
ComponentOne

MaskedTextBox for Silverlight

Copyright © 1987-2011 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

MaskedTextBox	1
MaskedTextBox for Silverlight Features	1
MaskedTextBox for Silverlight Quick Start	1
Step 1 of 4: Setting up the Application	2
Step 2 of 4: Customizing the Application	2
Step 3 of 4: Adding Code to the Application	4
Step 4 of 4: Running the Application	6
Working With MaskedTextBox	7
Basic Properties	8
Mask Formatting	8
Mask Elements	9
Literals	10
Prompts.....	10
Watermark	10
MaskedTextBox for Silverlight Layout and Appearance	11
MaskedTextBox for Silverlight Appearance Properties.....	11
Content Properties	11
Text Properties.....	11
Color Properties.....	12
Border Properties	12
Size Properties	12
Templates.....	12
MaskedTextBox for Silverlight Task-Based Help	13
Setting the Value	13
Adding a Mask for Currency.....	14
Changing the Prompt Character	15
Changing Font Type and Size	15
Locking the Control from Editing.....	16

MaskedTextBox

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



Getting Started

Get started with the following topics:

- [Key Features](#) (page 1)

- [Quick Start](#) (page 1)

- [Task-Based Help](#) (page 13)

MaskedTextBox for Silverlight Features

ComponentOne MaskedTextBox for Silverlight allows you to create customized, rich applications. Make the most of **MaskedTextBox for Silverlight** 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.

- **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.

MaskedTextBox for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **MaskedTextBox for Silverlight**. In this quick start you'll start in Visual Studio and create a new project, add **MaskedTextBox for Silverlight** 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 Silverlight**.

Step 1 of 4: Setting up the Application

In this step you'll begin in Visual Studio to create a Silverlight application using **MaskedTextBox for Silverlight**. 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. In Visual Studio 2008, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Silverlight Application**. Enter a **Name** for your project and click **OK**. The **New Silverlight Application** dialog box will appear.
3. Click **OK** to close the **New Silverlight Application** dialog box and create your project.
4. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once. Note that you cannot currently add Silverlight controls directly to the design area in Visual Studio, so you must add them to the XAML window as directed in the next step.
5. Navigate to the Toolbox and double-click the **StackPanel** icon to add the panel to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight"
x:Class="C1MaskedTextBox.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Width="400" Height="300">
  <Grid x:Name="LayoutRoot" Background="White">
<StackPanel></StackPanel>
  </Grid>
```

6. Add `x:Name="sp1" Width="Auto" Height="Auto" Orientation="Vertical" HorizontalAlignment="Center" VerticalAlignment="Center"` to the `<StackPanel>` tag so that it appears similar to the following:

```
<StackPanel x:Name="sp1" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center"></StackPanel>
```

Elements in the panel will now appear centered and vertically positioned.

You've successfully created a Silverlight application. In the next step you'll add and customize **Label** and **C1MaskedTextBox** controls and complete setting up the application.

Step 2 of 4: Customizing the Application

In the previous step you created a new Silverlight project and added a **StackPanel** to the application. In this step you'll continue by adding and customizing **Label** and **C1MaskedTextBox** controls.

Complete the following steps:

1. In the XAML window of the project, place the cursor between the `<StackPanel x:Name="sp1">` and `</StackPanel>` tags.
2. Navigate to the Toolbox and double-click the **Label** icon to add the control to the **StackPanel**.
3. Add `x:Name="lbl1" Content="Employee Information" Margin="2"` to the `<controls:Label>` tag so that it appears similar to the following:

```
<controls:Label x:Name="lbl1" Content="Employee Information"
Margin="2"></controls:Label>
```

- Place the cursor just after the `</controls:Label>` tag, navigate to the Toolbox, and double-click the **C1MaskedTextBox** icon to add the control to the **StackPanel**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:controls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Toolk
it" xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
x:Class="C1MaskedTextBox.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="400"
Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
        <StackPanel x:Name="spl" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center">
            <controls:Label x:Name="lbl1" Content="Employee Information"
Margin="2"></controls:Label>
            <c1:C1MaskedTextBox></c1:C1MaskedTextBox>
        </StackPanel>
    </Grid>
</UserControl>
```

Note that the `C1.Silverlight` namespace and `<c1:C1MaskedTextBox></c1:C1MaskedTextBox>` tags have been added to the project.

- Give your control a name by adding `x:Name="c1mtb1"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1"></c1:C1MaskedTextBox>
```

By giving it a unique identifier, you'll be able to access the control in code.

- Resize your control and set the alignment and margin by adding `Height="23"` `VerticalAlignment="Top"` `Margin="2"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1" Height="23" VerticalAlignment="Top"
Margin="2"></c1:C1MaskedTextBox>
```

Your control should now be resized on the page.

