
ComponentOne

NumericBox 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

NumericBox	1
NumericBox for Silverlight Features	1
NumericBox for Silverlight Quick Start	1
Step 1 of 4: Adding NumericBox for Silverlight to your Project	2
Step 2 of 4: Customizing the Application	4
Step 3 of 4: Adding Code to the Application	5
Step 4 of 4: Running the Application	8
About C1NumericBox	9
Basic Properties	9
Number Formatting	10
Input Validation.....	12
NumericBox for Silverlight Layout and Appearance	13
NumericBox for Silverlight Appearance Properties	13
Content Properties	13
Text Properties.....	13
Color Properties.....	13
Border Properties	14
Style Properties	14
Size Properties	14
Templates.....	15
NumericBox for Silverlight Task-Based Help	15
Setting the Start Value	16
Setting the Increment Value	16
Setting the Minimum and Maximum Values	17
Changing Font Type and Size	18
Hiding the Up and Down Buttons	18
Locking the Control from Editing.....	19

NumericBox

Display and edit numeric values in your Silverlight applications!

ComponentOne NumericBox™ for Silverlight provides a numeric text box control, **C1NumericBox**, which is similar to the standard Windows Forms **NumericUpDown** control and provides functionality for numeric input and editing right out of the box.

The **C1NumericBox** control contains a single numeric value that can be incremented or decremented by clicking the up or down buttons of the control. The user can also enter in a value, unless the **IsReadOnly** property is set to **True**.



Getting Started

Get started with the following topics:

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

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

- [Task-Based Help](#) (page 15)

NumericBox for Silverlight Features

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

- **Flexible Formatting**

The **Format** property enables you to use the familiar .NET format strings to display data in any way you wish.

- **Numeric Range Support**

Easily change the maximum and minimum values allowed for the editor.

- **Up/Down Buttons**

The **C1NumericBox** control includes up/down buttons to increment or decrement the value.

NumericBox for Silverlight Quick Start

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

You will create an application that includes five **C1NumericBox** controls. The controls will function as a lock and when the correct code number has been entered in each, the controls will become locked and inactive and a button will appear directing users to a Web site.

Step 1 of 4: Adding NumericBox for Silverlight to your Project

In this step you'll begin in Visual Studio to create a Silverlight application using **NumericBox for Silverlight**. When you add a **C1NumericBox** control to your application, you'll have a complete, functional numeric editor. You can further customize the control to your application.

To set up your project and add a **C1NumericBox** control 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:cl="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight"
x:Class="C1NumericBox.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.
7. In the XAML window of the project, place the cursor between the `<StackPanel>` and `</StackPanel>` tags.
8. Navigate to the Toolbox and double-click the **StackPanel** icon to add the panel to the existing **StackPanel**.
9. Add `x:Name="sp2" Width="Auto" Height="Auto" Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center"` to the `<StackPanel>` tag so that it appears similar to the following:

```
<StackPanel x:Name="sp2" Width="Auto" Height="Auto"
Orientation="Horizontal" HorizontalAlignment="Center"
VerticalAlignment="Center"></StackPanel>
```

Elements in the panel will now appear centered and horizontally positioned.
10. In the XAML window of the project, place the cursor between the `<StackPanel x:Name="sp2">` and `</StackPanel>` tags.
11. Navigate to the Toolbox and double-click the **C1NumericBox** 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="C1NumericBox.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="sp1" Width="Auto" Height="Auto"
  Orientation="Vertical" HorizontalAlignment="Center"
  VerticalAlignment="Center">
    <StackPanel x:Name="sp2" Width="Auto" Height="Auto"
  Orientation="Horizontal" HorizontalAlignment="Center"
  VerticalAlignment="Center">
      <c1:C1NumericBox></c1:C1NumericBox>
    </StackPanel>
  </StackPanel>
</Grid>
</UserControl>

```

Note that the C1.Silverlight namespace and `<c1:C1NumericBox></c1:C1NumericBox>` tags have been added to the project.

12. Give your control a name by adding `x:Name="c1nb1"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="c1nb1">
```

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

13. Resize your control and add a margin by adding `Width="40" Height="25" Margin="2"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="c1nb1" Width="40" Height="25" Margin="2">
```

Your control should now appear smaller on the page.

14. Set your control's limits by adding `Minimum="0" Maximum="9"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="c1nb1" Width="40" Height="25" Margin="2"
  Minimum="0" Maximum="9">
```

The `Minimum` and `Maximum` properties will set the minimum and maximum values that are allowed in the control. Users will not be able to enter values outside of that range providing built-in data validation.

