ComponentOne

MaskedTextBox for WPF

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third 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)

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

Table of Contents

ComponentOne MaskedTextBox for WPF Overview 1

Help with ComponentOne Studio for WPF 1

Key Features 1

MaskedTextBox for WPF Quick Start 3

Step 1 of 4: Setting up the Application 3

Step 2 of 4: Customizing the Application 4

Step 3 of 4: Adding Code to the Application 4

Step 4 of 4: Running the Application 6

About C1MaskedTextBox 9

Basic Properties 9

Mask Formatting 9

Mask Elements 10

Literals 11

Prompts 11

Watermark 11

Layout and Appearance 12

Appearance Properties 12

Content Properties 12

Text Properties 12

Color Properties 13

Border Properties 13

Size Properties 13

ComponentOne ClearStyle Technology 14

How ClearStyle Works 14

ClearStyle Properties 14

Templates 14

XAML Elements 15

MaskedTextBox for WPF Samples 17

MaskedTextBox for WPF Task-Based Help 17

Setting the Value 17
Adding a Mask for Currency 18
Changing the Prompt Character 19
Changing Font Type and Size........ 19
Locking the Control from Editing........ 20

ComponentOne MaskedTextBox for WPF Overview

Validate input in your WPF applications! **ComponentOne MaskedTextBox™ for WPF** provides a text box with a mask that automatically validates entered input, similar to the standard Microsoft WinForms **MaskedTextBox** control.

For a list of the latest features added to **ComponentOne Studio for WPF**, visit What's New in Studio for WPF.

Getting Started

Get started with the following topics:

- Key Features (page 1)

- Quick Start (page 3)

- Task-Based Help (page 17)

Help with ComponentOne Studio for WPF

Getting Started

For information on installing **ComponentOne Studio for WPF**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for WPF.

What's New

For a list of the latest features added to **ComponentOne Studio for WPF**, visit What's New in Studio for WPF.

Key Features

ComponentOne MaskedTextBox for WPF allows you to create customized, rich applications. Make the most of **MaskedTextBox for WPF** by taking advantage of the following key features:

- **Validate Data and Enhance Your UI**

  The ComponentOne masked text box control (C1MaskedTextBox) provides a text box with a mask that automatically validates the input. The edit mask enhances the UI by preventing end-users from entering invalid characters into the control. See Mask Formatting (page 9) for details and Adding a Mask for Currency (page 18) for an example.

- **Provide Clues to Prompt Users for Information**

  The masked text box control includes a Watermark property, which lets end-users know what type of information is expected. See Watermark (page 11) for details.

- **Easily Change Colors with ClearStyle**

- **C1MaskedTextBox** supports **ComponentOne ClearStyle™** technology which allows you to easily change control brushes without having to override templates. By just setting a few brush properties in Visual Studio you can quickly style the entire control. See ComponentOne ClearStyle Technology (page 14) for details.

MaskedTextBox for WPF Quick Start

The following quick start guide is intended to get you up and running with **MaskedTextBox for WPF**. In this quick start you'll start in Visual Studio and create a new project, add **MaskedTextBox for WPF** controls to your application, and customize the appearance and behavior of the controls.

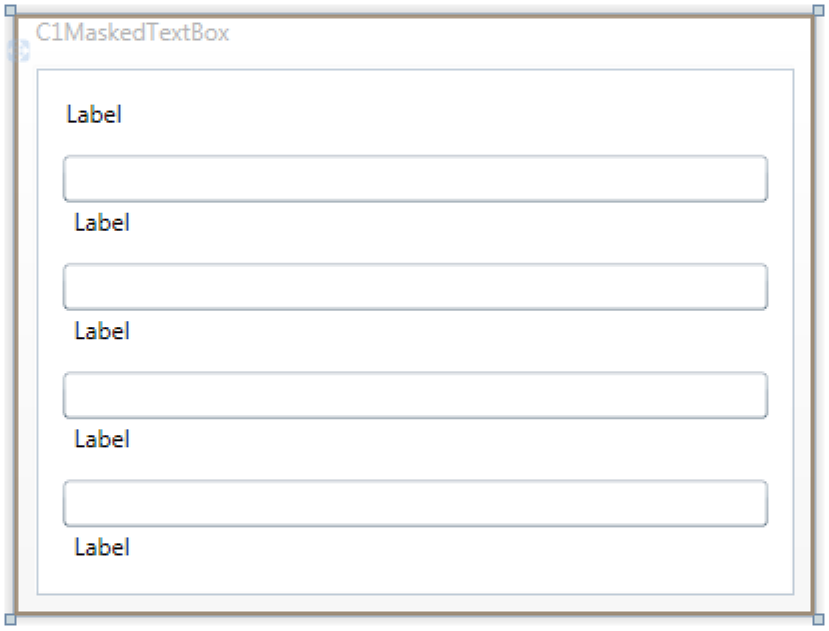
You will create a simple form using several C1MaskedTextBox controls that will demonstrate the difference between the **Text** and **Value** properties. The controls will include various masks and different appearance and behavior settings so that you can explore the possibilities of using **MaskedTextBox for WPF**.