- Set a watermark on the control by adding `Watermark="Employee ID"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Employee ID"></c1:C1MaskedTextBox>
```

The `WaterMark` property will set the text that appears in the control to prompt users at run time.

- Set a numeric mask on the control by adding `Mask="000-00-0000"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Employee ID" Mask="000-00-
0000"></c1:C1MaskedTextBox>
```

The `Mask` property will set the mask – users will now only be able to enter numbers into the control at run time.

- Add an event handler by adding `TextChanged="c1mtb1_TextChanged"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Employee ID" Mask="000-00-0000"
TextChanged="c1mtb1_TextChanged"></c1:C1MaskedTextBox>
```

You will add the event handler code later in the tutorial.

10. Place the cursor just after the `</c1:C1MaskedTextBox>` tag and add the following XAML to add three additional `C1MaskedTextBox` and four additional `Label` controls:

```
<controls:Label x:Name="lb12" FontSize="9" Margin="2"></controls:Label>
<c1:C1MaskedTextBox Name="clmtb2" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Name"
TextChanged="clmtb2_TextChanged"></c1:C1MaskedTextBox>
<controls:Label x:Name="lb13" FontSize="9" Margin="2"></controls:Label>
<c1:C1MaskedTextBox Name="clmtb3" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Hire Date" Mask="00/00/0000"
TextChanged="clmtb3_TextChanged"></c1:C1MaskedTextBox>
<controls:Label x:Name="lb14" FontSize="9" Margin="2"></controls:Label>
<c1:C1MaskedTextBox Name="clmtb4" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Phone Numer" Mask="(999) 000-0000"
TextChanged="clmtb4_TextChanged"></c1:C1MaskedTextBox>
<controls:Label x:Name="lb15" FontSize="9" Margin="2"></controls:Label>
```

Your application should now appear similar to the following:

Employee Information

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. Select **View | Code** to switch to Code view.
2. In Code view, add the following import statement to the top of the page:
 - Visual Basic
`Imports C1.Silverlight`
 - C#
`using C1.Silverlight;`
3. Add the following `C1MaskedTextBox_TextChanged` event handlers to the project:
 - Visual Basic

```
Private Sub clmtb1_TextChanged(ByVal sender As System.Object, ByVal e As System.Windows.Controls.TextChangedEventArgs) Handles clmtb1.TextChanged
    Me.lb12.Content = "Mask: " & Me.clmtb1.Mask & " Value: " & Me.clmtb1.Value & " Text: " & Me.clmtb1.Text
End Sub
Private Sub clmtb2_TextChanged(ByVal sender As System.Object, ByVal e As System.Windows.Controls.TextChangedEventArgs) Handles clmtb2.TextChanged
```

- ```

 Me.lbl3.Content = "Mask: " & Me.clmtb2.Mask & " Value: " &
Me.clmtb2.Value & " Text: " & Me.clmtb2.Text
 End Sub
 Private Sub clmtb3_TextChanged(ByVal sender As System.Object, ByVal e As
System.Windows.Controls.TextChangedEventArgs) Handles clmtb3.TextChanged
 Me.lbl4.Content = "Mask: " & Me.clmtb3.Mask & " Value: " &
Me.clmtb3.Value & " Text: " & Me.clmtb3.Text
 End Sub
 Private Sub clmtb4_TextChanged(ByVal sender As System.Object, ByVal e As
System.Windows.Controls.TextChangedEventArgs) Handles clmtb4.TextChanged
 Me.lbl5.Content = "Mask: " & Me.clmtb4.Mask & " Value: " &
Me.clmtb4.Value & " Text: " & Me.clmtb4.Text
 End Sub

```
- **C#**

```

private void clmtb1_TextChanged(object sender, TextChangedEventArgs e)
{
 this.lbl2.Content = "Mask: " + this.clmtb1.Mask + " Value: " +
this.clmtb1.Value + " Text: " + this.clmtb1.Text;
}
private void clmtb2_TextChanged(object sender, TextChangedEventArgs e)
{
 this.lbl3.Content = "Mask: " + this.clmtb2.Mask + " Value: " +
this.clmtb2.Value + " Text: " + this.clmtb2.Text;
}
private void clmtb3_TextChanged(object sender, TextChangedEventArgs e)
{
 this.lbl4.Content = "Mask: " + this.clmtb3.Mask + " Value: " +
this.clmtb3.Value + " Text: " + this.clmtb3.Text;
}
private void clmtb4_TextChanged(object sender, TextChangedEventArgs e)
{
 this.lbl5.Content = "Mask: " + this.clmtb4.Mask + " Value: " +
this.clmtb4.Value + " Text: " + this.clmtb4.Text;
}

```

4. Add code to the page's constructor so that it appears like the following:

- **Visual Basic**