15. Add `ValueChanged="c1nb1_ValueChanged"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="c1nb1" Width="40" Height="25" Margin="2"
  Minimum="0" Maximum="9" ValueChanged="c1nb1_ValueChanged">
```

You will add code for the `c1nb1_ValueChanged` event handler in a later step.

16. Add the following XAML just below the existing `<c1:C1NumericBox x:Name="c1nb1"></c1:C1NumericBox>` tags so that it appears similar to the following:

```

<c1:C1NumericBox x:Name="c1nb2" Width="40" Height="25" Margin="2"
  Minimum="0" Maximum="9"
  ValueChanged="c1nb2_ValueChanged"></c1:C1NumericBox>
<c1:C1NumericBox x:Name="c1nb3" Width="40" Height="25" Margin="2"
  Minimum="0" Maximum="9"
  ValueChanged="c1nb3_ValueChanged"></c1:C1NumericBox>
<c1:C1NumericBox x:Name="c1nb4" Width="40" Height="25" Margin="2"
  Minimum="0" Maximum="9"
  ValueChanged="c1nb4_ValueChanged"></c1:C1NumericBox>

```

```
<c1:C1NumericBox x:Name="c1nb5" Width="40" Height="25" Margin="2"
Minimum="0" Maximum="9"
ValueChanged="c1nb5_ValueChanged"></c1:C1NumericBox>
```

This will add four additional C1NumericBox controls, so that you have a total of five controls on the page.

Your application should now look similar to the following:



You've successfully created a Silverlight application, added **C1NumericBox** controls to the application, and customized those controls. In the next step you'll complete setting up the application.

Step 2 of 4: Customizing the Application

In the previous step you created a new Silverlight project and added five **C1NumericBox** controls to the application. In this step you'll continue by adding additional controls to customize the application.

Complete the following steps:

1. In the XAML window of the project, place the cursor just after the `<StackPanel x:Name="sp1">` tag.
2. Navigate to the Visual Studio Toolbox and double-click the standard **Label** control to it to your project.
3. Name the Label and add content to it by adding `x:Name="lbl1" Content="Enter Combination"` to the `<controls:Label>` tag so that it appears similar to the following:

```
<controls:Label x:Name="lbl1" Content="Enter Combination"
Margin="2"></controls:Label>
```
4. Place the cursor between the first `</StackPanel>` tag and the second `</StackPanel>` tag and add the following markup to create a second label:

```
<controls:Label x:Name="lbl2" Content="Invalid Combination"
Foreground="Red" Margin="2"></controls:Label>
```
5. Place the cursor between the `</controls:Label>` tag and the second `</StackPanel>` tag and add the following markup to create a hidden button:

```
<Button x:Name="btn1" Content="Enter" Height="25" Visibility="Collapsed"
Click="btn1_Click" Margin="2"></Button>
```

You will add the `btn1_Click` event handler later in code. Your application will now look similar to the following:



Your markup will now appear 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="C1NumericBox.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="sp1" Width="Auto" Height="Auto"
        Orientation="Vertical" HorizontalAlignment="Center"
        VerticalAlignment="Center">
        <controls:Label x:Name="lbl1" Content="Enter Combination"
        Margin="2"></controls:Label>
        <StackPanel x:Name="sp2" Width="Auto" Height="Auto"
        Orientation="Horizontal" HorizontalAlignment="Center"
        VerticalAlignment="Center">
            <c1:C1NumericBox x:Name="c1nb1" Width="40" Height="25"
            Minimum="0" Maximum="9" Margin="2"
            ValueChanged="c1nb1_ValueChanged"></c1:C1NumericBox>
            <c1:C1NumericBox x:Name="c1nb2" Width="40" Height="25"
            Minimum="0" Maximum="9" Margin="2"
            ValueChanged="c1nb2_ValueChanged"></c1:C1NumericBox>
            <c1:C1NumericBox x:Name="c1nb3" Width="40" Height="25"
            Minimum="0" Maximum="9" Margin="2"
            ValueChanged="c1nb3_ValueChanged"></c1:C1NumericBox>
            <c1:C1NumericBox x:Name="c1nb4" Width="40" Height="25"
            Minimum="0" Maximum="9" Margin="2"
            ValueChanged="c1nb4_ValueChanged"></c1:C1NumericBox>
            <c1:C1NumericBox x:Name="c1nb5" Width="40" Height="25"
            Minimum="0" Maximum="9" Margin="2"
            ValueChanged="c1nb5_ValueChanged"></c1:C1NumericBox>
        </StackPanel>
        <controls:Label x:Name="lbl2" Content="Invalid Combination"
        Foreground="Red" Margin="2"></controls:Label>
        <Button x:Name="btn1" Content="Enter" Visibility="Collapsed"
        Height="25" Margin="2" Click="Btn1_Click"></Button>
        </StackPanel>
        </Grid>
    </UserControl>