Step 1 of 4: Setting up the Application

In this step you'll begin in Visual Studio to create a WPF application using **MaskedTextBox for WPF**. When you add a C1MaskedTextBox control to your application, you'll have a complete, functional input editor. You can further customize the control to your application.

To set up your project and add C1MaskedTextBox controls to your application, complete the following steps:

1. Create a new WPF project in Visual Studio.
2. Resize the initial window by setting Window1's **Width** to "400".
3. Navigate to the Toolbox and double-click the **C1MaskedTextBox** icon to add the control to Window1. Repeat this step 3 more times to add a total of 4 **C1MaskedTextBox** controls.
4. In the Toolbox, double-click the **Label** icon to add the control to Window1. Repeat this step 4 more times to add a total of 5 standard **Label** controls.
5. Resize the controls and rearrange the controls on the window with the controls numbered smallest to largest from top to bottom alternating **Label** and **C1MaskedTextBox** controls. Your application should now appear similar to the following:



You've successfully created a WPF application and added **C1MakedTextBox** controls to the application. In the next step you'll customize those controls and complete setting up the application.

Step 2 of 4: Customizing the Application

In the previous step you created a new WPF project and added four **C1MaskedTextBox** and five **Label** controls to the application. In this step you'll continue by setting properties to customize those controls.

Complete the following steps:

1. In Design view, click once on the **Label1** control to select it, navigate to the Properties window, and set its **Content** property to "Employee Information".
2. Select each remaining **Label** control in turn, navigate to the Properties window, and set the following for each:
   - Delete the default "Label" text next to **Content** property.
   - Set the **FontSize** property to "9".
3. Switch to XAML view and customize **C1MaskedTextBox1** by adding `Watermark="Name"` to the `<c1:C1MaskedTextBox>` tag so it appears similar to the following:
   ```
   <c1:C1MaskedTextBox Height="23" Margin="21,46,167,0"
   Name="C1MaskedTextBox1" VerticalAlignment="Top" Watermark="Name" />
   ```
   This will add a watermark to the control.
4. Switch to XAML view and customize **C1MaskedTextBox2** by adding `Watermark="Employee ID" Mask="000-00-0000"` to the `<c1:C1MaskedTextBox>` tag so it appears similar to the following:
   ```
   <c1:C1MaskedTextBox Margin="14,98,12,0" Name="C1MaskedTextBox2"
   Height="23" VerticalAlignment="Top" Watermark="Employee ID" Mask="000-
   00-0000" />
   ```
   This will add a watermark and mask to the control.
5. Switch to XAML view and customize **C1MaskedTextBox3** by adding `Watermark="Hire Date" Mask="00/00/0000"` to the `<c1:C1MaskedTextBox>` tag so it appears similar to the following:
   ```
   <c1:C1MaskedTextBox Height="23" Margin="14,0,12,87"
   Name="C1MaskedTextBox3" VerticalAlignment="Bottom" Watermark="Hire
   Date" Mask="00/00/0000"/>
   ```
   This will add a watermark and mask to the control.
6. Switch to XAML view and customize **C1MaskedTextBox4** by adding `Watermark="Phone Number" Mask="(999) 000-0000"` to the `<c1:C1MaskedTextBox>` tag so it appears similar to the following:
   ```
   <my:C1MaskedTextBox Height="23" Margin="14,0,12,33"
   Name="C1MaskedTextBox4" VerticalAlignment="Bottom" Watermark="Phone
   Number" Mask="(999) 000-0000"/>
   ```
   This will add a watermark and mask to the control.

You've successfully set up your application's user interface. In the next step you'll add code to your application.

Step 3 of 4: Adding Code to the Application

In the previous steps you set up the application's user interface and added controls to your application. In this step you'll add code to your application to finalize it.

Complete the following steps:

1. In Design view, double-click **C1MaskedTextBox1** to switch to Code view and create the **C1MaskedTextBox1_TextChanged** event handler. Return to Design view and repeat this step with each

of the **C1MaskedTextBox** controls so that they each have a **C1MaskedTextBox1_TextChanged** event handler.