```

Public Sub New()
 InitializeComponent()
 Me.clmtb1_TextChanged(Nothing, Nothing)
 Me.clmtb2_TextChanged(Nothing, Nothing)
 Me.clmtb3_TextChanged(Nothing, Nothing)
 Me.clmtb4_TextChanged(Nothing, Nothing)
End Sub

```
- **C#**

```

public Page()
{
 InitializeComponent();
 this.clmtb1_TextChanged(null, null);
 this.clmtb2_TextChanged(null, null);
 this.clmtb3_TextChanged(null, null);
 this.clmtb4_TextChanged(null, null);
}

```

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 Silverlight 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 Silverlight's** run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. It will appear similar to the following:

**Employee Information**

Employee ID  
Mask: 000-00-0000 Value: \_\_\_\_\_ Text:

Name  
Mask: Value: Text:

Hire Date  
Mask: 00/00/0000 Value: \_\_\_\_\_ Text:

Phone Number  
Mask: (999) 000-0000 Value: \_\_\_\_\_ Text:

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

2. Enter a number in the first **CIMaskedTextBox** control:

**Employee Information**

999-00-0000|  
Mask: 000-00-0000 Value: 999000000 Text: 999-00-0000

Name  
Mask: Value: Text:

Hire Date  
Mask: 00/00/0000 Value: \_\_\_\_\_ Text:

Phone Number  
Mask: (999) 000-0000 Value: \_\_\_\_\_ Text:

The label below the control displays the mask, current value, and current text.

3. Enter a string in the second **CIMaskedTextBox** control:

Employee Information

999-00-0000  
Mask: 000-00-0000 Value: 999000000 Text: 999-00-0000

John Smith  
Mask: Value: John Smith Text: John Smith

Hire Date  
Mask: 00/00/0000 Value: \_\_\_\_\_ Text: \_\_\_\_\_

Phone Number  
Mask: (999) 000-0000 Value: \_\_\_\_\_ Text: \_\_\_\_\_

Notice that there was no mask added to this control – if you chose, you could type numbers or other characters in the control.

4. Try entering a string in the third **CIMaskedTextBox** 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.
5. Enter numbers in each of the remaining control. The application will appear similar to the following image:

Employee Information

999-00-0000  
Mask: 000-00-0000 Value: 999000000 Text: 999-00-0000

John Smith  
Mask: Value: John Smith Text: John Smith

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 CIMaskedTextBox control does not include literal characters, while the Text property does.

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

## Working With MaskedTextBox

**ComponentOne MaskedTextBox for Silverlight** includes the CIMaskedTextBox control, a simple control which provides a text box with a mask that automatically validates entered input. When you add the CIMaskedTextBox 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 Silverlight** 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 [MaskedTextBox for Silverlight Appearance Properties](#) (page 11) 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 <a href="#">Mask Formatting</a> (page 8) 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 Silverlight** 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 Silverlight** 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 <code>Text</code> 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 <code>[a-zA-Z]</code> in regular expressions.                         |
| ?       | Letter, optional. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to <code>[a-zA-Z]?</code> 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 8) 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**.

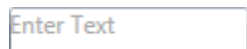
## Watermark

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" Width="120" Name="C1MaskedTextBox1"
 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.

# MaskedTextBox for Silverlight 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 Silverlight's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

## MaskedTextBox for Silverlight Appearance Properties

**ComponentOne MaskedTextBox for Silverlight** 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 <a href="#">Mask Formatting</a> (page 8) 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                                                                                                    |
|-----------------------------|----------------------------------------------------------------------------------------------------------------|
| <a href="#">FontFamily</a>  | Gets or sets the font family of the control. This is a dependency property.                                    |
| <a href="#">FontSize</a>    | Gets or sets the font size. This is a dependency property.                                                     |
| <a href="#">FontStretch</a> | Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property. |
| <a href="#">FontStyle</a>   | Gets or sets the font style. This is a dependency property.                                                    |

|                               |                                                                                            |
|-------------------------------|--------------------------------------------------------------------------------------------|
| <a href="#">FontWeight</a>    | Gets or sets the weight or thickness of the specified font. This is a dependency property. |
| <a href="#">TextAlignment</a> | Gets or sets how the text should be aligned in the <b>C1MaskedTextBox</b> .                |

## Color Properties

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

| Property                   | Description                                                                                     |
|----------------------------|-------------------------------------------------------------------------------------------------|
| <a href="#">Background</a> | Gets or sets a brush that describes the background of a control. This is a dependency property. |
| <a href="#">Foreground</a> | 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                                                                                            |
|---------------------------------|--------------------------------------------------------------------------------------------------------|
| <a href="#">BorderBrush</a>     | Gets or sets a brush that describes the border background of a control. This is a dependency property. |
| <a href="#">BorderThickness</a> | 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                                                                               |
|---------------------------|-------------------------------------------------------------------------------------------|
| <a href="#">Height</a>    | Gets or sets the suggested height of the element. This is a dependency property.          |
| <a href="#">MaxHeight</a> | Gets or sets the maximum height constraint of the element. This is a dependency property. |
| <a href="#">MaxWidth</a>  | Gets or sets the maximum width constraint of the element. This is a dependency property.  |
| <a href="#">MinHeight</a> | Gets or sets the minimum height constraint of the element. This is a dependency property. |
| <a href="#">MinWidth</a>  | Gets or sets the minimum width constraint of the element. This is a dependency property.  |
| <a href="#">Width</a>     | Gets or sets the width of the element. This is a dependency property.                     |

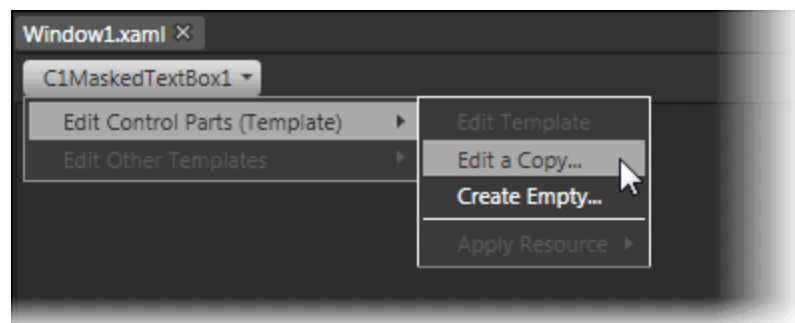
## Templates

One of the main advantages to using a Silverlight 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 Silverlight applications, you can provide your own UI for data managed by **ComponentOne MaskedTextBox for Silverlight**. 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.

# MaskedTextBox for Silverlight 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 Silverlight** product, please see the [MaskedTextBox for Silverlight Quick Start](#) (page 1) first.

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

Each task-based help topic also assumes that you have created a new Silverlight 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 Microsoft Expression Blend, in XAML, and in code.

### At Design Time in Blend

To set the `Value` property in Blend, complete the following steps:

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

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

## In XAML

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" Width="120" Name="C1MaskedTextBox1"
Value="123"></c1:C1MaskedTextBox>
```

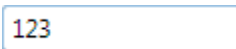
## In Code

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:



## 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 Microsoft Expression Blend, in XAML, and in code. For more details about mask characters, see [Mask Elements](#) (page 9).

### At Design Time in Blend

To set the Mask property in Blend, complete the following steps:

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

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

## In XAML

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" Width="120" Name="C1MaskedTextBox1"
Mask="$999,999.00"></c1:C1MaskedTextBox>
```

## In Code

To set the Value property add the following code to your project:

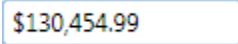
- Visual Basic  
`C1MaskedTextBox1.Mask = "$999,999.00"`
- C#  
`c1MaskedTextBox1.Mask = "$999,999.00";`

## Run your project and observe:

The mask will appear in the control:



Enter a number; notice that the mask is filled:



## 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 Microsoft Expression Blend, in XAML, and in code. For more details about the PromptChar property, see [Prompts](#) (page 10).

### At Design Time in Blend

To set the PromptChar property at in Blend, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab 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

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" Name="C1MaskedTextBox1" Width="120"
 Mask="0000" PromptChar="#"></c1:C1MaskedTextBox>
```

### In Code

To set the PromptChar property add the following code to your project:

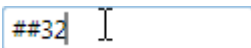
- Visual Basic

```
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:



## Changing Font Type and Size

You can change the appearance of the text in the grid by using the text properties in Microsoft Expression Blend's Properties tab, through XAML, or through code.

### At Design Time in Blend

To change the font of the grid to Arial 10pt in Blend, complete the following:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab, and set **FontFamily** property to "Arial" (or a font of your choice).
3. In the Properties window, set the **FontSize** property to **10**.

This will 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" Name="C1MaskedTextBox1" Width="120"
 FontSize="10" FontFamily="Arial"></c1:C1MaskedTextBox>
```

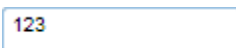
### In Code

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:



## 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 in Blend

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 tab and check the **IsReadOnly** check box.

This will 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" Name="C1MaskedTextBox1" 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 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.

123