```

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 **C1NumberBox**, **Label**, and **Button** 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. Add the following imports statements to the top of the page:
 - Visual Basic


```
Imports C1.Silverlight
Imports System.Windows.Media
Imports System.Windows.Browser
```
 - C#


```
using C1.Silverlight;
using System.Windows.Media;
using System.Windows.Browser;
```
3. Initialize the following global variables just inside the **Page** class:

- Visual Basic


```
Dim nb1 As Integer = 5
Dim nb2 As Integer = 2
Dim nb3 As Integer = 3
Dim nb4 As Integer = 7
Dim nb5 As Integer = 9
```
- C#


```
int nb1 = 5;
int nb2 = 2;
int nb3 = 3;
int nb4 = 7;
int nb5 = 9;
```

These numbers will be used as the correct 'code' in the application. When the user enters the correct combination of numbers at run time the button will appear.

4. Add code to the **Button1_Click** event handler so that it appears like the following:

- Visual Basic


```
Private Sub btn1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles btn1.Click
    HtmlPage.PopupWindow(New Uri("http://www.componentone.com"), "new",
Nothing)
End Sub
```
- C#


```
private void btn1_Click(object sender, RoutedEventArgs e)
{
    HtmlPage.PopupWindow(New Uri("http://www.componentone.com"), "new",
null);
}
```

When the button is pressed at run time it will open the ComponentOne Web site.

5. Next add the following custom **NBValidation** event to your code:

- Visual Basic


```
Private Sub NBValidation()
    If Me.c1nb1.Value = nb1 And Me.c1nb2.Value = nb2 And Me.c1nb3.Value
= nb3 And Me.c1nb4.Value = nb4 And Me.c1nb5.Value = nb5 Then
        Me.lbl2.Foreground = New SolidColorBrush(Colors.Green)
        Me.lbl2.Content = "Combination Valid"
        Me.c1nb1.IsReadOnly = True
        Me.c1nb2.IsReadOnly = True
        Me.c1nb3.IsReadOnly = True
        Me.c1nb4.IsReadOnly = True
        Me.c1nb5.IsReadOnly = True
        Me.btn1.Visibility = Windows.Visibility.Visible
    End If
End Sub
```
- C#


```
private void NBValidation()
{
    if (this.c1nb1.Value == nb1 & this.c1nb2.Value == nb2 &
this.c1nb3.Value == nb3 & this.c1nb4.Value == nb4 & this.c1nb5.Value ==
nb5)
    {
        this.lbl2.Foreground = new SolidColorBrush(Colors.Green);
```

```

        this.lbl2.Content = "Combination Valid";
        this.clnb1.IsReadOnly = true;
        this.clnb2.IsReadOnly = true;
        this.clnb3.IsReadOnly = true;
        this.clnb4.IsReadOnly = true;
        this.clnb5.IsReadOnly = true;
        this.btn1.Visibility = Windows.Visibility.Visible;
    }
}

```

When the user enters the correct numbers (as indicated in step 3 above) the C1NumericBox controls will be set to read only and will no longer be editable, the text of the label below the controls will change to indicate the correct code has been entered, and a button will appear allowing users to enter the ComponentOne Web site.

6. Add **C1NumericBox_ValueChanged** event handlers to initialize **NBValidation**. The code will look like the following:

- Visual Basic

```

Private Sub clnb1_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
clnb1.ValueChanged
    NBValidation()
End Sub
Private Sub clnb2_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
clnb2.ValueChanged
    NBValidation()
End Sub
Private Sub clnb3_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
clnb3.ValueChanged
    NBValidation()
End Sub
Private Sub clnb4_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
clnb4.ValueChanged
    NBValidation()
End Sub
Private Sub clnb5_ValueChanged(ByVal sender As System.Object, ByVal e As
C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
clnb5.ValueChanged
    NBValidation()
End Sub

```

- C#

```

private void clnb1_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
private void clnb2_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
private void clnb3_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)

```

```

    {
        NBValidation();
    }
    private void c1nb4_ValueChanged(object sender,
    PropertyChangedEventArgs<double> e)
    {
        NBValidation();
    }
    private void c1nb5_ValueChanged(object sender,
    PropertyChangedEventArgs<double> e)
    {
        NBValidation();
    }
}

```

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

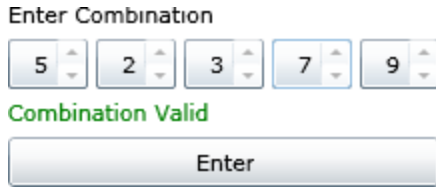
Now that you've created a Silverlight application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **NumericUpDown for Silverlight's** run-time behavior, complete the following steps:

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

The application will appear similar to the following:



2. Click the **Up** arrow in the first (left-most) **C1NumericBox** control until **5** is displayed. Note that the number increased by 1 each time you click – this is because the Increment property is set to **1** by default.
3. Click inside the second **C1NumericBox**, highlight the "0" value, and type "2" to replace it.
4. Try clicking the **Down** button in the third **C1NumericBox** control and notice that the number does not change. This is because the Minimum property was set to **0** and so the control will not accept values less than zero. Click the **Up** button until **3** is displayed.
5. In the fourth **C1NumericBox** control, place the cursor in front of the **0** and click. Enter "5" so that "50" is displayed.
6. Click inside the last **C1NumericBox** control. Notice that the **50** inside the fourth **C1NumericBox** was reset to **9**. That's because the Maximum property was set to **9** so the control will not accept values greater than nine.
7. Enter **9** in the last **C1NumericBox** control.
8. Click the **Down** button of the fourth **C1NumericBox** control twice so **7** is displayed. Note that the text of the second Label changed and the button is now visible:

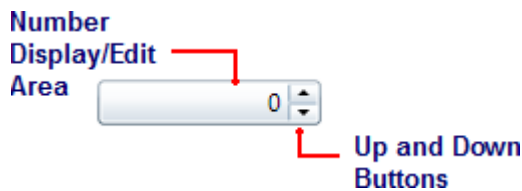


9. Try typing inside a **C1NumericBox** control or clicking its **Up** or **Down** buttons, notice that you cannot. That is because the `IsReadOnly` property was set to **True** when the correct number sequence was entered and the controls are now locked from editing.
10. Click the now-visible **Enter** button to navigate to the ComponentOne Web site.

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

About C1NumericBox

ComponentOne NumericBox for Silverlight includes the `C1NumericBox` control, a simple control which provides numeric input and editing. When you add the `C1NumericBox` control to a XAML window, it exists as a completely functional numeric editor. By default, the control's interface looks similar to the following image:



It consists of the following elements:

- **Up and Down Buttons**
The **Up** and **Down** buttons allow users to change the value displayed in the control. Each time a button is clicked the `Value` changes by the amount indicated by the `Increment` property (by default 1). By default the **Up** and **Down** buttons are visible; to hide the buttons set the `ShowButtons` property to **False**.
- **Number Display/Edit Area**
The current `Value` is displayed in the number display/editing area. Users can type in the box to change the `Value` property. By default users can edit this number; to lock the control from editing set `IsReadOnly` to **True**.

Basic Properties

ComponentOne NumericBox 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 [NumericBox for Silverlight Appearance Properties](#) (page 13) for more information about properties that control appearance.

The following properties let you customize the `C1NumericBox` control:

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

Value	Gets or sets the numeric value in the <code>C1NumericBox</code> .
Minimum	Gets or sets the minimum value allowed for the <code>C1NumericBox</code> control.
Maximum	Gets or sets the maximum value allowed for the <code>C1NumericBox</code> .
Increment	Gets or sets the increment applied when the user presses the up/down arrow keys or the Up or Down buttons.
Format	Gets or sets the format of the <code>C1NumericBox</code> control.

Number Formatting

You can change how the number displayed in the `C1NumericBox` control will appear by setting the `Format` property. **ComponentOne NumericBox for Silverlight** supports the standard number formatting strings defined by Microsoft. For more information, see [MSDN](#).

The `Format` string consists of a letter or a letter and number combination defining the format. By default, the `Format` property is set to "F0". The letter indicates the format type, here "F" for fixed-point, and the number indicates the number of decimal places, here none.

The following formats are available:

Format Specifier	Name	Description
C or c	Currency	The number is converted to a string that represents a currency amount. The conversion is controlled by the currency format information of the current NumberFormatInfo object. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default currency precision given by the current NumberFormatInfo object is used.
D or d	Decimal	This format is supported only for integral types. The number is converted to a string of decimal digits (0-9), prefixed by a minus sign if the number is negative. The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier. The following example formats an Int32 value with the Decimal format specifier.
E or e	Scientific (exponential)	The number is converted to a string of the form "-d.ddd...E+ddd" or "-d.ddd...e+ddd", where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The precision specifier indicates the desired number of digits after the decimal point. If the precision specifier is omitted, a default of six digits after the decimal point is used. The case of the format specifier indicates whether to prefix the exponent with an 'E' or an 'e'. The exponent always consists of a plus or minus sign and a minimum of three digits. The exponent is padded with zeros to meet this minimum, if required.
F or f	Fixed-point	The number is converted to a string of the form "-ddd.ddd..."

		<p>where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative.</p> <p>The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision is given by the NumberDecimalDigits property of the current NumberFormatInfo object.</p>
G or g	General	<p>The number is converted to the most compact of either fixed-point or scientific notation, depending on the type of the number and whether a precision specifier is present. If the precision specifier is omitted or zero, the type of the number determines the default precision, as indicated by the following list.</p> <ul style="list-style-type: none"> • Byte or SByte: 3 • Int16 or UInt16: 5 • Int32 or UInt32: 10 • Int64: 19 • UInt64: 20 • Single: 7 • Double: 15 • Decimal: 29 <p>Fixed-point notation is used if the exponent that would result from expressing the number in scientific notation is greater than -5 and less than the precision specifier; otherwise, scientific notation is used. The result contains a decimal point if required and trailing zeroes are omitted. If the precision specifier is present and the number of significant digits in the result exceeds the specified precision, then the excess trailing digits are removed by rounding.</p> <p>The exception to the preceding rule is if the number is a Decimal and the precision specifier is omitted. In that case, fixed-point notation is always used and trailing zeroes are preserved.</p> <p>If scientific notation is used, the exponent in the result is prefixed with 'E' if the format specifier is 'G', or 'e' if the format specifier is 'g'. The exponent contains a minimum of two digits. This differs from the format for scientific notation produced by the 'E' or 'e' format specifier, which includes a minimum of three digits in the exponent.</p>
N or n	Number	<p>The number is converted to a string of the form "-d,ddd,ddd.ddd...", where '-' indicates a negative number symbol if required, 'd' indicates a digit (0-9), ',' indicates a thousand separator between number groups, and '.' indicates a decimal point symbol. The actual negative number pattern, number group size, thousand separator, and decimal separator are specified by the NumberNegativePattern, NumberGroupSizes, NumberGroupSeparator, and NumberDecimalSeparator properties, respectively, of the current NumberFormatInfo object.</p> <p>The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision is given by the NumberDecimalDigits property of the current NumberFormatInfo object.</p>
P or p	Percent	<p>The number is converted to a string that represents a percent as defined by the NumberFormatInfo.PercentNegativePattern property if</p>

		the number is negative, or the NumberFormatInfo...:PercentPositivePattern property if the number is positive. The converted number is multiplied by 100 in order to be presented as a percentage. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision given by the current NumberFormatInfo object is used.
R or r	Round-trip	This format is supported only for the Single and Double types. The round-trip specifier guarantees that a numeric value converted to a string will be parsed back into the same numeric value. When a numeric value is formatted using this specifier, it is first tested using the general format, with 15 spaces of precision for a Double and 7 spaces of precision for a Single. If the value is successfully parsed back to the same numeric value, it is formatted using the general format specifier. However, if the value is not successfully parsed back to the same numeric value, then the value is formatted using 17 digits of precision for a Double and 9 digits of precision for a Single. Although a precision specifier can be present, it is ignored. Round trips are given precedence over precision when using this specifier.
X or x	Hexadecimal	This format is supported only for integral types. The number is converted to a string of hexadecimal digits. The case of the format specifier indicates whether to use uppercase or lowercase characters for the hexadecimal digits greater than 9. For example, use 'X' to produce "ABCDEF", and 'x' to produce "abcdef". The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier.
Any other single character	(Unknown specifier)	(An unknown specifier throws a FormatException at runtime.)

Input Validation

You can use the `Minimum` and `Maximum` properties to set a numeric range that users are limited to at run time. If the `Minimum` and `Maximum` properties are set, users will not be able to pick a number larger than the `Minimum` or smaller than the `Maximum`.

When setting the `Minimum` and `Maximum` properties, the `Minimum` should be smaller than the `Maximum`. Also be sure to set the `Value` property to a number within the `Minimum` and `Maximum` range.

You can also choose a mode for range validation using the `RangeValidationMode` property. This property controls when the entered number is validated. You can set `RangeValidationMode` to one of the following options:

Option	Description
Always	This mode does not allow users to enter out of range values.
AlwaysTruncate	This mode does not allow users to enter out of range values. The value will be truncated if the limits are exceeded.
OnLostFocus	This mode truncates the value when the control loses focus.

NumericBox for Silverlight Layout and Appearance

The following topics detail how to customize the `C1NumericBox` 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 grid and to customize grid actions.

NumericBox for Silverlight Appearance Properties

ComponentOne NumericBox 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 **`C1NumericBox`** control:

Property	Description
Format	Gets or sets the value for the Format of the <code>C1NumericBox</code> .
Watermark	Gets or sets the watermark content displayed when the control is empty.

Text Properties

The following properties let you customize the appearance of text in the **`C1NumericBox`** 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 <code>C1NumericBox</code> .

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.
SelectionBackground	Gets or sets the brush that fills the background of the selected text.
SelectionForeground	Gets or sets the brush used for the selected text in the C1NumericBox.

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.

Style Properties

The following properties let you set styles:

Property	Description
FocusVisualStyle	Gets or sets a property that enables customization of appearance, effects, or other style characteristics that will apply to this element when it captures keyboard focus. This is a dependency property.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1NumericBox** control:

Property	Description
ActualHeight	Gets the rendered height of this element. This is a dependency property.
ActualWidth	Gets the rendered width of this element. This is a dependency property.
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.

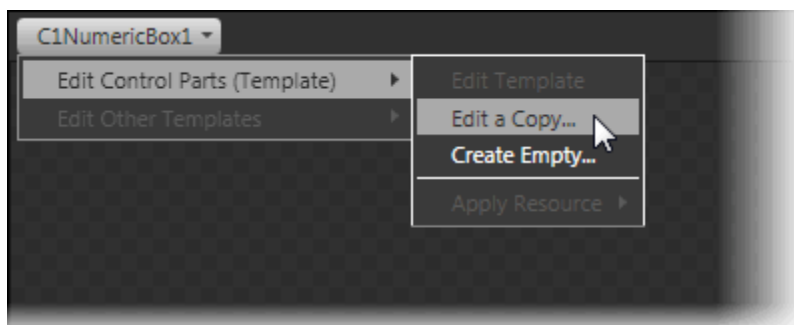
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

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 NumericBox 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 C1NumericBox 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.

NumericBox 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 C1NumericBox control in general. If you are unfamiliar with the **ComponentOne NumericBox for Silverlight** product, please see the [NumericBox for Silverlight Quick Start](#) (page 1) first.

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

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

Setting the Start Value

The Value property determines the currently selected number. By default the C1NumericBox control starts with its Value set to 0 but you can customize this number 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 C1NumericBox 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

For example, to set the Value property add `Value="123"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
Value="123"></c1:C1NumericBox>
```

In Code

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

- Visual Basic
`C1NumericBox1.Value = 123`
- C#
`c1NumericBox1.Value = 123;`

Run your project and observe:

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



Setting the Increment Value

The Increment property determines by how much the Value property changes when the **Up** or **Down** button is clicked at run time. By default the C1NumericBox control starts with its Increment set to 1 but you can customize this number at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

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

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and enter a number, for example "20", in the text box next to the Increment property.

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

In XAML

For example, to set the Increment property to **20** add `Increment="20"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
Increment="20"></c1:C1NumericBox>
```

In Code

For example, to set the Increment property to **20** add the following code to your project:

- Visual Basic
`C1NumericBox1.Increment = 20`
- C#
`c1NumericBox1.Increment = 20;`

Run your project and observe:

Click the **Up** and then the **Down** button a few times or press the **Up** and **Down** arrow keys on the keyboard. Notice that the Value changes in steps of 20. You can still edit the value directly by clicking in the text box and entering a number that falls between that step.

Setting the Minimum and Maximum Values

You can use the Minimum and Maximum properties to set a numeric range that users are limited to at run time. If the Minimum and Maximum properties are set, users will not be able to pick a number larger than the Minimum or smaller than the Maximum.

Note: When setting the Minimum and Maximum properties, the Minimum should be smaller than the Maximum. Also be sure to set the Value property to a number within the Minimum and Maximum range. In the following example, the default value **0** falls within the range chosen.

At Design Time in Blend

To set the Minimum and Maximum in Blend, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and enter a number, for example **500**, next to the Maximum property.
3. In the Properties tab, enter a number, for example **-500**, next to the Minimum property.

This will set Minimum and Maximum values.

In XAML

To set the Minimum and Maximum in XAML add `Maximum="500" Minimum="-500"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
  Maximum="500" Minimum="-500"></c1:C1NumericBox>
```

In Code

To set the Minimum and Maximum add the following code to your project:

- Visual Basic
`C1NumericBox1.Minimum = -500`
`C1NumericBox1.Maximum = 500`
- C#
`c1NumericBox1.Minimum = -500;`
`c1NumericBox1.Maximum = 500;`

Run your project and observe:

Users will be limited to the selected range at run time.

Changing Font Type and Size

You can change the appearance of the text in the grid by using the text properties in the C1NumericBox Properties tab in Blend, 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 C1NumericBox control once to select it.
2. Navigate to the Properties tab, click the **FontFamily** drop-down arrow, and set the property to "Arial" (or a font of your choice).
3. In the Properties tab, click the **FontSize** drop-down arrow and set the 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:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
  FontFamily="Arial" FontSize="10"></c1:C1NumericBox>
```

In Code

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

- Visual Basic
`C1NumericBox1.FontSize = 10`
`C1NumericBox1.FontFamily = New System.Windows.Media.FontFamily("Arial")`
- C#
`c1NumericBox1.FontSize = 10;`
`c1NumericBox1.FontFamily = new System.Windows.Media.FontFamily("Arial");`

Run your project and observe:

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



Hiding the Up and Down Buttons

By default buttons are visible in the C1NumericBox control to allow users to increment and decrement the value in the box by one step. You can choose to hide the **Up** and **Down** buttons in the C1NumericBox control at run time. To hide the **Up** and **Down** buttons you can set the ShowButtons property to **False**.

At Design Time in Blend

To hide the **Up** and **Down** buttons in Blend, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and uncheck the ShowButtons check box.

This will set the ShowButtons property to **False**.

In XAML

For example, to hide the **Up** and **Down** buttons in XAML add `ShowButtons="False"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
ShowButtons="False"></c1:C1NumericBox>
```

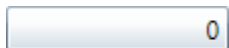
In Code

For example, to hide the **Up** and **Down** buttons add the following code to your project:

- Visual Basic
`C1NumericBox1.ShowButtons = False`
- C#
`c1NumericBox1.ShowButtons = false;`

Run your project and observe:

The **Up** and **Down** buttons will not be visible:



Locking the Control from Editing

By default the C1NumericBox 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 C1NumericBox control from run-time editing in Blend, complete the following steps:

1. Click the C1NumericBox 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

For example, to hide the **Up** and **Down** buttons in XAML add `IsReadOnly="True"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
IsReadOnly="True"></c1:C1NumericBox>
```

In Code

For example, to hide the **Up** and **Down** buttons add the following code to your project:

- Visual Basic
`C1NumericBox1.IsReadOnly = True`
- C#
`c1NumericBox1.IsReadOnly = true;`

Run your project and observe:

The control is locked from editing; notice that the **Up** and **Down** buttons are grayed out and inactive:

