Windows.Forms;
using C1.Win.C1InputPanel;
```

7. Set the **Padding** properties for **inputLabel1**, **inputLabel2**, **inputTextBox1**, and **inputButton1** by adding the following code to the **Form_Load** event:

- Visual Basic


```
inputLabel1.Padding = New Padding(5,20,20,10)
inputLabel2.Padding = New Padding(5, 0, 20, 0)
inputTextBox1.Padding = New Padding(0, 10, 0, 0)
```

```
inputButton1.Padding = new Padding(120, 22, 0, 0)
```

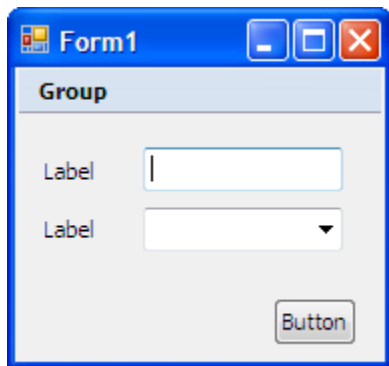
- C#

```
inputLabel1.Padding = new Padding(5,20,20,10);  
inputLabel2.Padding = new Padding(5, 0, 20, 0);  
inputTextBox1.Padding = new Padding(0, 10, 0, 0);  
inputButton1.Padding = new Padding(120, 22, 0, 0);
```

8. Run the program.

✔ **This Topic Illustrates the Following:**

The result of this topic will resemble the following image:



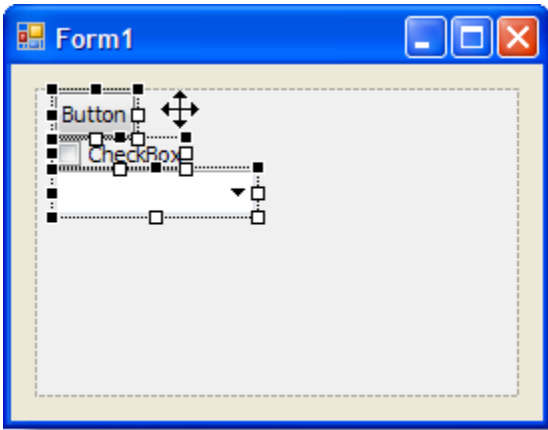
Arranging the Input Components into Columns

By default, components added to **C1InputPanel** start in a new row. Using either the Designer or the Code Editor, you can organize the input items in many arrangements. To organize them in columns, complete one of the following:

Arranging Input Components Using the Designer

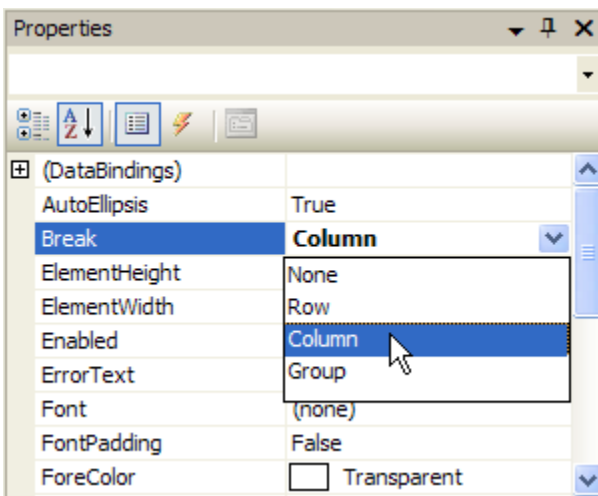
To arrange input components in the designer, complete these steps:

1. Add the following components to your form:
 - C1InputPanel
 - InputButton
 - InputCheckBox
 - InputComboBox
2. While holding CTRL, select each of the components that you want to align in a column.

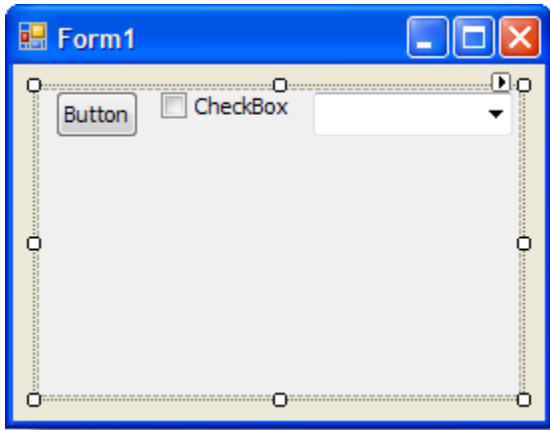


Note: If you have an InputGroupHeader on your form, do not select it while completing Step 1: The **Break** property will not be available if you do so.

3. Choose **Column** from the **Break** property.



4. Select **OK**. All the input items you selected are now arranged columns.



Arranging Input Components Programmatically

To arrange input components programmatically, complete these steps:

1. Import the following namespace into the project:

- Visual Basic
Imports C1.Win.C1InputPanel

- C#
using C1.Win.C1InputPanel;

2. Add **C1InputPanel** to the **Form_Load** event:

- Visual Basic
Dim NewPanel As New C1InputPanel
Controls.Add(NewPanel)
NewPanel.Height = 300
NewPanel.Width = 300

- C#
C1InputPanel NewPanel = new C1InputPanel();
Controls.Add(NewPanel);
NewPanel.Height = 300;
NewPanel.Width = 300;

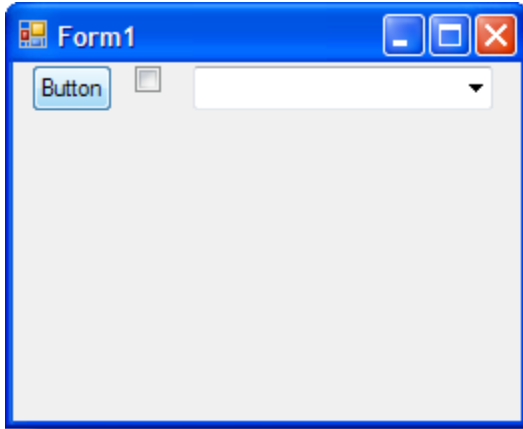
3. Add the following code to the **Form_Load** event to add an **InputComboBox**, an **InputCheckBox**, and an **InputButton** to the **C1InputPanel**:

- Visual Basic
Dim NewComboBox As New InputButton
NewPanel.Items.Add(NewButton)
NewButton.Text = "Button"
NewButton.Break = BreakType.Column
Dim New CheckBox As New InputCheckBox
NewPanel.Items.Add(New CheckBox)
New CheckBox.Break = BreakType.Column
Dim NewComboBox As New InputComboBox
NewPanel.Items.Add(NewComboBox)
NewComboBox.Width = 150
NewComboBox.Break = BreakType.Column

- C#
InputButton NewComboBox = new InputButton();
NewPanel.Items.Add(NewComboBox);