2. In Code view, add the following import statement to the top of the page:

   - Visual Basic

   ```
   Imports C1.WPF
   ```

   - C#

   ```
   using C1.WPF;
   ```

3. Add code to the **C1MaskedTextBox1_TextChanged** event handler so that it appears like the following:

   - Visual Basic

   ```
   Private Sub C1MaskedTextBox1_TextChanged(ByVal sender As System.Object,
   ByVal e As System.Windows.Controls.TextChangedEventArgs) Handles
   C1MaskedTextBox1.TextChanged
       Me.Label2.Content = "Mask: " & Me.C1MaskedTextBox1.Mask & "  Value:
   " & Me.C1MaskedTextBox1.Value & "  Text: " & Me.C1MaskedTextBox1.Text
   End Sub
   ```

   - C#

   ```
   private void c1MaskedTextBox1_TextChanged(object sender,
   TextChangedEventArgs e)
   {
       this.label2.Content = "Mask: " + this.c1MaskedTextBox1.Mask + "
   Value: " + this.c1MaskedTextBox1.Value + " Text: " +
   this.c1MaskedTextBox1.Text;
   }
   ```

4. Add code to the **C1MaskedTextBox2_TextChanged** event handler so that it appears like the following:

   - Visual Basic

   ```
   Private Sub C1MaskedTextBox2_TextChanged(ByVal sender As System.Object,
   ByVal e As System.Windows.Controls.TextChangedEventArgs) Handles
   C1MaskedTextBox2.TextChanged
       Me.Label3.Content = "Mask: " & Me.C1MaskedTextBox2.Mask & "  Value:
   " & Me.C1MaskedTextBox2.Value & "  Text: " & Me.C1MaskedTextBox2.Text
   End Sub
   ```

   - C#

   ```
   private void c1MaskedTextBox2_TextChanged(object sender,
   TextChangedEventArgs e)
   {
       this.label3.Content = "Mask: " + this.c1MaskedTextBox2.Mask + "
   Value: " + this.c1MaskedTextBox2.Value + " Text: " +
   this.c1MaskedTextBox2.Text;
   }
   ```

5. Add code to the **C1MaskedTextBox3_TextChanged** event handler so that it appears like the following:

   - Visual Basic

   ```
   Private Sub C1MaskedTextBox3_TextChanged(ByVal sender As System.Object,
   ByVal e As System.Windows.Controls.TextChangedEventArgs) Handles
   C1MaskedTextBox3.TextChanged
       Me.Label4.Content = "Mask: " & Me.C1MaskedTextBox3.Mask & "  Value:
   " & Me.C1MaskedTextBox3.Value & "  Text: " & Me.C1MaskedTextBox3.Text
   End Sub
   ```

   - C#

   ```
   private void c1MaskedTextBox3_TextChanged(object sender,
   TextChangedEventArgs e)
   ```

```
{
    this.label4.Content = "Mask: " + this.c1MaskedTextBox3.Mask + "
Value: " + this.c1MaskedTextBox3.Value + " Text: " +
this.c1MaskedTextBox3.Text;
}
```

6. Add code to the **C1MaskedTextBox4_TextChanged** event handler so that it appears like the following:

   - Visual Basic

     ```
     Private Sub C1MaskedTextBox4_TextChanged(ByVal sender As System.Object,
     ByVal e As System.Windows.Controls.TextChangedEventArgs) Handles
     C1MaskedTextBox4.TextChanged
         Me.Label5.Content = "Mask: " & Me.C1MaskedTextBox4.Mask & "  Value:
     " & Me.C1MaskedTextBox4.Value & "  Text: " & Me.C1MaskedTextBox4.Text
     End Sub
     ```

   - C#

     ```
     private void c1MaskedTextBox4_TextChanged(object sender,
     TextChangedEventArgs e)
     {
         this.label5.Content = "Mask: " + this.c1MaskedTextBox4.Mask + "
     Value: " + this.c1MaskedTextBox4.Value + " Text: " +
     this.c1MaskedTextBox4.Text;
     }
     ```

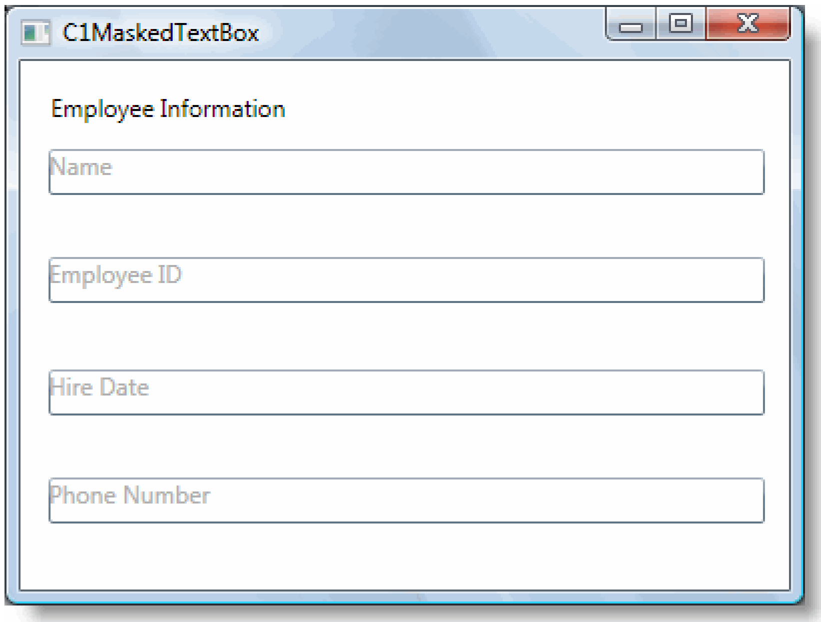
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 4 of 4: Running the Application

Now that you've created a WPF application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **MaskedTextBox for WPF**'s run-time behavior, complete the following steps:

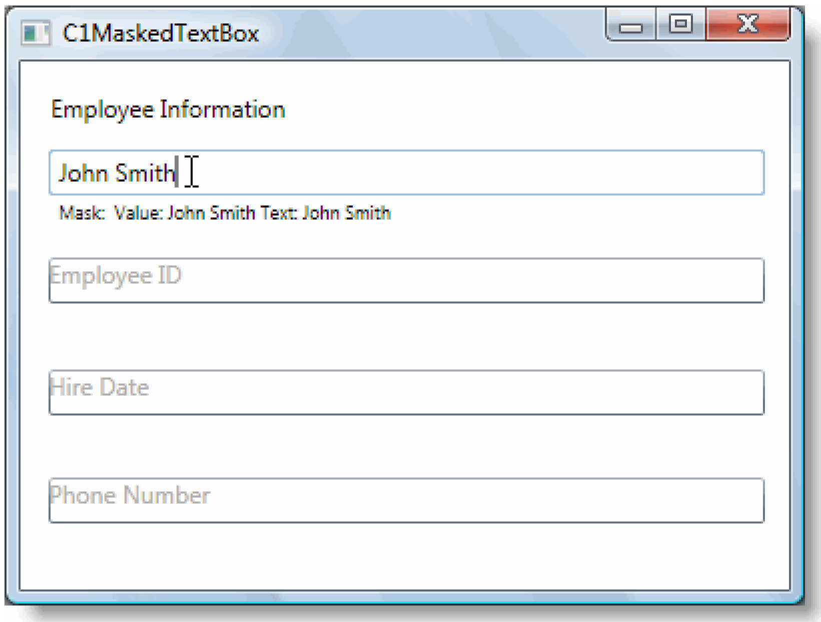
1. From the **Project** menu, select **Test Solution** to view how your application will appear at run time.

   The application will appear similar to the following:



Notice the watermark that appears in each **C1MaskedTextBox** control.

2. Enter text in the first **C1MaskedTextBox** control:

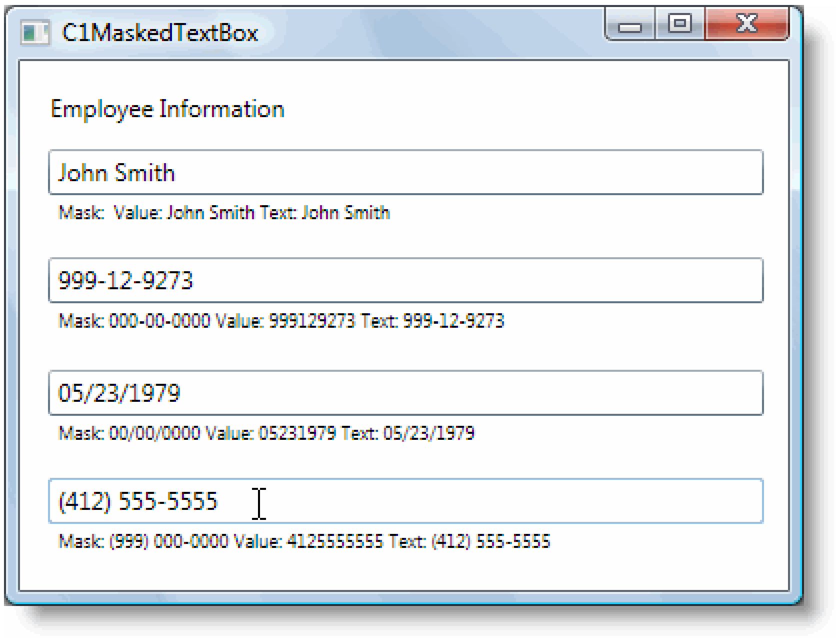


Mask: Value: John Smith Text: John Smith

The label below the control will display the mask, current value, and current text. Notice that there was no mask added to this control.

3. Try entering a string in the second **C1MaskedTextBox** control. Notice that you cannot – that is because the Mask property was set to only accept numbers. Enter a numeric value instead – notice that this does work.

4. Enter numbers in each of the remaining controls. The application will appear similar to the following image:



Employee Information

John Smith

Mask: Value: John Smith Text: John Smith

999-12-9273

Mask: 000-00-0000 Value: 999129273 Text: 999-12-9273

05/23/1979

Mask: 00/00/0000 Value: 05231979 Text: 05/23/1979

(412) 555-5555

Mask: (999) 000-0000 Value: 4125555555 Text: (412) 555-5555

Notice that the Value property displayed under each C1MaskedTextBox control does not include literal characters, while the **Text** property does.

Congratulations! You've completed the **MaskedTextBox for WPF** quick start and created a **MaskedTextBox for WPF** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

About C1MaskedTextBox

ComponentOne MaskedTextBox for WPF includes the C1MaskedTextBox control, a simple control which provides a text box with a mask that automatically validates entered input. When you add the C1MaskedTextBox control to a XAML window, it exists as a completely functional text box which you can further customize with a mask. By default, the control's interface looks similar to the following image:

The C1MaskedTextBox control appears like a text box and includes a basic text input area which can be customized.

Basic Properties

ComponentOne MaskedTextBox for WPF includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see Appearance Properties (page 12) for more information about properties that control appearance.

The following properties let you customize the C1MaskedTextBox control:

Property	Description
Mask	Gets or sets the input mask to use at run time. See Mask Formatting (page 9) for more information.
PromptChar	Gets or sets the character used to show spaces where user is supposed to type.
Text	Gets or sets the text content of this element.
TextMaskFormat	Gets or sets a value that determines whether literals and prompt characters are included in the Value property.
Value	Gets the formatted content of the control as specified by the TextMaskFormat property.
Watermark	Gets or sets the content of the watermark.

The **Text** property of the C1MaskedTextBox exposes the control's full content. The Value property exposes only the values typed by the user, excluding template characters specified in the Mask. For example, if the Mask property is set to "99-99" and the control contains the string "55-55", the **Text** property would return "55-55" and the Value property would return "5555".

Mask Formatting

You can provide input validation and format how the content displayed in the C1MaskedTextBox control will appear by setting the Mask property. **ComponentOne MaskedTextBox for WPF** supports the standard number formatting strings defined by Microsoft and the Mask property uses the same syntax as the standard **MaskedTextBox** control in WinForms. This makes it easier to re-use masks across applications and platforms.

By default, the Mask property is not set and no input mask is applied. When a mask is applied, the Mask string should consist of one or more of the masking elements. Other elements that may be displayed in the control are literals and prompts which may also be used if allowed by the TextMaskFormat property.

The following table lists some example masks:

Mask	Behavior
00/00/0000	A date (day, numeric month, year) in international date format. The "/" character is a logical date separator, and will appear to the user as the date separator appropriate to the application's current culture.
00->L<LL-0000	A date (day, month abbreviation, and year) in United States format in which the three-letter month abbreviation is displayed with an initial uppercase letter followed by two lowercase letters.
(999)-000-0000	United States phone number, area code optional. If users do not want to enter the optional characters, they can either enter spaces or place the mouse pointer directly at the position in the mask represented by the first 0.
$999,999.00	A currency value in the range of 0 to 999999. The currency, thousandth, and decimal characters will be replaced at run time with their culture-specific equivalents.

You can set the TextMaskFormat property to one of the following elements to define what is included in the mask:

Option	Description
IncludePrompt	Return text input by the user as well as any instances of the prompt character.
IncludeLiterals	Return text input by the user as well as any literal characters defined in the mask.
IncludePromptAndLiterals	Return text input by the user as well as any literal characters defined in the mask and any instances of the prompt character.
ExcludePromptAndLiterals	Return only text input by the user.

The following topics detail mask, literal, and prompt elements that can be used or displayed.

Mask Elements

ComponentOne MaskedTextBox for WPF supports the standard number formatting strings defined by Microsoft. The Mask string should consist of one or more of the masking elements as detailed in the following table:

Element	Description
0	Digit, required. This element will accept any single digit between 0 and 9.
9	Digit or space, optional.
#	Digit or space, optional. If this position is blank in the mask, it will be rendered as a space in the **Text** property. Plus (+) and minus (-) signs are allowed.
L	Letter, required. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to [a-zA-Z] in regular expressions.
?	Letter, optional. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to [a-zA-Z]? in regular expressions.
&	Character, required.
C	Character, optional. Any non-control character.
A	Alphanumeric, optional.

a	Alphanumeric, optional.
.	Decimal placeholder. The actual display character used will be the decimal symbol appropriate to the format provider.
,	Thousands placeholder. The actual display character used will be the thousands placeholder appropriate to the format provider.
:	Time separator. The actual display character used will be the time symbol appropriate to the format provider.
/	Date separator. The actual display character used will be the date symbol appropriate to the format provider.
$	Currency symbol. The actual character displayed will be the currency symbol appropriate to the format provider.
<	Shift down. Converts all characters that follow to lowercase.
>	Shift up. Converts all characters that follow to uppercase.
|	Disable a previous shift up or shift down.
\	Escape. Escapes a mask character, turning it into a literal. "\\" is the escape sequence for a backslash.
All other characters	Literals. All non-mask elements will appear as themselves within C1MaskedTextBox. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user.

The decimal (.), thousandths (,), time (:), date (/), and currency ($) symbols default to displaying those symbols as defined by the application's culture.

Literals

In addition to the mask elements defined in the Mask Formatting (page 9) topic, other characters can be included in the mask. These characters are *literals*. Literals are non-mask elements that will appear as themselves within C1MaskedTextBox. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user.

For example, if the Mask property has been set to "(999)-000-0000" to define a phone number, the mask characters include the "9" and "0" elements. The remaining characters, the dashes and parentheses, are literals. These characters will appear as they in the C1MaskedTextBox control.

Note that the TextMaskFormat property must be set to **IncludeLiterals** or **IncludePromptAndLiterals** for literals to be used. If you do not want literals to be used, set TextMaskFormat to **IncludePrompt** or **ExcludePromptAndLiterals**.

Prompts

You can choose to include prompt characters in the **C1MaskedTextBox** control. The prompt character defined that text that will appear in the control to prompt the user to enter text. The prompt character indicates to the user that text can be entered, and can be used to detail the type of text allowed. By default the underline "_" character is used.

Note that the TextMaskFormat property must be set to **IncludePrompt** or **IncludePromptAndLiterals** for prompt characters to be used. If you do not want prompt characters to be used, set TextMaskFormat to **IncludeLiterals** or **ExcludePromptAndLiterals**.

Watermark

Using the Watermark property you can provide contextual clues of what value users should enter in a C1MaskedTextBox control. The watermark is displayed in the control while not text has been entered. To add a watermark, add the text `Watermark="Watermark Text"` to the `<c1:C1MaskedTextBox>` tag in the XAML markup for any **C1MaskedTextBox** control.

So, for example, enter `Watermark="Enter Text"` to the `<c1:C1MaskedTextBox>` tag so that appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Margin="21,46,167,0"
Name="C1MaskedTextBox1" VerticalAlignment="Top" Watermark="Enter Text"
/>
```

The control will appear similar to the following at run time:

If you click within the control and enter text, you will notice that the watermark disappears.

Layout and Appearance

The following topics detail how to customize the C1MaskedTextBox control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of WPF's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

Appearance Properties

ComponentOne MaskedTextBox for WPF includes several properties that allow you to customize the appearance of the control. You can change the appearance of the text displayed in the control and customize graphic elements of the control. The following topics describe some of these appearance properties.

Content Properties

The following properties let you customize the appearance of content in the **C1MaskedTextBox** control:

Property	Description
Mask	Gets or sets the input mask to use at run time. See Mask Formatting (page 9) for more information.
PromptChar	Gets or sets the character used to show spaces where user is supposed to type.
Watermark	Gets or sets the content of the watermark.

Text Properties

The following properties let you customize the appearance of text in the **C1MaskedTextBox** control:

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.

FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.
TextAlignment	Gets or sets how the text should be aligned in the **C1MaskedTextBox**.

Color Properties

The following properties let you customize the colors used in the control itself:

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Border Properties

The following properties let you customize the control's border:

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1MaskedTextBox** control:

Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

ComponentOne ClearStyle Technology

ComponentOne ClearStyle™ technology is a new, quick and easy approach to providing Silverlight and WPF control styling. ClearStyle allows you to create a custom style for a control without having to deal with the hassle of XAML templates and style resources.

Currently, to add a theme to all standard WPF controls, you must create a style resource template. In Microsoft Visual Studio this process can be difficult; this is why Microsoft introduced Expression Blend to make the task a bit easier. Having to jump between two environments can be a bit challenging to developers who are not familiar with Blend or do not have the time to learn it. You could hire a designer, but that can complicate things when your designer and your developers are sharing XAML files.

That's where ClearStyle comes in. With ClearStyle the styling capabilities are brought to you in Visual Studio in the most intuitive manner possible. In most situations you just want to make simple styling changes to the controls in your application so this process should be simple. For example, if you just want to change the row color of your data grid this should be as simple as setting one property. You shouldn't have to create a full and complicated-looking template just to simply change a few colors.

How ClearStyle Works

Each key piece of the control's style is surfaced as a simple color property. This leads to a unique set of style properties for each control. For example, a **Gauge** has **PointerFill** and **PointerStroke** properties, whereas a **DataGrid** has **SelectedBrush** and **MouseOverBrush** for rows.

Let's say you have a control on your form that does not support ClearStyle. You can take the XAML resource created by ClearStyle and use it to help mold other controls on your form to match (such as grabbing exact colors). Or let's say you'd like to override part of a style set with ClearStyle (such as your own custom scrollbar). This is also possible because ClearStyle can be extended and you can override the style where desired.

ClearStyle is intended to be a solution to quick and easy style modification but you're still free to do it the old fashioned way with ComponentOne's controls to get the exact style needed. ClearStyle does not interfere with those less common situations where a full custom design is required.

ClearStyle Properties

The following table lists all of the ClearStyle-supported properties in the C1MaskedTextBox control as well as a description of the property:

Property	**Description**
Background	Gets or sets a brush that describes the background of a control. The default **Background** color is White.
FocusBrush	A brush used to define the appearance of the control, when the control is in focus.
MouseOverBrush	A brush used to define the appearance of the control, when the control is in moused over.
SelectionBackground	A brush used to define the background appearance of the control, when the control is selected.
SelectionForeground	A brush used to define the background appearance of the control, when the control is selected.

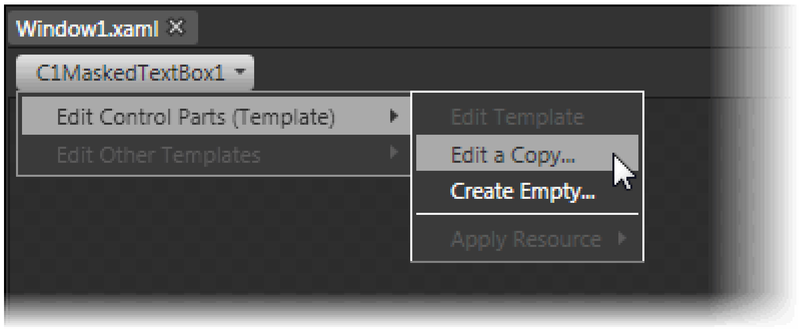
Templates

One of the main advantages to using a WPF control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for WPF applications, you can provide your own UI for data managed by **ComponentOne MaskedTextBox for WPF**. Extensible Application Markup

Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1MaskedTextBox control and, in the menu, selecting **Edit Control Parts (Templates)**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.



Note: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

Note that you can use the Template property to customize the template.

XAML Elements

Several auxiliary XAML elements are installed with **ComponentOne MaskedTextBox for WPF**. These elements include templates and themes and are located in the **MaskedTextBox for WPF** installation directory.

Included Auxiliary XAML Elements

The following auxiliary XAML element is included with **MaskedTextBox for WPF**:

Element	Folder	Description
generic.xaml	XAML	Specifies the templates for different styles and the initial style of the control.

You can incorporate elements from this file into your project, for example, to create your own theme based on the default theme.

MaskedTextBox for WPF Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with the ComponentOne Studios. Samples can be accessed from the **ComponentOne Studio for WPF ControlExplorer**. To view samples, on your desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for WPF | Samples | WPF ControlExplorer**.

C# Samples

The following C# sample is included:

Sample	**Description**
ControlExplorer	The **MaskedTextBox** page in the **ControlExplorer** sample demonstrates how to add content to and customize the C1MaskedTextBox control.

MaskedTextBox for WPF Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1MaskedTextBox control in general. If you are unfamiliar with the **ComponentOne MaskedTextBox for WPF** product, please see the MaskedTextBox for WPF Quick Start (page 3) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne MaskedTextBox for WPF** product.

Each task-based help topic also assumes that you have created a new WPF project.

Setting the Value

The Value property determines the currently visible text. By default the C1MaskedTextBox control starts with its Value not set but you can customize this at design time, in XAML, and in code.

At Design Time

To set the Value property at run time, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties window, and enter a number, for example "123", in the text box next to the Value property.

   This will have set the Value property to the number you chose.

In XAML

For example, to set the Value property add `Value="123"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1MaskedTextBox1" VerticalAlignment="Top"
Width="120" Value="123"></c1:C1MaskedTextBox>
```

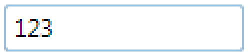
In Code

For example, to set the Value property add the following code to your project:

- Visual Basic
```
C1MaskedTextBox1.Value = "123"
```
- C#
```
c1MaskedTextBox1.Value = "123";
```

Run your project and observe:

Initially **123** (or the number you chose) will appear in the control:


123

Adding a Mask for Currency

You can easily add a mask for currency values using the Mask property. By default the C1MaskedTextBox control starts with its Mask not set but you can customize this at design time, in XAML, and in code. For more details about mask characters, see Mask Elements (page 10).

At Design Time

To set the Mask property at run time, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties window and enter "$999,999.00" in the text box next to the Mask property.

   This will have set the Mask property to the number you chose.

In XAML

For example, to set the Mask property add `Mask="$999,999.00"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:
```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1MaskedTextBox1" VerticalAlignment="Top"
Width="120" Mask="$999,999.00"></c1:C1MaskedTextBox>
```

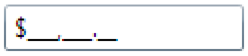
In Code

For example, to set the Value property add the following code to your project:

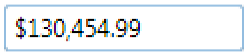
- Visual Basic
```
C1MaskedTextBox1.Mask = "$999,999.00"
```
- C#
```
c1MaskedTextBox1.Mask = "$999,999.00";
```

Run your project and observe:

The mask will appear in the control:


$___,___.__

Enter a number; notice that the mask is filled:


$130,454.99

Changing the Prompt Character

The PromptChar property sets the characters that are used to prompt users in the C1MaskedTextBox control. By default the PromptChar property is set to an underline character ("_") but you can customize this at design time, in XAML, and in code. For more details about the PromptChar property, see Prompts (page 11).

At Design Time

To set the PromptChar property at run time, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties window and enter "0000" in the text box next to the Mask property to set a mask.
3. In the Properties window, enter "#" (the pound character) in the text box next to the PromptChar property

In XAML

For example, to set the PromptChar property add `Mask="0000" PromptChar="#"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1MaskedTextBox1" VerticalAlignment="Top"
Width="120" Mask="0000" PromptChar="#"></c1:C1MaskedTextBox>
```

In Code

For example, to set the PromptChar property add the following code to your project:

- Visual Basic

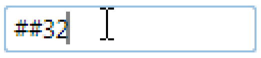
```
Dim x As Char = "#"c
C1MaskedTextBox1.Mask = "0000"
C1MaskedTextBox1.PromptChar = x
```

- C#

```
char x = '#';
this.c1MaskedTextBox1.Mask = "0000";
this.c1MaskedTextBox1.PromptChar = x;
```

Run your project and observe:

The pound character will appear as the prompt in the control. In the following image, the number 32 was entered in the control:



```
##32
```

Changing Font Type and Size

You can change the appearance of the text in the grid by using the text properties in the C1MaskedTextBox Properties window, in XAML, or in code.

At Design Time

To change the font of the grid to Arial 10pt in the Properties window at design time, complete the following:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties window, and set **FontFamily** property to "Arial".
3. In the Properties window, set the **FontSize** property to **10**.

   This will have set the control's font size and style.

In XAML

For example, to change the font of the control to Arial 10pt in XAML add `FontFamily="Arial" FontSize="10"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1MaskedTextBox1" VerticalAlignment="Top"
Width="120" FontSize="10" FontFamily="Arial"></c1:C1MaskedTextBox>
```

In Code

For example, to change the font of the grid to Arial 10pt add the following code to your project:

- Visual Basic

```
C1MaskedTextBox1.FontSize = 10
C1MaskedTextBox1.FontFamily = New
System.Windows.Media.FontFamily("Arial")
```

- C#

```
c1MaskedTextBox1.FontSize = 10;
c1MaskedTextBox1.FontFamily = new
System.Windows.Media.FontFamily("Arial");
```

Run your project and observe:

The control's content will appear in Arial 10pt font:

123

Locking the Control from Editing

By default the C1MaskedTextBox control's Value property is editable by users at run time. If you want to lock the control from being edited, you can set the **IsReadOnly** property to **True**.

At Design Time

To lock the C1MaskedTextBox control from run-time editing, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties window, and check the **IsReadOnly** check box.

   This will have set the **IsReadOnly** property to **False**.

In XAML

To lock the C1MaskedTextBox control from run-time editing in XAML, add `IsReadOnly="True"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1MaskedTextBox1" VerticalAlignment="Top"
Width="120" IsReadOnly="True"></c1:C1MaskedTextBox>
```

In Code

To lock the C1MaskedTextBox control from run-time editing, add the following code to your project:

- Visual Basic

```
C1MaskedTextBox1.IsReadOnly = True
```

- C#

```
c1MaskedTextBox1.IsReadOnly = true;
```

Run your project and observe:

The control is has been locked from editing. Try to click the cursor within the control – notice that the text insertion point (the blinking vertical line) will not appear in the control.