```
NewComboBox.Text = "Button";
NewComboBox.Break = BreakType.Column;
InputCheckBox New CheckBox = new InputCheckBox();
NewPanel.Items.Add(NewCheckBox);
New CheckBox.Break = BreakType.Column;
InputComboBox NewComboBox = new InputComboBox();
NewPanel.Items.Add(NewComboBox);
NewComboBox.Width = 150;
NewComboBox.Break = BreakType.Column;
```

4. Run the program and observe that the components are arranged in columns.



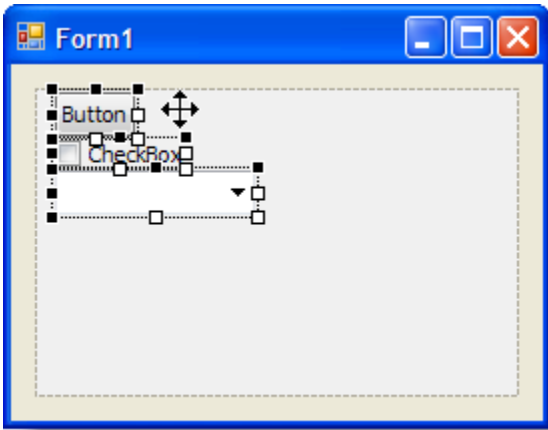
Arranging the Input Components into Groups

You can arrange the input components in several arrangements. To organize input components into groups, complete one of the following:

Arranging Input Components into Groups Using the Designer

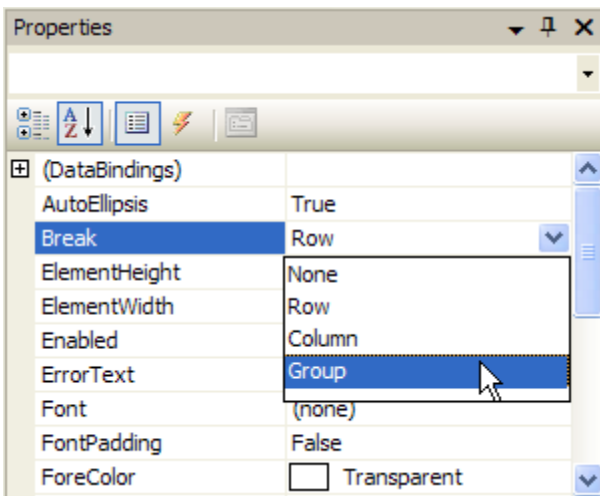
To arrange input components into groups, complete these steps:

1. Add the following components to your form:
 - C1InputPanel
 - InputButton
 - InputCheckBox
 - InputComboBox
2. While holding the **CTRL** button, select each of the components that you want to align in a group.



Note: If you have an **InputGroupHeader** on your form, do not select it while completing Step 1; The **Break** property will not be available if you do so.

3. Choose **Column** from the **Break** property.



4. Select **OK**. The input items you selected are now arranged in a group.

Arranging Input Components Programmatically

To arrange input components into groups, complete these steps:

1. Import the following namespace into your project:

- Visual Basic
Imports C1.Win.C1InputPanel

- C#
using C1.Win.C1InputPanel;

2. Add **C1InputPanel** to the **Form_Load** event:

- Visual Basic
Dim NewPanel As New C1InputPanel
Controls.Add(NewPanel)

```
NewPanel.Height = 300
NewPanel.Width = 300
```

- C#

```
C1InputPanel NewPanel = new C1InputPanel();
Controls.Add(NewPanel);
NewPanel.Height = 300;
NewPanel.Width = 300;
```

3. Add this code, which creates the input components and adds them to the panel, to the **Form_Load** event:

- Visual Basic

```
Dim NewComboBox As New InputButton
NewPanel.Items.Add(NewButton)
NewButton.Text = "Button"
NewButton.Break = BreakType.Group
Dim NewCheckBox As New InputCheckBox
NewPanel.Items.Add(NewCheckBox)
NewCheckBox.Break = BreakType.Group
Dim NewComboBox As New InputComboBox
NewPanel.Items.Add(NewComboBox)
NewComboBox.Width = 150
NewComboBox.Break = BreakType.Group
```

- C#

```
InputButton NewComboBox = new InputButton();
NewPanel.Items.Add(NewComboBox);
NewComboBox.Text = "Button";
NewComboBox.Break = BreakType.Group;
InputCheckBox NewCheckBox = new InputCheckBox();
NewPanel.Items.Add(NewCheckBox);
NewCheckBox.Break = BreakType.Group;
InputComboBox NewComboBox = new InputComboBox();
NewPanel.Items.Add(NewComboBox);
NewComboBox.Width = 150;
NewComboBox.Break = BreakType.Group;
```

4. Run the program and notice that the components are arranged in groups.

Creating a Spacing Scheme for Input Components

By default, the components added to a C1InputPanel control are in close proximity to one another. By setting the value of the **ChildSpacing** property, you can create a uniform spacing scheme for your panel.

Note: Once the ChildSpacing property is set, all components that you add to the page will conform to that setting. However, you can still change the spacing of individual components by manipulating their Padding property. For more information, see [Arranging Input Components Using Padding](#) (page 53).

Setting the ChildSpacing Property the Designer

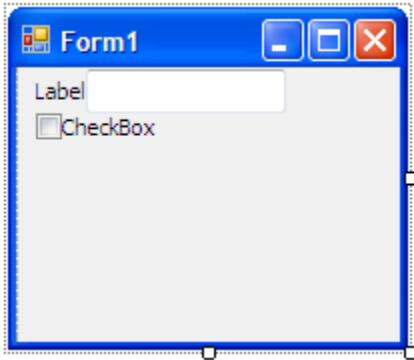
To set the ChildSpacing property, complete these steps:

1. Click **C1InputPanel**'s smart tag (🔗) to open the **C1Input Tasks** menu. Select **Edit Items**.

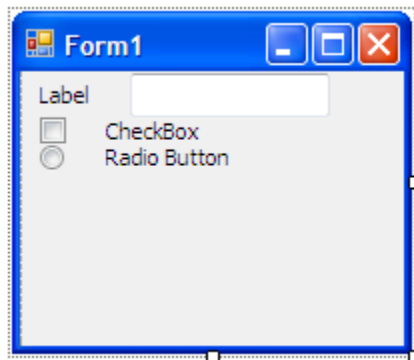
The **C1InputPanel Collection Editor** opens.

2. To add components to the panel, select them from the list box and then click **Add**. For this topic, we will add the following components to the panel:
 - InputLabel
 - InputTextBox

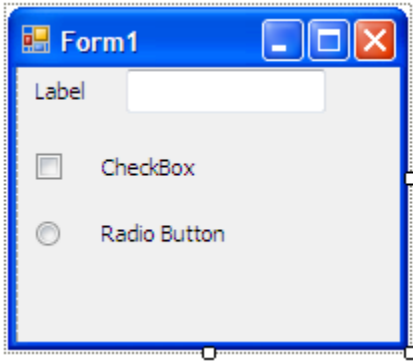
- InputCheckBox
 - InputRadioButton
3. Click **OK** to close the **C1InputPanel Collection Editor**. Observe that your panel resembles the following image:



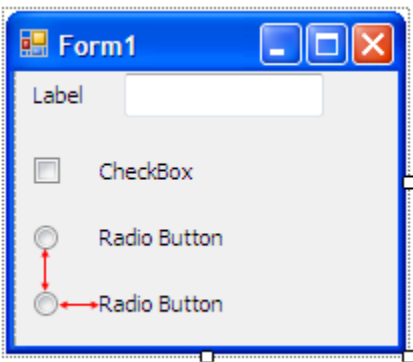
4. With **C1InputPanel** selected on the form, navigate to the Properties window and expand the node for the **ChildSpacing** Property. Observe that you can set the **Width** or **Height** of the property.
5. Set the **Width** of the **ChildSpacing** property to "20" and press ENTER. Notice that there is now a wider space between the components and their labels.



6. Set the Height of the **ChildSpacing** property to "20" and press ENTER. Notice that the vertical spacing between the components has increased.



7. Click **C1InputPanel**'s smart tag (📌) to open the **C1Input Tasks** menu. Select an **InputRadioButton** from the **Add New Item** drop-down list. Observe that the spacing of the new component adheres to the **ChildSpacing** property settings.



Setting the ChildSpacing Property Programmatically



To set the **ChildSpacing** property, complete these steps:

1. Import the following namespace into your project:
 - Visual Basic
`Imports C1.Win.C1InputPanel`
 - C#
`using C1.Win.C1InputPanel;`
2. To set the **ChildSpacing** property, add following code to the **Form_Load** event:
 - Visual Basic
`c1InputPanel1.ChildSpacing = New Size(20, 20)`
 - C#
`c1InputPanel1.ChildSpacing = new Size(20, 20);`
3. Run the program.

Moving Items in a C1InputPanel

To arrange the collection of input components using the **C1InputPanel Item Collection Editor**:

1. Select the **C1InputPanel** control and click on its smart tag (📌) to open the **C1InputPanel Tasks** menu.
2. Click on **Edit Items** from the **C1InputPanel Tasks** menu.

3. Select the item in the list box that you want to move and click on the **Move Up** button  to move the item up a position or click on the **Move Down**  button to move the item down a position.

Removing Input Components from C1InputPanel



You can remove input components from the C1InputPanel control using the designer or code.

Remove Input Components in the Form Window

To remove an input component using the keyboard, just select the component and press the DELETE key.

Remove Input Components Using the Designer

To remove an input component, complete these steps:

1. Complete these steps to remove components using the collection editor:
2. Click C1InputPanel's smart tag  to open the **C1InputPanel Tasks** menu. Select **Edit Items**.
3. The **C1InputPanel Collection Editor** opens.
4. In the Members pane, select the component you wish to remove and then click the **Remove** button .
5. Click **OK** to close the **C1InputPanel Collection Editor**.

Remove Input Components Programmatically

Input components can be removed from the panel with the InputComponentCollection.Remove method. The following code example will remove an **InputButton** component from the panel:

- Visual Basic


```
c1InputPanel1.Items.Remove(inputButton1)
```
- C#

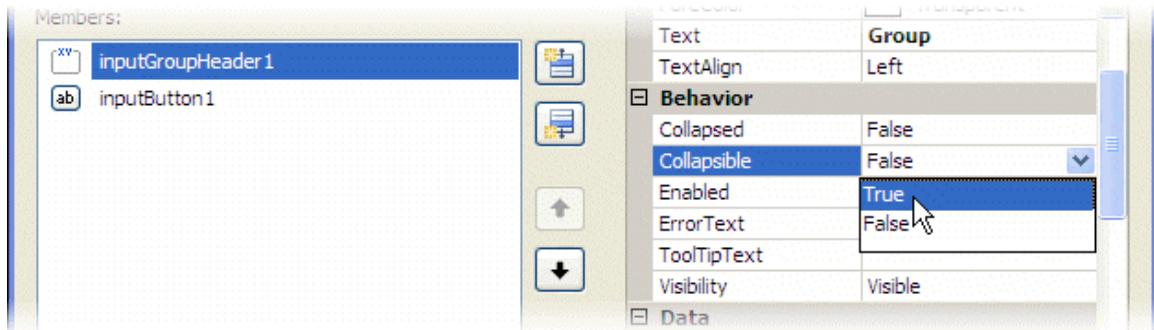
```
c1InputPanel1.Items.Remove(inputButton1);
```

Creating a Collapsible Input Group

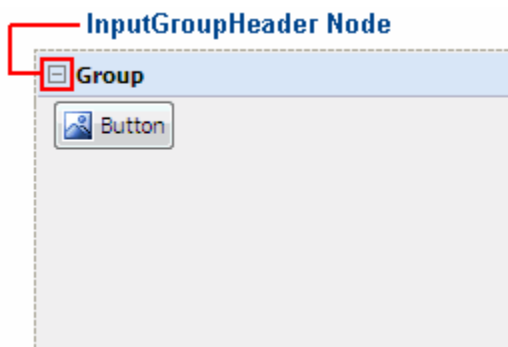
You can use the [InputGroupHeader](#) component to create a collapsible group. When the project is run, users can collapse or expand the group's input components by clicking a node.

To create a collapsible input group, follow these steps:

1. Click C1InputPanel's smart tag  to open the **C1Input Tasks** menu and then select **Edit Items**. The **C1InputPanel Item Collection Editor** opens.
2. Select an [InputGroupHeader](#) component from the drop-down list and click **Add** to add it to the panel.
3. Select an [InputButton](#) component from the drop-down list and click **Add** to add it to the panel.
4. In the Members pane, select [InputGroupHeader](#) and then set its Collapsible property to **True**.



- Click **OK** to close the **C1InputPanel Item Collection Editor**. Observe that there is a node present on the **InputGroupHeader** component.



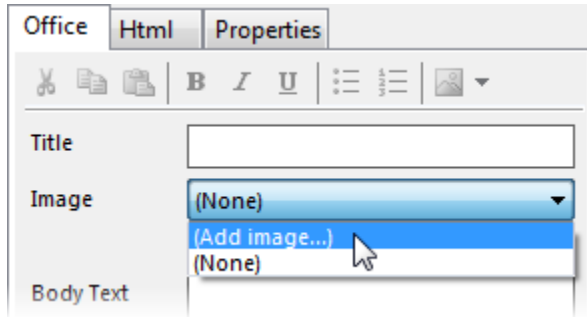
- Press **F5** to build the project.
C1InputPanel loads with the input group expanded.
- Click the node to collapse the group, and then click the node again to expand the group.

Creating Rich ToolTips for C1InputPanel Items

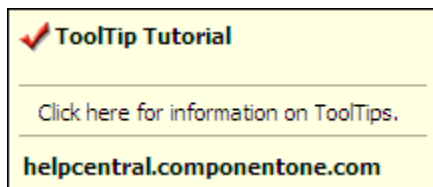
You can easily create rich ToolTips for input components using the **ToolTip Editor**. In this topic, we will add a ToolTip to an **InputButton** component.

To add a rich ToolTip to an input item, follow these steps:

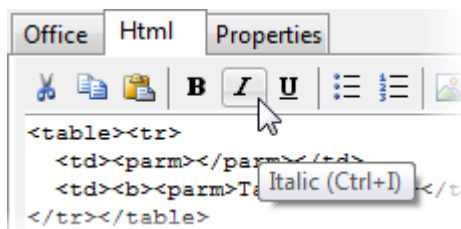
- Click **C1InputPanel**'s smart tag (📌) to open the **C1Input Tasks** menu. Select **Edit Items**.
The **C1InputPanel Item Collection Editor** opens.
- Select an [InputButton](#) component from the drop-down list and click **Add** to include it in the panel.
- Locate the **ToolTipText** property and click its ellipsis button (...).
The **ToolTipText** editor opens with the **Office** tab page in focus.
- Complete the following tasks in the **Office** tab page:
 - Set the **Title** to "ToolTip Tutorial".
 - Click the **Image** drop-down and select **Add Image** to access the **Open** dialog box. Select a small image (an icon of 16x16 pixels is best) and click **Open**.



- c. Check the **Top Separator** box.
- d. Type the following text into the **Body Text** text box: "Click for information on ToolTips."
- e. Check the **Bottom Separator** box.
- f. Type "helpcentral.componentone.com" into the **Subtitle** text box.
- g. The Preview pane will display an image similar to the following:



8. Click the Html tab to bring its tab page into focus and then complete the following steps:
 - a. In the editor, delete the `` and `` tags that enclose "helpcentral.componentone.com".
 - b. Select "Click here for information on ToolTips." and click the **Italics** button.



- c. Place your cursor in front of the title ("ToolTip Tutorial") and type `` . Move your cursor to the end of the title and type ``.

The Preview pane will display an image similar to the following:



9. Click the Properties tab to bring its tab page into focus and then complete the following steps:
 - a. Click the ellipsis (...) button next to the **BackgroundImage** property to access the **Select Resource** dialog box.
 - b. Click the **Local resource** radio button, click **Import** to access the **Open** dialog box, and select a background image. Click **Open** to import the image.
 - c. Set the following properties:
 - Set the **BackgroundImageLayout** property to **Tile**.
 - Set the **Border** property to **True**.
 - Set the **BorderColor** property to **Red**.
 - Set the **IsBalloon** property to **True**.
 - Set the **IntialDelay** property to "50".
10. Click **OK** to close the **ToolTip Editor**, then click **OK** to close the **C1InputPanel Collection Editor**.
11. Press F5 to build the project and then hover over the input button with your cursor. The result will resemble the following image:



Exporting and Importing XML Layout Files

The layout of **C1InputPanel**'s input components can be saved as an XML file. In this topic, you will learn how to export and import an XML layout file for a **C1InputPanel** component.

Step 1: Export the XML Layout

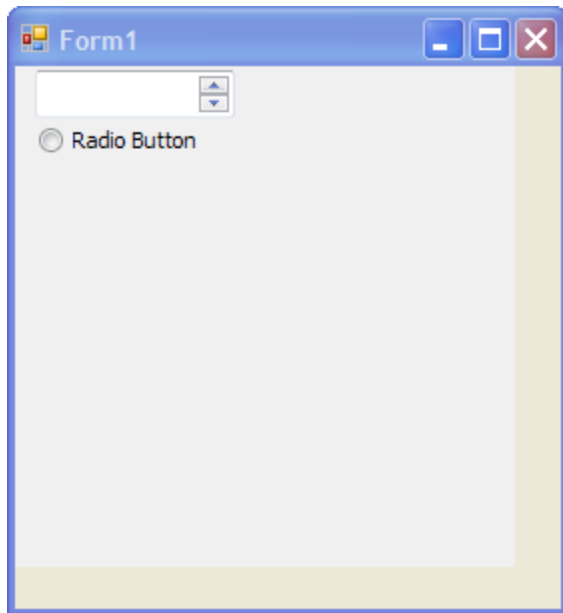
To export an XML layout, follow these instructions:

1. Add a **C1InputPanel** control to the form.
2. Click **C1InputPanel**'s smart tag (☒) to open the **C1InputPanel Tasks** menu and select **Edit Items**.

The **C1InputPanel Collection Editor** opens.

3. Select an **InputNumericBox** from the drop-down list and click **Add**.
4. Select an **InputRadioButton** from the drop-down list and click **Add**.
5. Click **OK** to close the **C1InputPanel Collection Editor**.

The **C1InputPanel** now displays an **InputNumericBox** and an **InputRadioButton**.

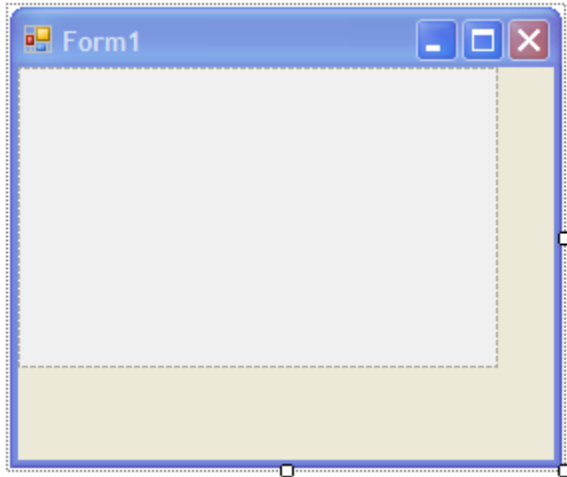


6. Click **C1InputPanel**'s smart tag (📌) to open the **C1InputPanel Tasks** menu and select **Save as XML file**.
The **Save as XML file** dialog box opens.
7. Save the "TutorialLayout.xml". Take note of where you save this; you'll need it later.
8. Click **OK** to close the **Save as XML file** dialog box.
9. Delete **c1InputPanel1** from the Form.

Step 2: Import the XML Layout

To import the XML layout that we created in **Step 1: Export the XML Layout**, complete these instructions:

1. Add a **C1InputPanel** control to your blank Form. The control will be blank.



2. Click **C1InputPanel**'s smart tag (📌) to open the **C1InputPanel Tasks** menu and select **Load from XML file**.

The **Load from XML** dialog box opens.

3. Navigate to the location where you saved "TutorialLayout.xml", select the file, and then click **OK** to close the **Load from XML** dialog box. Observe that the components you added in **Step 1: Export the XML Layout** now appear on the **C1InputPanel** control.

Formatting C1InputPanel's Error Frame

The **C1InputPanel** component features an error frame that provides a visual alert to the user when an invalid value is entered in any of the input panel components. In this topic, you will learn how to format the error frame.

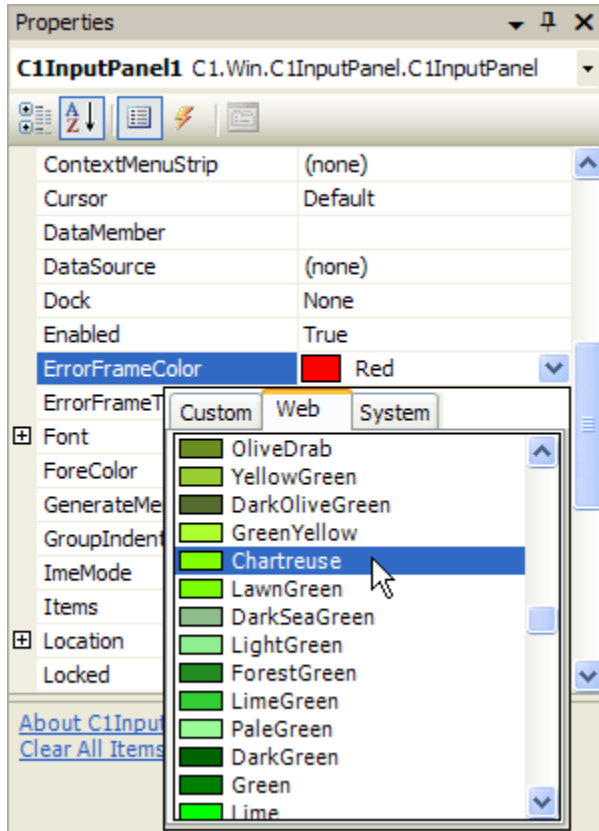
The **ErrorFrameColor** property is used to set the color of the error frame, and the **ErrorFrameThickness** property is used to set the thickness of the error frame. The default color is red, and the default width is 2.

To customize the **C1InputPanel**'s error frame:

Formatting the Error Frame Using the Designer

To format the error frame, complete these steps:

1. Click **C1InputPanel**'s smart tag (📌), click the **Add New Item** dropdown arrow, and select [InputNumericBox](#), from the list of items.
2. From the Properties window, select the **C1InputPanel** control and edit the **ErrorFrameColor** property to a desired color. For this example, select **Chartreuse**.

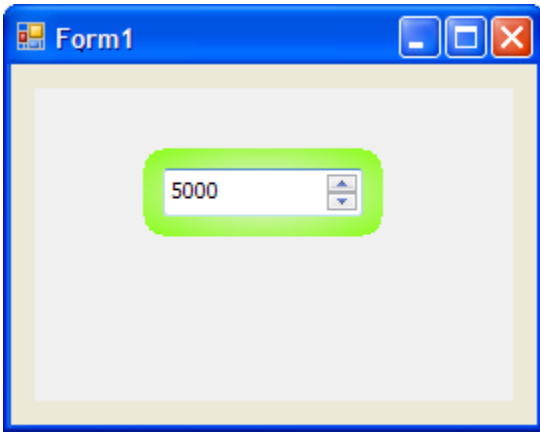


3. Edit the **ErrorFrameThickness** to a desired size. For this example, enter **10**.

Note: To see the whole **ErrorFrame** when the program runs, you might need to increase the padding of the **InputNumericBox**. For more information, see [Arranging Input Components Using Padding](#) (page 53).

4. Build the project.
5. In the **InputNumericBox**, enter a number greater than 100 and press ENTER on your keyboard. The number must be greater than 100 because the default maximum value for the **InputNumericBox** component is 100. Edit the **C1InputPanel** value to change this setting.

Notice the thick, chartreuse-colored border around the **InputNumericBox**; this indicates an error by the end-user.



Formatting the Error Frame Programmatically

To format the error frame, complete these steps:

1. Import the following namespace into your project:

- Visual Basic
`Imports C1.Win.C1InputPanel`

- C#
`using C1.Win.C1InputPanel;`

2. Place the **InputPanel** control and the **InputNumericBox** component on your form by adding the following code to the **Form_Load** event.

- Visual Basic
`Dim NewPanel As New C1InputPanel
Controls.Add(NewPanel)

Dim NewNumericBox As InputNumericBox = New InputNumericBox
X.Items.Add(NewNumericBox)`

- C#
`C1InputPanel NewPanel = new C1InputPanel();
Controls.Add(NewPanel);

InputNumericBox NewNumericBox = new InputNumericBox();
X.Items.Add(NewNumericBox);`

3. To change the color of the error frame, add the following code beneath the last line of code that created to the **C1InputPanel** control. For this example, we'll change the color to **Chartreuse**.

- Visual Basic
`NewPanel.ErrorFrameColor = Color.Chartreuse`

- C#
`NewPanel.ErrorFrameColor = Color.Chartreuse;`

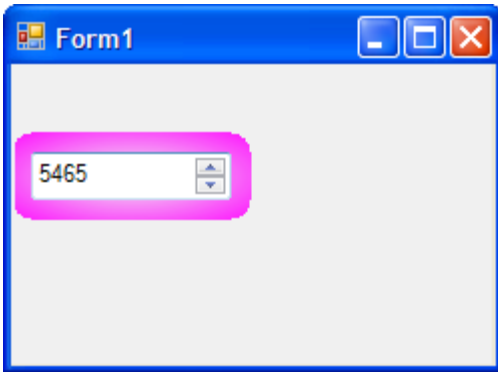
4. To change the size of the error frame, add the following code beneath the last line of code that created to the **C1InputPanel** control. For this example, we'll change the size to **10**.

- Visual Basic
`NewPanel.ErrorFrameThickness = 10`

- C#
`NewPanel.ErrorFrameThickness = 10;`

Note: To see the whole **ErrorFrame** when the program runs, you might need to increase the padding around the **InputNumericBox**. For more information, see [Arranging Input Components Using Padding](#) (page 53).

5. Run the program.
6. In the **InputNumericBox**, enter a number greater than **100** and press ENTER on your keyboard. The number must be greater than 100 because the default maximum value for the **InputNumericBox** component is 100. Edit the **C1InputPanel.InputNumericBox.Maximum** value to change this setting.
Notice the thick, fuchsia-colored border around the **InputNumericBox**; this indicates an error by the end-user.



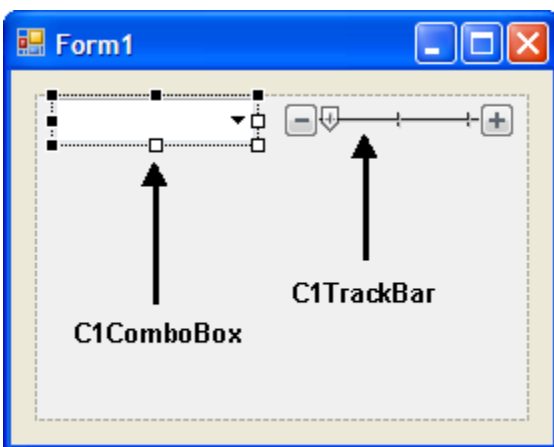
Using C1InputPanel's InputSeparator

By adding the **InputSeparator** component to your **C1InputPanel** control, you can create a visual marker between input components.

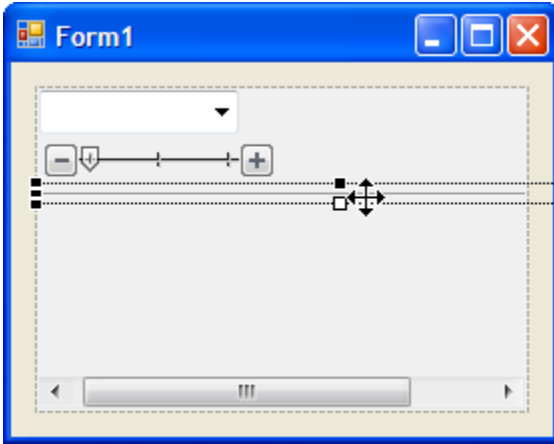
Adding an **InputSeparator** in the Designer

To add an **InputSeparator** component to **C1InputPanel**, complete these steps:

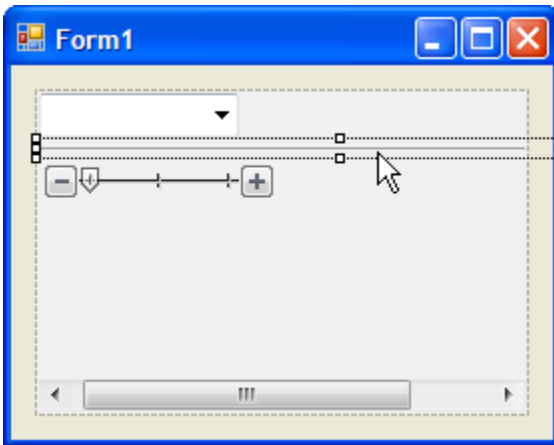
1. Add an **InputPanel** control, an **InputPanel.ComboBox** component and an **InputPanel.TrackBar** to your form.



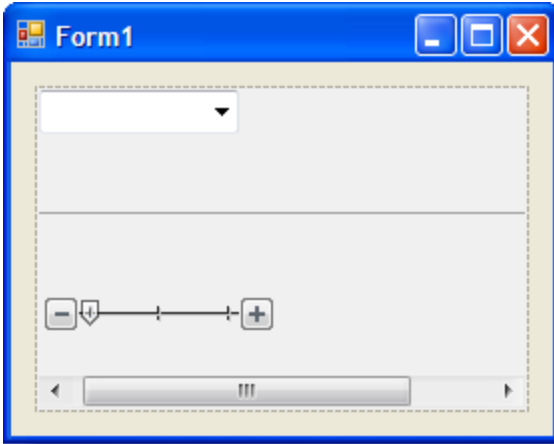
2. Click the **C1InputPanel** smart tag (▾) and select the **InputSeparator** from the **Add New Item** dropdown.
3. Using the cursor, pick up the **InputSeparator** component on your form and place it in between the **InputComboBox** component and the **InputTrackBar** component.



4. Your form should now have the **InputSeparator** component placed in between your two components.



5. To add space in between your two components, increase the distance of the **InputSeparator.Height** property. By default, the value of this property is 7. For this example, increase the value to **75**.



Adding a Separator Bar Programmatically

To add an InputSeparator component to C1InputPanel, complete these steps:

1. Import the following namespace into your project:

- Visual Basic
Imports C1.Win.C1InputPanel

- C#
using C1.Win.C1InputPanel;

2. To place the InputPanel control, the InputComboBox component and the InputTrackBar on your form, add the following code to the **Form_Load** event.

- Visual Basic
Dim X As New C1InputPanel
Controls.Add(X)

Dim T As InputTrackBar = New InputTrackBar
X.Items.Add(T)

Dim Q As InputComboBox = New InputComboBox
X.Items.Add(Q)

- C#
C1InputPanel X = new C1InputPanel();
Controls.Add(X);

InputTrackBar T = new InputTrackBar();
X.Items.Add(T);

InputComboBox Q = new InputComboBox();
X.Items.Add(Q);

3. In between the lines of code that create the **InputTrackBar** component and the **InputComboBox** component, insert the following code to create the **InputSeparator**.

- Visual Basic
Dim A As InputSeparator = New InputSeparator
X.Items.Add(A)

- C#
InputSeparator A = new InputSeparator();
X.Items.Add(A);

4. To increase the spacing between the two components, add the following code to the **InputSeparator** lines.

- Visual Basic
`A.Height = 75`

- C#
`A.Height = 75;`

Working with InputDataNavigator Items

The following topics illustrate how to add and remove items and buttons from the [InputDataNavigator](#).

Adding Custom Images to InputDataNavigator Items

In this topic, you will add a custom image to an [InputDataNavigator](#) item using the designer and code.

The image used in this topic appears in the table below. To use this image, right-click the graphic to open its context menu, select **Save Picture As**, and then save the image as a .bmp file.



If you prefer, you can use your own 16 x 16 bitmap image.

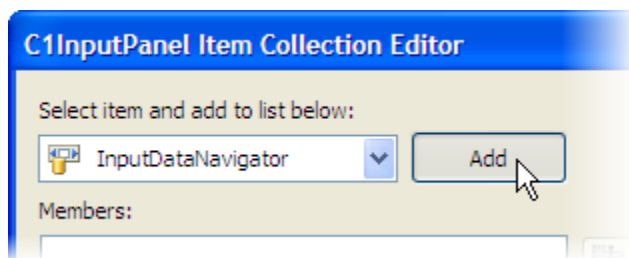
Adding Custom Images Using the Designer

To add custom images, complete these steps:

1. Click C1InputPanel's smart tag (🔗) to open the **C1Input Tasks** menu and then select **Edit Items**.

The **C1InputPanel Collection Editor** opens.

2. Select an [InputDataNavigator](#) from the drop-down list and click **Add**.



An [InputDataNavigator](#) component appears in the Members pane.

3. Select **inputDataNavigator1** from the Members pane to reveal its list of properties.

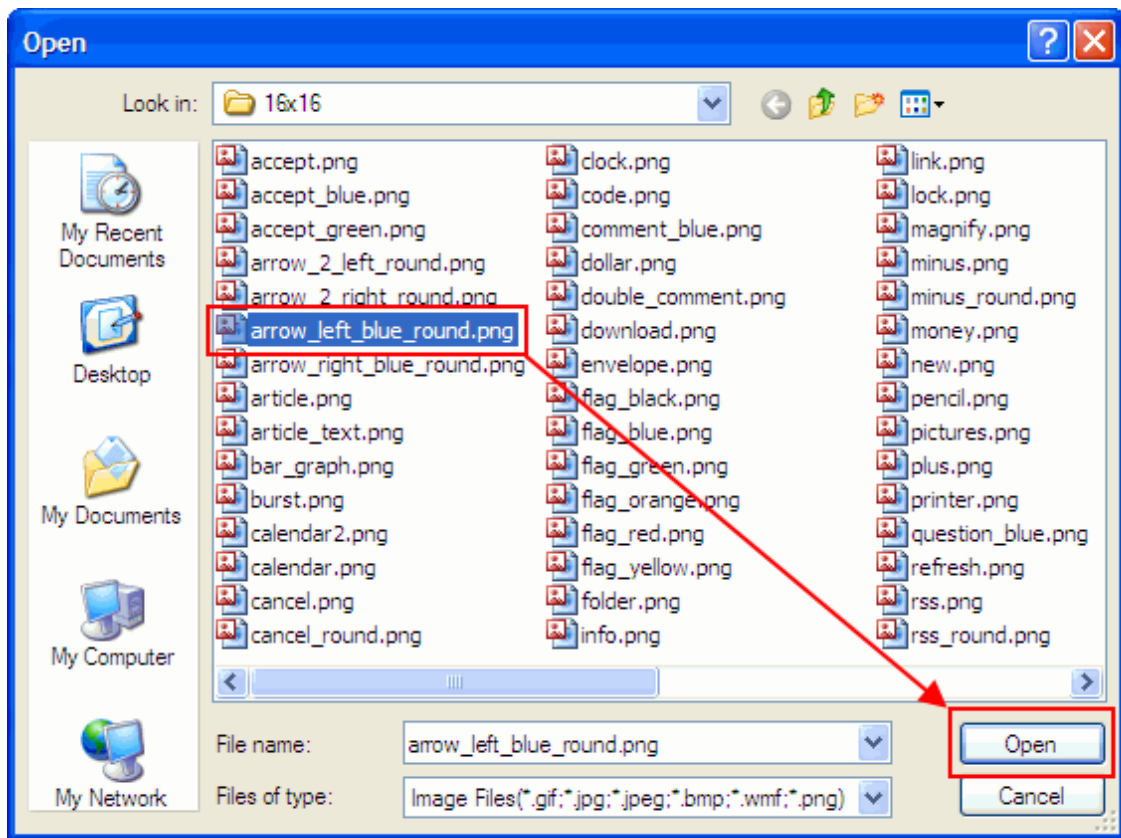
4. Locate the MovePreviousImage property and click its ellipsis button (⋮).

The **Select Resource** dialog box opens.

5. Click **Import**.

The **Open** dialog box opens.

6. Navigate to your custom image, select the image file, and then click **Open** to open the file.

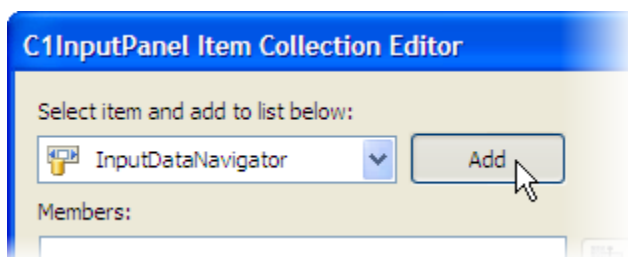


7. Click **OK** to close the **Select Resource** dialog box.
8. Click **OK** to close the **C1InputPanel Collection Editor** and observe that the **Move Previous** button now appears with your custom image.

Adding Custom Images Programmatically

To add custom images, complete these steps:

1. Click C1InputPanel's smart tag (🔗) to open the **C1Input Tasks** menu and then select **Edit Items**.
The **C1InputPanel Collection Editor** opens.
2. Select an [InputDataNavigator](#) from the drop-down list and click **Add**.



An **InputDataNavigator** component appears in the Members pane.

3. Click **OK** to close the **C1InputPanel Collection Editor**.

4. Double-click **Form1** to add a **Form_Load** event to Code view.

5. Import the following namespace into your project:

- Visual Basic

```
Imports C1.Win.InputPanel
```

- C#

using C1.Win.InputPanel; Add the following code to the **Form_Load** event to set the **MovePreviousImage** property to the path of your image:

- Visual Basic

```
inputDataNavigator1.MovePreviousImage =  
System.Drawing.Bitmap.FromFile("C:\Images\YourCustomImage.gif")
```

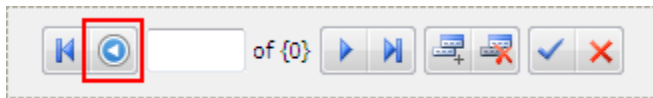
- C#

```
inputDataNavigator1.MovePreviousImage =  
System.Drawing.Bitmap.FromFile(@"C:\Images\YourCustomImage.gif");
```

7. Build the project and observe that the **Move Previous** button now appears with your custom image.

✔ This Topic Illustrates the Following:

The following image shows [InputDataNavigator](#)'s **Move Previous** button with a custom image:



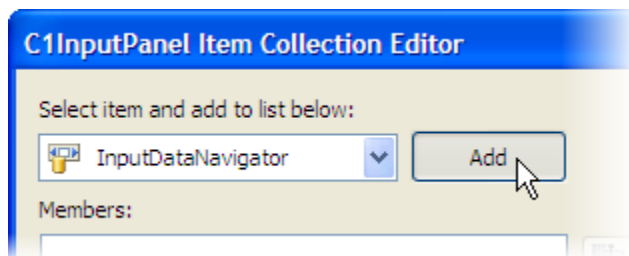
Choosing Items for InputDataNavigator

When an [InputDataNavigator](#) component is added to the C1InputPanel, it appears with ten items by default. In this topic, we will explain how to remove these default buttons from the **InputDataNavigator** component.

Choosing InputDataNavigator Buttons Using the Designer

To choose items for the **InputDataNavigator**, complete these steps:

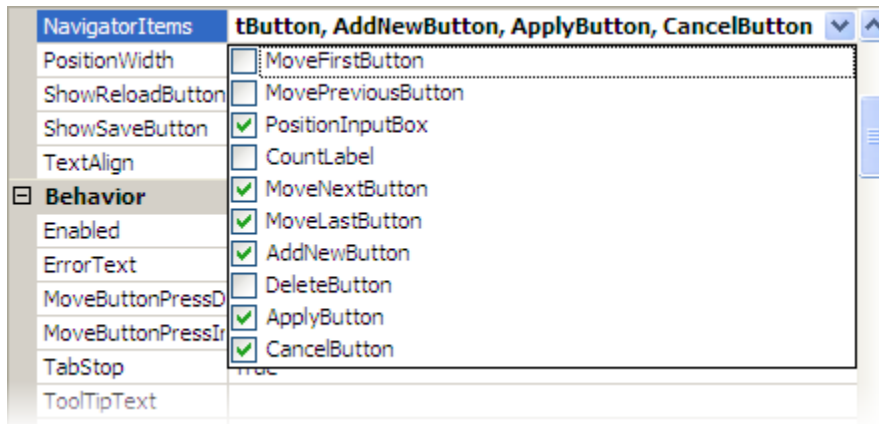
1. Click C1InputPanel's smart tag (🔗) to open the **C1Input Tasks** menu and then select **Edit Items**. The **C1InputPanel Collection Editor** opens.
2. Select an [InputDataNavigator](#) from the drop-down list and click **Add**.



An [InputDataNavigator](#) component appears in the Members pane.

3. Select **inputDataNavigator1** from the Members pane to reveal its list of properties.

- In the Properties grid, click the drop-down for the **NavigatorItems** property and deselect the check boxes for the buttons you wish to have removed. For this example, we will remove the **MoveFirstButton**, **MoveLastButton**, **CountLabel**, and **DeleteButton** items.



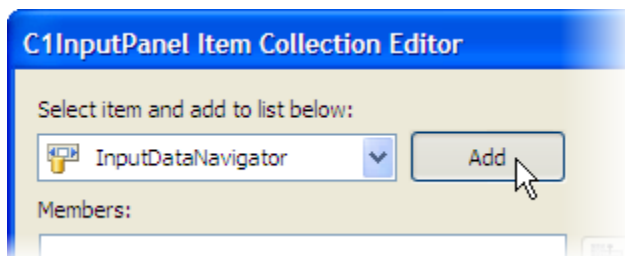
- Click **OK** to close the **C1InputPanel Collection Editor**.

Choosing InputDataNavigator Items Programmatically

To choose **InputDataNavigator** items, add the items you need by setting the **NavigatorButtons** property. Once that property is set, the default settings will be overridden and only the buttons that you specified will appear.

In this section, you will exclude the **MoveFirstButton**, **MoveLastButton**, **CountLabel**, and **DeleteButton** from the **InputDataNavigator** by completing the following steps:

- Click C1InputPanel's smart tag (🔗) to open the **C1Input Tasks** menu and then select **Edit Items**.
The **C1InputPanel Collection Editor** opens.
- Select an [InputDataNavigator](#) from the drop-down list and click **Add**.



An [InputDataNavigator](#) component appears in the Members pane.

- Click **OK** to close the **C1InputPanel Collection Editor**.
- Double-click **Form1** to add a **Form_Load** event to Code view.
- Import the following namespace into your project:

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

6. Add the following code to the **Form_Load** event to add specific buttons to the [InputDataNavigator](#) component:

- Visual Basic

```
inputDataNavigator1.NavigatorItems =  
InputNavigatorItems.ApplyButton Or InputNavigatorItems.AddNewButton  
Or InputNavigatorItems.MovePreviousButton Or  
InputNavigatorItems.PositionInputBox  
Or InputNavigatorItems.CancelButton Or  
InputNavigatorItems.MoveNextButton
```

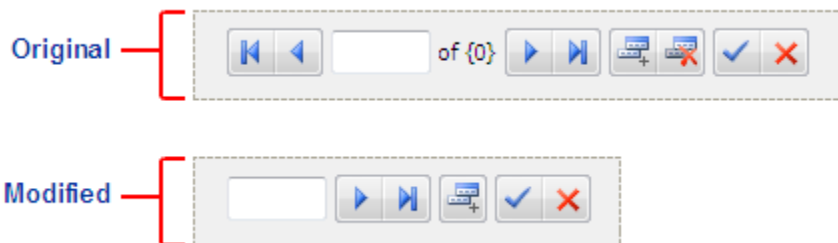
- C#

```
inputDataNavigator1.NavigatorItems =  
InputNavigatorItems.ApplyButton | InputNavigatorItems.AddNewButton  
| InputNavigatorItems.MovePreviousButton |  
InputNavigatorItems.PositionInputBox  
| InputNavigatorItems.CancelButton |  
InputNavigatorItems.MoveNextButton;
```

7. Build the project.

✔ This Topic Illustrates the Following:

The following graphic compares the default **InputDataNavigator** with one that has been modified using the steps in this topic:



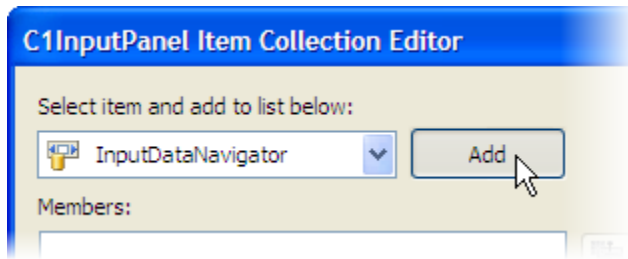
Adding the Reload, Save, and Edit Buttons

When an [InputDataNavigator](#) component is added to the **C1InputPanel** control, it appears with eight default buttons. Three additional buttons – "Save", "Edit", and "Reload" – do not appear on the form by default, but can easily be added to the control.

Using the Designer

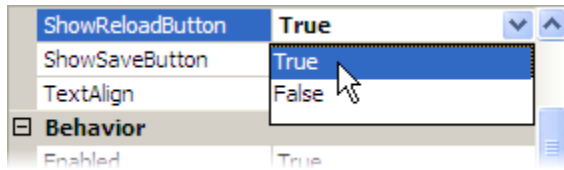
Complete the following steps:

1. Click **C1InputPanel**'s smart tag (📌) to open the **C1Input Tasks** menu and then select **Edit Items**.
The **C1InputPanel Collection Editor** opens.
2. Select an [InputDataNavigator](#) from the drop-down list and click **Add**.



An [InputDataNavigator](#) component appears in the Members pane.

3. Select **inputDataNavigator1** from the Members pane to reveal its list of properties and then complete the following:
 - Set the **ShowReloadButton** property to **True**.

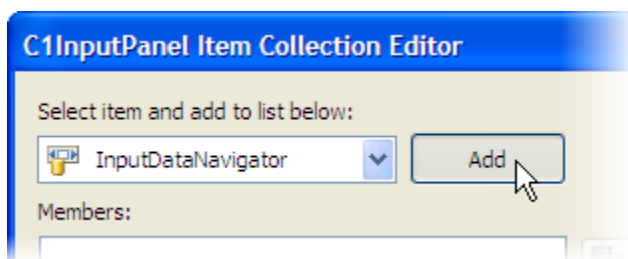


- Set the **ShowSaveButton** property to **True**.
 - Set the **ShowEditButton** property to **True**.
4. Click **OK** to close the **C1InputPanel Collection Editor**.

Using Code

Complete the following steps:

1. Click C1InputPanel's smart tag (🔗) to open the **C1Input Tasks** menu and then select **Edit Items**.
The **C1InputPanel Collection Editor** opens.
2. Select an **InputDataNavigator** from the drop-down list and click **Add**.



An [InputDataNavigator](#) component appears in the Members pane.

3. Click **OK** to close the **C1InputPanel Collection Editor**.
4. Double-click **Form1** to add a **Form_Load** event to Code view.
5. Import the following namespace to your project:
 - Visual Basic

```
Imports C1.Win.C1InputPanel
```

- C#

```
using C1.Win.C1InputPanel;
```

6. Add the this code to the **Form_Load** event:

- Visual Basic

```
inputDataNavigator1.ShowReloadButton = True  
inputDataNavigator1.ShowSaveButton = True  
inputDataNavigator1.ShowEditButton = True
```

- C#

```
inputDataNavigator1.ShowReloadButton = true;  
inputDataNavigator1.ShowSaveButton = true;  
inputDataNavigator1.ShowEditButton = true;
```

7. Run the program.

✔ **This Topic Illustrates the Following:**

The following graphic compares the default [InputDataNavigator](#) with one that has been modified using the steps in this topic:

