ComponentOne

NumericBox for WPF

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)

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

Table of Contents

ComponentOne NumericBox for WPF Overview

Display and edit numeric values in your WPF Applications!

ComponentOne NumericBox™ for WPF 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**.

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

Getting Started

Get started with the following topics:

- Key Features (page 15)

- Quick Start (page 17)

- Task-Based Help (page 35)

Installing NumericBox for WPF

The following sections provide helpful information on installing **ComponentOne NumericBox for WPF**.

Studio for WPF Setup Files

The installation program will create the directory **C:\Program Files\ComponentOne\Studio for WPF**, which contains the following subdirectories:

Bin

Contains copies of all ComponentOne binaries (DLLs, EXEs). For **Component NumericBox for WPF**, the following DLLs are installed:

- C1.WPF.dll
- C1.WPF.Expression.Design.dll
- C1.WPF.VisualStudio.Design.dll
- C1.WPF.Expression.Design.4.dll
- C1.WPF.VisualStudio.Design.4.dll

In addition, the following files from the Microsoft WPF Toolkit are also installed:

- WPFToolkit.dll
- WPFToolkit.Design.dll
- WPFToolkit.VisualStudio.Design.dll

For more information about the Microsoft WPF Toolkit, see CodePlex. The C1.WPF.dll and WPFToolkit.dll assemblies are required for deployment.

C1WPF\XAML

Contains the full XAML definitions of C1NumericBox styles and templates which can be used for creating your own custom styles and templates.

The **ComponentOne Studio for WPF Help Setup** program installs integrated Microsoft Help 2.0 and Microsoft Help Viewer help to the **C:\Program Files\ComponentOne\Studio for WPF** directory in the following folders:

H2Help	Contains Microsoft Help 2.0 integrated documentation for all Studio components.
HelpViewer	Contains Microsoft Help Viewer Visual Studio 2010 integrated documentation for all Studio components.

Samples

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

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

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

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

Common	Contains support and data files that are used by many of the demo programs.
Studio for WPF	Contains samples for **Studio for WPF**.

Samples can be accessed from the **ComponentOne Studio for WPF ControlExplorer**. To view samples, on your desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for WPF | Control Explorer**.

Esri Maps

Esri® files are installed with **ComponentOne Studio for Silverlight**, **ComponentOne Studio for WPF**, and **ComponentOne Studio for Windows Phone** by default to the following folders:

32-bit machine : C:\Program Files\ESRI SDKs\<Platform>\<version number>

64-bit machine: C:\Program Files (x86)\ESRI SDKs\<Platform>\<version number>

Files are provided for multiple languages, including: English, German (de), Spanish (es), French (fr), Italian (it), Japanese (ja), Portuguese (pt-BR), Russian (ru) and Chinese (zh-CN).

See Using Maps Powered by Esri (page 2) or visit the Esri website at http://www.esri.com for additional information.

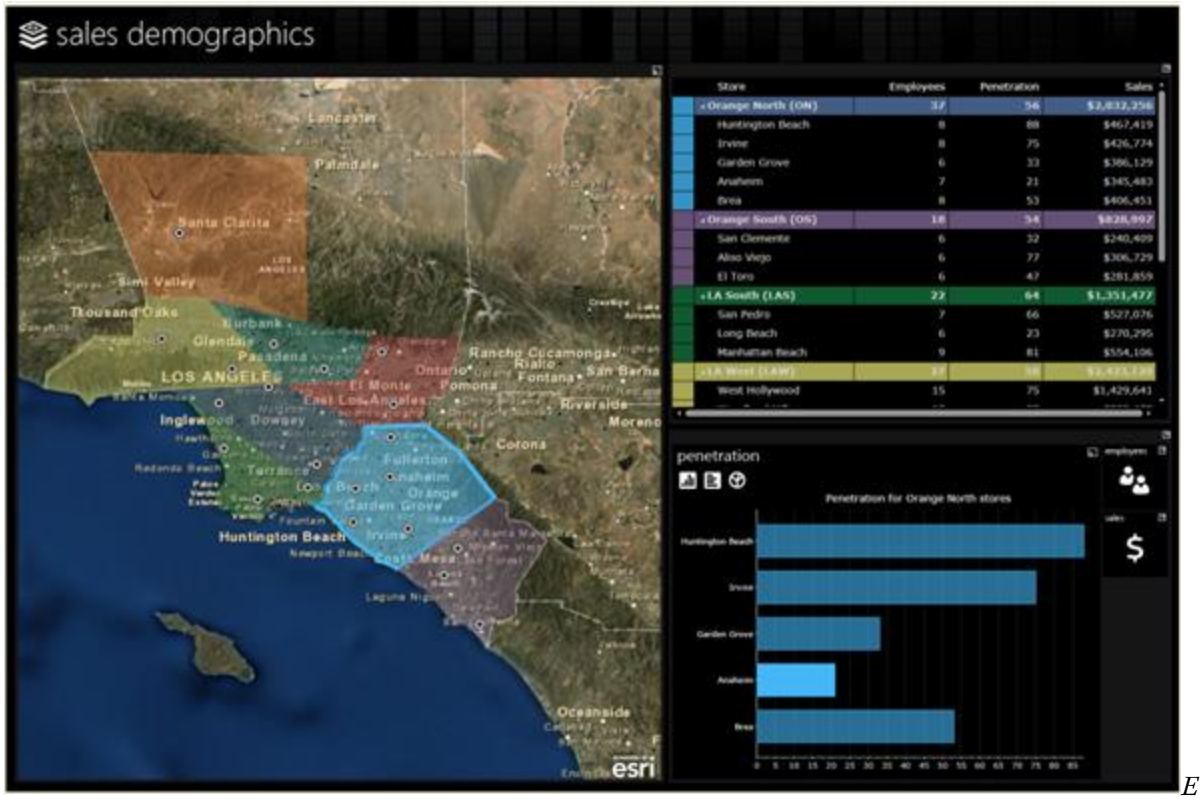
Using Maps Powered by Esri

Easily transform GIS data into business intelligence with controls for Silverlight, WPF, and Windows Phone powered by Esri® software.

By using the ComponentOne award-winning UI controls, you'll have the tools you need to seamlessly create rich, map-enabled user interfaces.

Benefits of Maps powered by Esri:

- Esri knows maps: Esri is the leading online map and GIS provider.
- Maps are technical: Using maps within your application is a very technical thing, so you don't want to take your chance using anyone but the best.
- Company of choice: Esri is the company of choice of many top companies and government agencies.
- Fulfill any developers' mapping needs: Esri mapping tools are flexible and will fill the needs of any mapping solution.



Esri Map Example

There are no additional charges for using the Esri maps included with ComponentOne products. Simply create a free online account at http://www.arcgisonline.com to start taking advantage of the Esri map controls. Esri licensing terms can be found in our Licensing Information and End User Licensing Agreement at http://www.componentone.com/SuperPages/Licensing/.

To learn more about Esri and Esri maps, please visit Esri at http://www.esri.com. There you will find detailed support, including documentation, forums, samples, and much more.

See the Studio for WPF Setup Files (page 1) topic for more information on the Esri files installed with this product.

System Requirements

System requirements include the following:

Operating Systems:	Microsoft Windows® XP with Service Pack 2 (SP2)
	Windows Vista™
	Windows 7
	Windows 2008 Server
Environments:	.NET Framework 3.5 or later
	Visual Studio® 2005 extensions for .NET Framework 2.0 November 2006 CTP
	Visual Studio® 2008 or later

Microsoft® Expression® Blend Compatibility:	**NumericBox for WPF** includes design-time support for Expression Blend.

Note: The **C1.WPF.VisualStudio.Design.dll** assembly is required by Visual Studio 2008 and the **C1.WPF.Expression.Design.dll** assembly is required by Expression Blend. The **C1.WPF.Expression.Design.dll** and **C1.WPF.VisualStudio.Design.dll** assemblies installed with **NumericBox for WPF** should always be placed in the same folder as **C1.WPF.dll**; the DLLs should NOT be placed in the Global Assembly Cache (GAC).

Installing Demonstration Versions

If you wish to try **ComponentOne NumericBox for WPF** and do not have a serial number, follow the steps through the installation wizard and use the default serial number.

The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that registered versions will stamp every application you compile so that a ComponentOne banner will not appear when your users run the applications.

Uninstalling NumericBox for WPF

To uninstall **ComponentOne Studio for WPF**:

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

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

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

End-User License Agreement

All of the ComponentOne licensing information, including the ComponentOne end-user license agreements, frequently asked licensing questions, and the ComponentOne licensing model, is available online at http://www.componentone.com/SuperPages/Licensing/.

Licensing FAQs

This section describes the main technical aspects of licensing. It may help the user to understand and resolve licensing problems he may experience when using ComponentOne .NET and ASP.NET products.

What is Licensing?

Licensing is a mechanism used to protect intellectual property by ensuring that users are authorized to use software products.

Licensing is not only used to prevent illegal distribution of software products. Many software vendors, including ComponentOne, use licensing to allow potential users to test products before they decide to purchase them.

Without licensing, this type of distribution would not be practical for the vendor or convenient for the user. Vendors would either have to distribute evaluation software with limited functionality, or shift the burden of managing software licenses to customers, who could easily forget that the software being used is an evaluation version and has not been purchased.

How does Licensing Work?

ComponentOne uses a licensing model based on the standard set by Microsoft, which works with all types of components.

> **Note:** The **Compact Framework** components use a slightly different mechanism for run-time licensing than the other ComponentOne components due to platform differences.

When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system. (Users can also enter the serial number by clicking the **License** button on the **About Box** of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog box.)

When a licensed component is added to a form or Web page, Visual Studio obtains version and licensing information from the newly created component. When queried by Visual Studio, the component looks for licensing information stored in the system and generates a run-time license and version information, which Visual Studio saves in the following two files:

- An assembly resource file which contains the actual run-time license.
- A "licenses.licx" file that contains the licensed component strong name and version information.

These files are automatically added to the project.

In WinForms and ASP.NET 1.x applications, the run-time license is stored as an embedded resource in the assembly hosting the component or control by Visual Studio. In ASP.NET 2.x applications, the run-time license may also be stored as an embedded resource in the **App_Licenses.dll** assembly, which is used to store all run-time licenses for all components directly hosted by WebForms in the application. Thus, the **App_licenses.dll** must always be deployed with the application.

The **licenses.licx** file is a simple text file that contains strong names and version information for each of the licensed components used in the application. Whenever Visual Studio is called upon to rebuild the application resources, this file is read and used as a list of components to query for run-time licenses to be embedded in the appropriate assembly resource. Note that editing or adding an appropriate line to this file can force Visual Studio to add run-time licenses of other controls as well.

Note that the **licenses.licx** file is usually not shown in the Solution Explorer; it appears if you press the **Show All Files** button in the Solution Explorer's Toolbox or, from Visual Studio's main menu, select **Show All Files** on the **Project** menu.

Later, when the component is created at run time, it obtains the run-time license from the appropriate assembly resource that was created at design time and can decide whether to simply accept the run-time license, to throw an exception and fail altogether, or to display some information reminding the user that the software has not been licensed.

All ComponentOne products are designed to display licensing information if the product is not licensed. None will throw licensing exceptions and prevent applications from running.

Common Scenarios

The following topics describe some of the licensing scenarios you may encounter.

Creating components at design time

This is the most common scenario and also the simplest: the user adds one or more controls to the form, the licensing information is stored in the **licenses.licx** file, and the component works.

Note that the mechanism is exactly the same for Windows Forms and Web Forms (ASP.NET) projects.

Creating components at run time

This is also a fairly common scenario. You do not need an instance of the component on the form, but would like to create one or more instances at run time.

In this case, the project will not contain a **licenses.licx** file (or the file will not contain an appropriate run-time license for the component) and therefore licensing will fail.

To fix this problem, add an instance of the component to a form in the project. This will create the **licenses.licx** file and things will then work as expected. (The component can be removed from the form after the **licenses.licx** file has been created).

Adding an instance of the component to a form, then removing that component, is just a simple way of adding a line with the component strong name to the **licenses.licx** file. If desired, you can do this manually using notepad or Visual Studio itself by opening the file and adding the text. When Visual Studio recreates the application resources, the component will be queried and its run-time license added to the appropriate assembly resource.

Inheriting from licensed components

If a component that inherits from a licensed component is created, the licensing information to be stored in the form is still needed. This can be done in two ways:

- Add a **LicenseProvider** attribute to the component.

  This will mark the derived component class as licensed. When the component is added to a form, Visual Studio will create and manage the **licenses.licx** file and the base class will handle the licensing process as usual. No additional work is needed. For example:

```
[LicenseProvider(typeof(LicenseProvider))]
class MyGrid: C1.Win.C1FlexGrid.C1FlexGrid
{
  // ...
}
```

- Add an instance of the base component to the form.

  This will embed the licensing information into the **licenses.licx** file as in the previous scenario and the base component will find it and use it. As before, the extra instance can be deleted after the **licenses.licx** file has been created.

Please note that ComponentOne licensing will not accept a run-time license for a derived control if the run-time license is embedded in the same assembly as the derived class definition and the assembly is a DLL. This restriction is necessary to prevent a derived control class assembly from being used in other applications without a design-time license. If you create such an assembly, you will need to take one of the actions previously described create a component at run time.

Using licensed components in console applications

When building console applications, there are no forms to add components to and therefore Visual Studio won't create a **licenses.licx** file.

In these cases, create a temporary Windows Forms application and add all the desired licensed components to a form. Then close the Windows Forms application and copy the **licenses.licx** file into the console application project.

Make sure the **licenses.licx** file is configured as an embedded resource. To do this, right-click the **licenses.licx** file in the Solution Explorer window and select **Properties**. In the Properties window, set the **Build Action** property to **Embedded Resource**.

Using licensed components in Visual C++ applications

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

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

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

Using licensed components with automated testing products

Automated testing products that load assemblies dynamically may cause them to display license dialog boxes. This is the expected behavior since the test application typically does not contain the necessary licensing information and there is no easy way to add it.

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

For example:

```
#if AUTOMATED_TESTING
   [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")]
#endif
   public class MyDerivedControl : C1LicensedControl
   {
         // ...
   }
```

Note that the **AssemblyConfiguration** string may contain additional text before or after the given string, so the **AssemblyConfiguration** attribute can be used for other purposes as well. For example:

```
[AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime,BetaVersion")]
```

THIS METHOD SHOULD ONLY BE USED UNDER THE SCENARIO DESCRIBED. It requires a design-time license to be installed on the testing machine. Distributing or installing the license on other computers is a violation of the EULA.

Troubleshooting

We try very hard to make the licensing mechanism as unobtrusive as possible, but problems may occur for a number of reasons.

Below is a description of the most common problems and their solutions.

I have a licensed version of a ComponentOne product but I still get the splash screen when I run my project.

If this happens, there may be a problem with the licenses.licx file in the project. It either doesn't exist, contains wrong information, or is not configured correctly.

First, try a full rebuild (**Rebuild All** from the Visual Studio **Build** menu). This will usually rebuild the correct licensing resources.

If that fails follow these steps:

1. Open the affected project.
2. Select an instance of the updated component.
3. In the Visual Studio Properties window, change any property. It does not matter which property you change; you can change it back to the previous value.
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**) and run the solution.

Alternative 1: Follow these steps:

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

Alternative 2: Follow these steps:

1. Open the affected project.
2. Delete the licenses.licx file from the project.
3. Add a new instance of the updated component to the form.
4. Rebuild and run the solution. The nag screen should not appear.
5. Remove the newly added component from the form.

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

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

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

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

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

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

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

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

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

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

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

Option 2 – Continue to use the components you have.

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

Technical Support

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

Some methods for obtaining technical support include:

- **Online Resources**
  ComponentOne provides customers with a comprehensive set of technical resources in the form of FAQs, samples and videos, Version Release History, searchable Knowledge base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**
  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.
- **Product Forums**
  ComponentOne's product forums are available for users to share information, tips, and techniques regarding ComponentOne products. ComponentOne developers will be available on the forums to share insider tips and technique and answer users' questions. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.
- **Installation Issues**
  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.
- **Documentation**
  Microsoft integrated ComponentOne documentation can be installed with each of our products, and documentation is also available online. If you have suggestions on how we can improve our documentation, please email the Documentation team. Please note that e-mail sent to the Documentation

team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

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

Redistributable Files

ComponentOne NumericBox for WPF is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.WPF.dll

In addition, the following file from the Microsoft WPF Toolkit is also installed and is redistributable:

- WPFToolkit.dll

Site licenses are available for groups of multiple developers. Please contact Sales@ComponentOne.com for details.

About this Documentation

You can create your applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, we will use the **Design** workspace of Blend for most examples.

Acknowledgements

ComponentOne

If you have any suggestions or ideas for new features or controls, please call us or write:

Corporate Headquarters

ComponentOne LLC

201 South Highland Avenue

3rd Floor

Pittsburgh, PA 15206 • USA

412.681.4343

412.681.4384 (Fax)

http://www.componentone.com/

ComponentOne Doc-To-Help

This documentation was produced using ComponentOne Doc-To-Help® Enterprise.

XAML and XAML Namespaces

XAML is a declarative XML-based language that is used as a user interface markup language in Windows Presentation Foundation (WPF) and the .NET Framework 3.0 or later. With XAML you can create a graphically

rich customized user interface, perform data binding, and much more. For more information on XAML, please see http://www.microsoft.com.

XAML Namespaces

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

When you create a Microsoft Expression Blend project, a XAML file is created for you and some initial namespaces are specified:

Namespace	Description
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"	This is the default Windows Presentation Foundation namespace.
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"	This is a XAML namespace that is mapped to the **x:** prefix. The **x:** prefix provides a quick, easy way to reference the namespace, which defines many commonly-used features necessary for WPF applications.

When you add a C1NumericBox control to the window in Microsoft Expression Blend or Visual Studio, **Blend** or **Visual Studio** automatically creates an XML namespace for the control. The namespace looks like the following:

```
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
```

The namespace value is **c1**. This is a unified namespace; once this is in the project, all ComponentOne WPF controls found in your references will be accessible through XAML (and IntelliSense). Note that you still need to add references to the assemblies for each control you need to use.

You can also choose to create your own custom name for the namespace. For example:

```
xmlns:MyNB="http://schemas.componentone.com/winfx/2006/xaml"
```

You can now use your custom namespace when assigning properties, methods, and events. For example, use the following XAML to add a border around the NumericBox:

```
<MyNB:C1NumericBox Name="C1NumericBox1" BorderThickness="10,10,10,10">
```

Creating a Microsoft Blend Project

To create a new Blend project, complete the following steps:

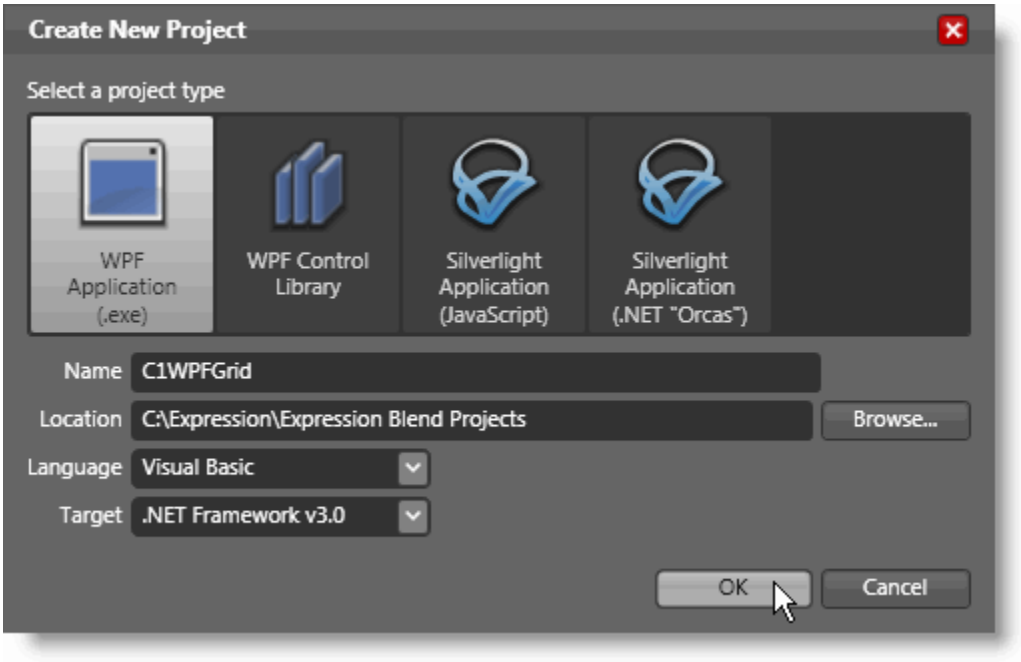
1. From the **File** menu, select **New Project** or click **New Project** in the Blend startup window.

   The **Create New Project** dialog box opens.

2. Make sure **WPF Application (.exe)** is selected and enter a name for the project in the Name text box. The **WPF Application (.exe)** creates a project for a Windows-based application that can be built and run while being designed.

3. Select the **Browse** button to specify a location for the project.

4. Select a language from the **Language** drop-down box and click **OK**.



A new Blend project with a XAML window is created.

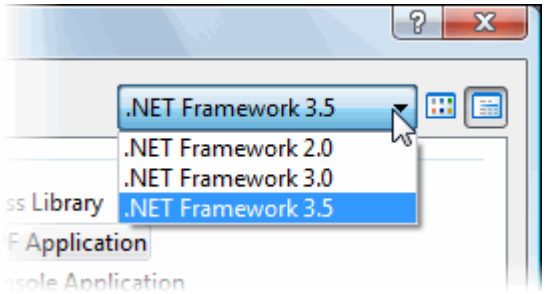
Creating a .NET Project in Visual Studio

To create a new .NET project in Visual Studio 2008, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**.

   The **New Project** dialog box opens.

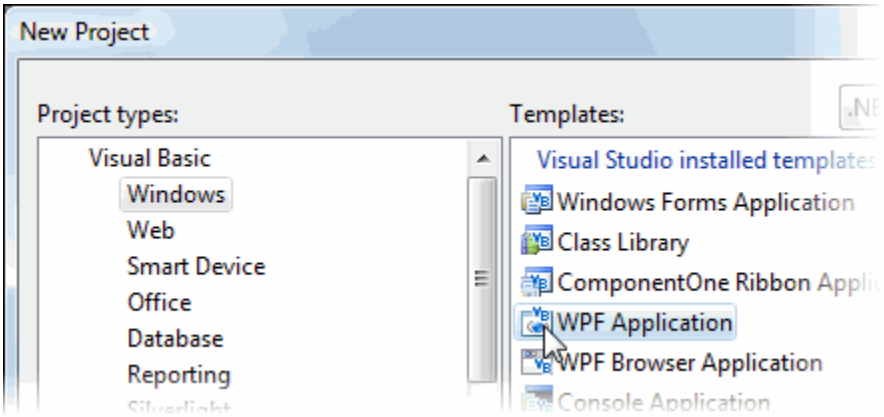
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



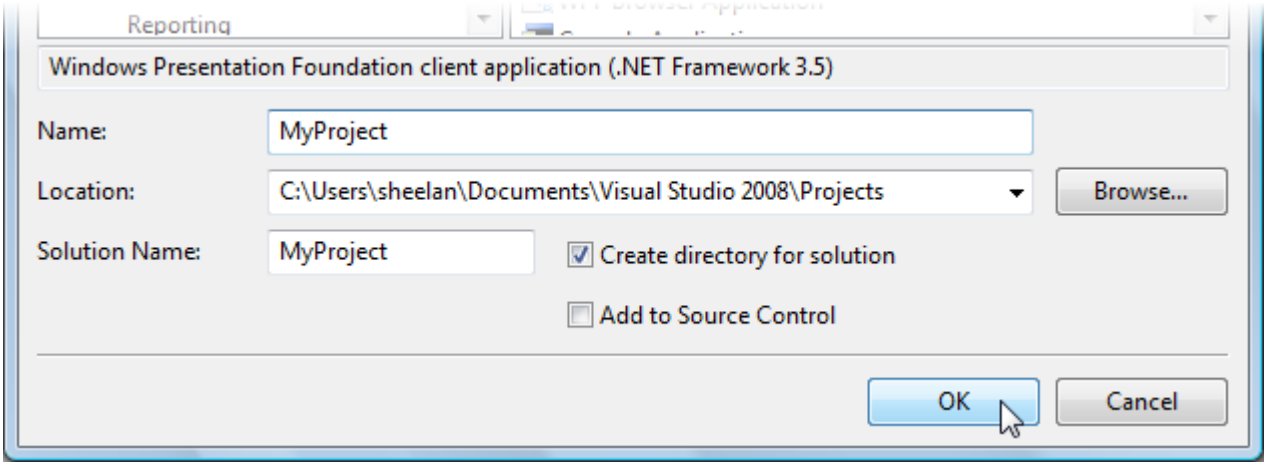
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

> **Note:** In Visual Studio 2005 select **NET Framework 3.0** under **Visual Basic** or **Visual C#** in the Project types menu.

4. Choose **WPF Application** from the list of **Templates** in the right pane.



5. Enter a name for your application in the **Name** field and click **OK**.



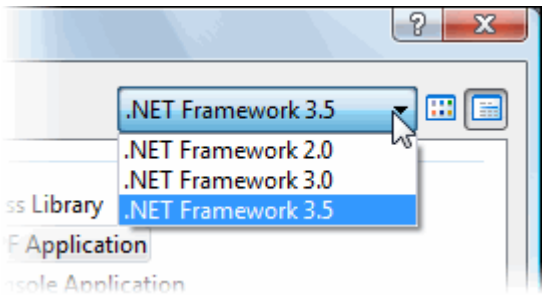
A new Microsoft Visual Studio .NET WPF project is created with a XAML file that will be used to define your user interface and commands in the application.

Note: You can create your WPF applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, Blend will be used for most examples.

Creating an XAML Browser Application (XBAP) in Visual Studio

To create a new XAML Browser Application (XBAP) in Visual Studio 2008, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**. The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



3. Under Project types, select either **Visual Basic** or **Visual C#**.
4. Choose **WPF Browser Application** from the list of **Templates** in the right pane.

> **Note:** If using Visual Studio 2005, you may need to select **XAML Browser Application (WPF)** after selecting **NET Framework 3.0** under **Visual Basic** or **Visual C#** in the left-side menu.

5. Enter a name for your application in the **Name** field and click **OK**.

   A new Microsoft Visual Studio .NET WPF Browser Application project is created with a XAML file that will be used to define your user interface and commands in the application.

Adding the NumericBox for WPF Components to a Blend Project

In order to use C1NumericBox or another **ComponentOne NumericBox for WPF** component in the Design workspace of Blend, you must first add a reference to the **C1.WPF** assembly and then add the component from Blend's **Asset Library**.

To add a reference to the assembly:

1. Select **Project | Add Reference**.
1. Browse to find the **C1.WPF.dll** assembly installed with **NumericBox for WPF**.

> **Note:** The **C1.WPF.dll** file is installed to **C:\Program Files\ComponentOne\Studio for WPF\bin** by default.

2. Select **C1.WPF.dll** and click **Open**. A reference is added to your project.

To add a component from the Asset Library:

1. Once you have added a reference to the **C1.WPF** assembly, click the **Asset Library** button  in the Blend Toolbox. The **Asset Library** appears.
2. Click the **Controls** drop-down arrow and select **All**.
3. Select **C1NumericBox**The component will appear in the Toolbox below the **Asset Library** button.
4. Double-click the **C1NumericBox** component in the Toolbox to add it to **Window1.xaml**.

Adding the NumericBox for WPF Components to a Visual Studio Project

When you install **ComponentOne NumericBox for WPF** the C1NumericBox control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

ComponentOne NumericBox for WPF provides the following control:

- C1NumericBox

To use a **NumericBox for WPF** panel or control, add it to the window or add a reference to the **C1.WPF** assembly to your project.

Manually Adding NumericBox for WPF to the Toolbox

When you install **NumericBox for WPF**, the following **NumericBox for WPF** control and panel will appear in the Visual Studio Toolbox customization dialog box:

- C1NumericBox

To manually add the C1NumericBox control to the Visual Studio Toolbox, complete the following steps:

1. Open the Visual Studio IDE (Microsoft Development Environment). Make sure the Toolbox is visible (select **Toolbox** in the **View** menu, if necessary) and right-click the Toolbox to open its context menu.
2. To make **NumericBox for WPF** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1WPFNumericBox**, for example.
3. Right-click the tab where the component is to appear and select **Choose Items** from the context menu.

   The **Choose Toolbox Items** dialog box opens.
4. In the dialog box, select the **WPF Components** tab.
5. Sort the list by Namespace (click the *Namespace* column header) and select the check boxes for components belonging to the **C1.WPF** namespace. Note that there may be more than one component for each namespace.

Adding NumericBox for WPF to the Window

To add **ComponentOne NumericBox for WPF** to a window or page, complete the following steps:

1. Add the C1NumericBox control to the Visual Studio Toolbox.
2. Double-click C1NumericBox or drag the control onto the window.

Adding a Reference to the Assembly

To add a reference to the **NumericBox for WPF** assembly, complete the following steps:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne NumericBox for WPF** assembly from the list on the **.NET** tab or on the **Browse** tab, browse to find the **C1.WPF.dll** assembly and click **OK**.
3. Double-click the window caption area to open the code window. At the top of the file, add the following **Imports** statements (**using** in C#):

   ```
   Imports C1.WPF
   ```

   This makes the objects defined in the **NumericBox for WPF** assembly visible to the project.

Key Features

ComponentOne NumericBox for WPF allows you to create customized, rich applications. Make the most of **NumericBox for WPF** 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. See the Number Formatting (page 26) topic for more information.

- **Numeric Range Support**

  Easily change the maximum and minimum values allowed for the editor. See the Input Validation (page 28) topic for more information.

- **Up/Down Buttons**

  The C1NumericBox control includes up/down buttons to increment or decrement the value. See Working with C1NumericBox (page 25) for more information.

NumericBox for WPF Quick Start

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

In this step you'll begin in Visual Studio to create a WPF application using **NumericBox for WPF**. 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.

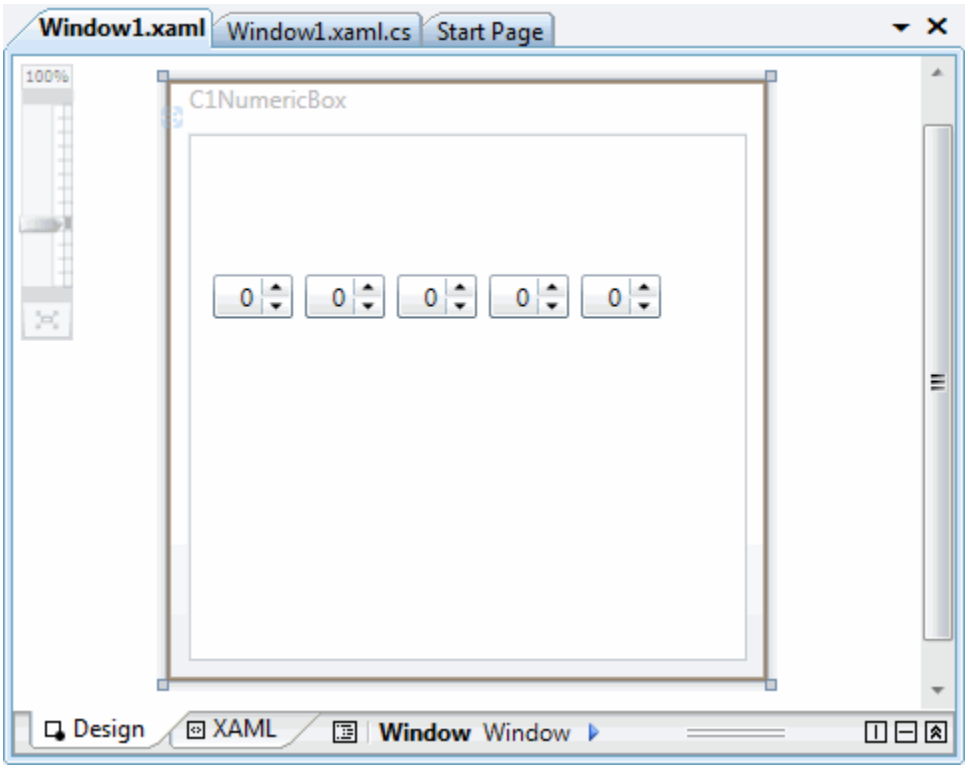
To set up your project and add a C1NumericBox control to your application, complete the following steps:

1. Create a new WPF project in Visual Studio. For more information about creating a WPF project, see Creating a .NET Project in Visual Studio (page 12).
2. Navigate to the Toolbox and double-click the **C1NumericBox** icon to add the control to Window1.
3. Click once on the **C1NumericBox1** control to select it, and navigate to the Properties window.
4. In the Properties window, set the following properties:

Property	Value
Width	40
Minimum	0
Maximum	9

   The **Width** property will resize the control. 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.
5. In the Design view, right-click the **C1NumericBox1** control and select **Copy**.
6. Right-click the window and select **Paste** to create the **C1NumericBox2** control with the same settings.
7. Repeat steps 5 and 6 three more times to create a total of five **C1NumericBox** controls.
8. In Design view, re-position each of the controls so that they appear next to each other and are numbered **C1NumericBox1** to **C1NumericBox5** from left to right.

   Your application should now look similar to the following:



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

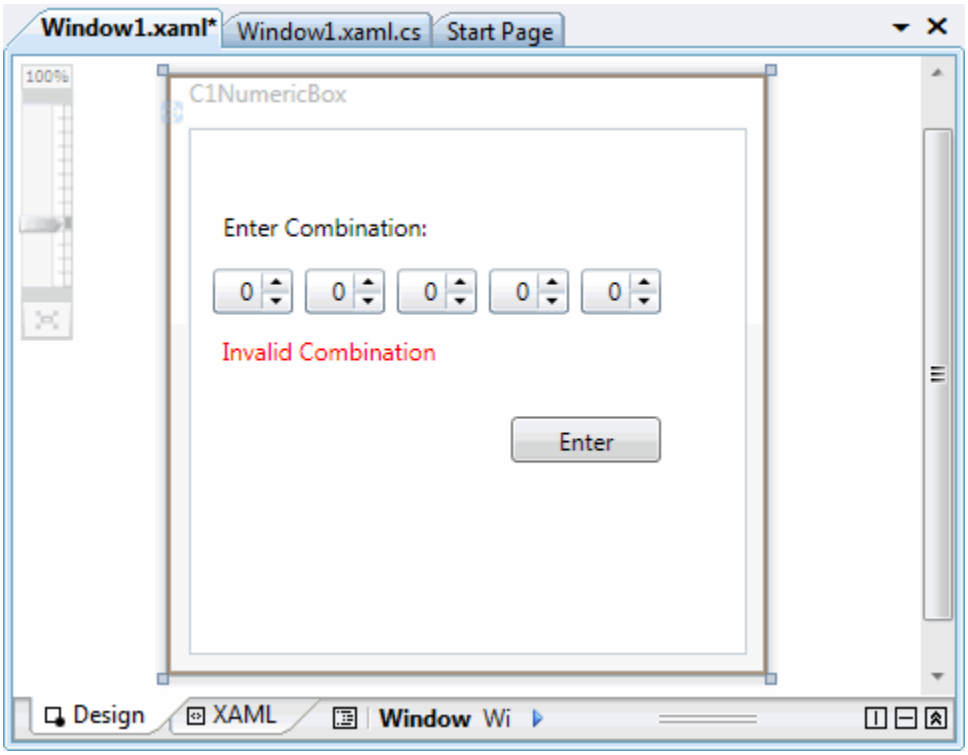
Step 2 of 4: Customizing the Application

In the previous step you created a new WPF 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. Navigate to the Visual Studio Toolbox and double-click the standard **Label** control twice to add **Label1** and **Label2** to your project.
2. In the Visual Studio Toolbox, and double-click the standard **Button** control to add **Button1** to your project.
3. Click **Label1** once to select it, and in the Properties window set its **Content** property to "Enter Combination:".
4. Click **Label2** once to select it, and in the Properties window set its **Content** property to "Invalid Combination" and its **Foreground** property to **Red**.
5. Click **Button1** once to select it, and in the Properties window set its **Content** property to "Enter" and its **Visibility** property to **Hidden**.

   Your application will now look similar to the following:



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. Double-click **Button1** to switch to Code view and create the **Button1_Click** event handler.
2. Add the following imports statements to the top of the page:
   - Visual Basic
     ```
     Imports C1.WPF
     Imports System.Windows.Media
     Imports System.Diagnostics
     ```
   - C#
     ```
     using C1.WPF;
     using System.Windows.Media;
     using System.Diagnostics;
     ```
3. Initialize the following global variables just inside class **Window1**:
   - 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 Button1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles Button1.Click
    Process.Start("http://www.componentone.com")
End Sub
```

   - C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    Process.Start("http://www.componentone.com");
}
```

   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.C1NumericBox1.Value = nb1 And Me.C1NumericBox2.Value = nb2
And Me.C1NumericBox3.Value = nb3 And Me.C1NumericBox4.Value = nb4 And
Me.C1NumericBox5.Value = nb5 Then
        Me.Label1.Foreground = Brushes.Green
        Me.Label1.Content = "Combination Valid"
        Me.C1NumericBox1.IsReadOnly = True
        Me.C1NumericBox2.IsReadOnly = True
        Me.C1NumericBox3.IsReadOnly = True
        Me.C1NumericBox4.IsReadOnly = True
        Me.C1NumericBox5.IsReadOnly = True
        Me.Button1.Visibility = Windows.Visibility.Visible
    End If
End Sub
```

   - C#

```
private void NBValidation()
{
    if (this.c1NumericBox1.Value == nb1 & this.c1NumericBox2.Value ==
nb2 & this.c1NumericBox3.Value == nb3 & this.c1NumericBox4.Value == nb4
& this.c1NumericBox5.Value == nb5)
    {
        this.label2.Foreground = Brushes.Green;
        this.label2.Content = "Combination Valid";
        this.c1NumericBox1.IsReadOnly = true;
        this.c1NumericBox2.IsReadOnly = true;
        this.c1NumericBox3.IsReadOnly = true;
        this.c1NumericBox4.IsReadOnly = true;
        this.c1NumericBox5.IsReadOnly = true;
        this.button1.Visibility = 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. Choose **View | Designer** to return to Design view.
7. Click **C1NumericBox1** to select it, and navigate to the Properties window.
8. Click the **Events** (lightning bolt) button on the Properties window to view events.
9. Double-click the box next to the **ValueChanged** event. This will switch to Code view and create the **C1NumericBox1_ValueChanged** event handler.
10. Enter the code in the **C1NumericBox1_ValueChanged** event handler to initialize **NBValidation**. It will look like the following:

    - Visual Basic

```
Private Sub C1NumericBox1_ValueChanged(ByVal sender As System.Object,
ByVal e As C1.WPF.PropertyChangedEventArgs(Of System.Double)) Handles
C1NumericBox1.ValueChanged
    NBValidation()
End Sub
```

    - C#

```
private void c1NumericBox1_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
```

11. Repeat steps 6 to 9 for each additional **C1NumericBox** control so that **NBValidation** is initialized in all five.

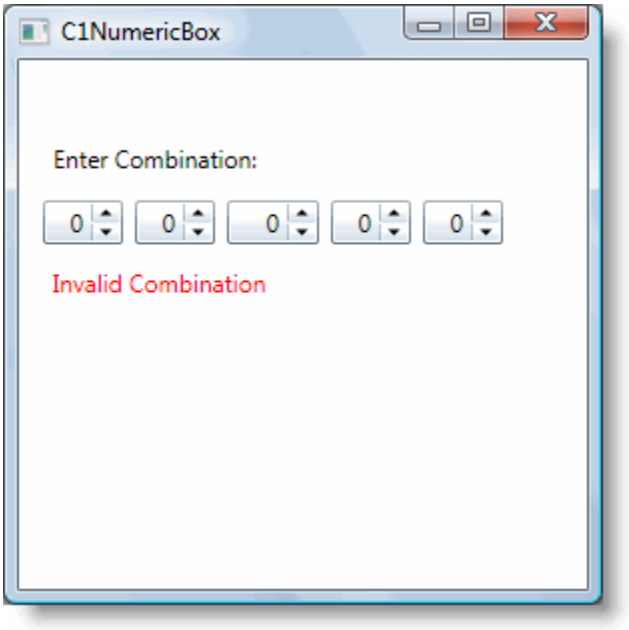
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 4 of 4: Running the Application

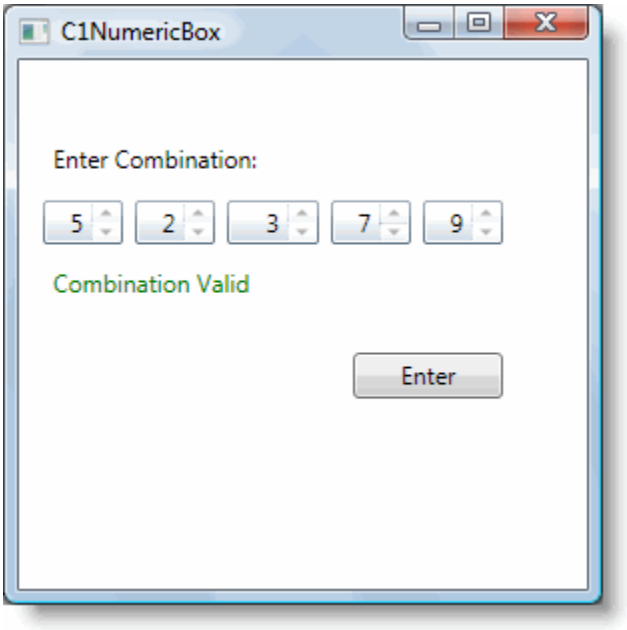
Now that you've created a WPF application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **NumericBox for WPF**'s run-time behavior, complete the following steps:

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:

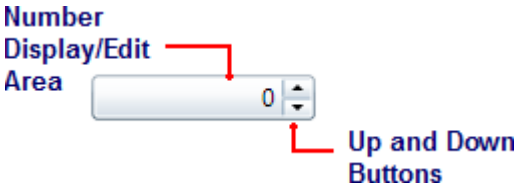


9. Try typing inside a **C1NumericBox** control or clicking its **Up** or **Down** buttons, notice that you cannot. That is because the IsReadOnly property was set to **True** when the correct number sequence was entered and the controls are now locked from editing.
10. Click the now-visible **Enter**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

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

Working with C1NumericBox

ComponentOne NumericBox for WPF 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 WPF includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see Appearance Properties (page 29) 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 C1NumericBox.
Minimum	Gets or sets the minimum value allowed for the C1NumericBox control.
Maximum	Gets or sets the maximum value allowed for the C1NumericBox.
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 C1NumericBox control.

Number Formatting

You can change how the number displayed in the C1NumericBox control will appear by setting the Format property. **ComponentOne NumericBox for WPF** 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…" where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. 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.
G or g	General	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. • Byte or SByte: 3 • Int16 or UInt16: 5 • Int32 or UInt32: 10 • Int64: 19 • UInt64: 20 • Single: 7 • Double: 15 • Decimal: 29 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. 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. 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.
N or n	Number	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. 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 or p	Percent	The number is converted to a string that represents a percent as defined by the NumberFormatInfo.PercentNegativePattern property if 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:

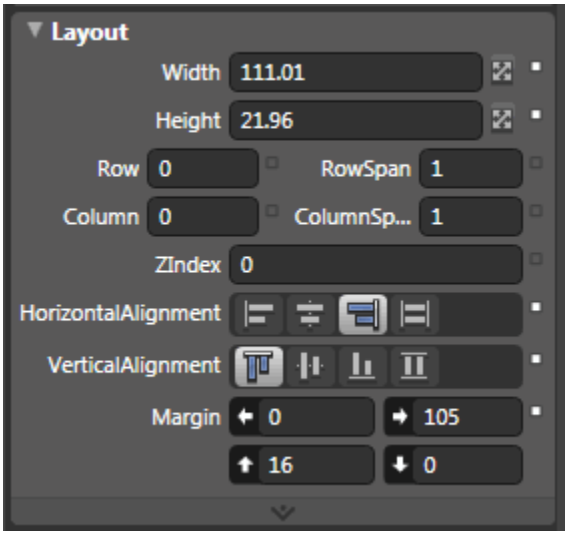
Option	Description
Always	This mode does not allow users to enter out of range values.
AlwaysTruncate	This mode does not allow users to enter out of range values. The value will be truncated if the limits are exceeded.
OnLostFocus	This mode truncates the value when the control loses focus.

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 WPF's XAML-based styling. You can also use templates to format and layout the grid and to customize grid actions.

Layout in a Panel

You can easily lay out the C1NumericBox and other controls in your WPF application, using the attached layout properties. For example, you can lay out your control in a **Grid** panel with its **Row**, **ColumnSpan**, and **RowSpan** properties and in a **Canvas** panel with its **Left** and **Top** properties. For example, the C1NumericBox control includes the following **Layout** properties when located within a **Grid** panel:



You can change the sizing, alignment, and location of the C1NumericBox control within the **Grid** panel.

Appearance Properties

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

Content Properties

The following properties let you customize the appearance of content in the **C1NumericBox** control:

Property	Description
Format	Gets or sets the value for the Format of the C1NumericBox.
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 **C1NumericBox**.

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.

ComponentOne ClearStyle Technology

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

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

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

How ClearStyle Works

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

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

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

ClearStyle Properties

The following table lists all of the ClearStyle-supported properties in the C1NumericBox control as well as a description of the property:

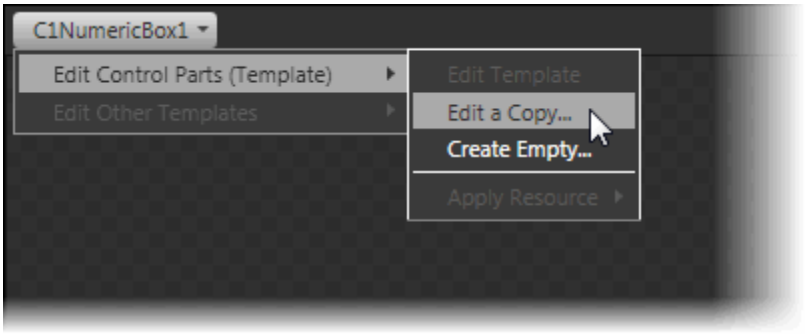
Property	Description
Background	Gets or sets a brush that describes the background of a control. The default **Background** color is White.
FocusBrush	A brush used to define the appearance of the control, when the control is in focus.
MouseOverBrush	A brush used to define the appearance of the control,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

Templates

One of the main advantages to using a WPF control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for WPF applications, you can provide your own UI for data managed by **ComponentOne NumericBox for WPF**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the 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.

XAML Elements

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

Included Auxiliary XAML Elements

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

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

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

NumericBox for WPF Samples

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

C# Samples

The following C# sample is included:

Sample	Description
ControlExplorer	The **NumericBox** page in the **ControlExplorer** sample demonstrates how to add content to and customize the C1NumericBox control.

NumericBox for WPF Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1NumericBox control in general. If you are unfamiliar with the **ComponentOne NumericBox for WPF** product, please see the NumericBox for WPF Quick Start (page 17) first.

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

Each task-based help topic also assumes that you have created a new WPF project. For additional information on this topic, see Creating a .NET Project in Visual Studio (page 12) or Creating a Microsoft Blend Project (page 11).

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 XAML, and in code.

At Design Time

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

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties window, and enter a number, for example "123", in the text box next to the Value property.

   This will 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 Height="21.96" HorizontalAlignment="Right"
Margin="0,16,105,0" Name="C1NumericBox1" VerticalAlignment="Top"
Width="111.01" Value="123" />
```

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 XAML, and in code.

At Design Time

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

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties window, 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 Height="21.96" HorizontalAlignment="Right"
Margin="0,16,105,0" Name="C1NumericBox1" VerticalAlignment="Top"
Width="111.01" Increment="20" />
```

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

To set the Minimum and Maximum at run time, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties window, and enter a number, for example **500**, next to the Maximum property.
3. In the Properties window, 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 Height="21.96" HorizontalAlignment="Right"
Margin="0,16,105,0" Name="C1NumericBox1" VerticalAlignment="Top"
Width="111.01" Maximum="500" Minimum="-500" />
```

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 window, through XAML, or through code.

At Design Time

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

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties window, and set **FontFamily** property to "Arial".
3. In the Properties window, set the **FontSize** property to **10**.

   This will 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 Height="21.96" HorizontalAlignment="Right"
Margin="0,16,105,0" Name="C1NumericBox1" VerticalAlignment="Top"
Width="111.01" FontFamily="Arial" FontSize="10" />
```

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

To hide the **Up** and **Down** buttons at run time, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties window, 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 Height="21.96" HorizontalAlignment="Right"
Margin="0,16,105,0" Name="C1NumericBox1" VerticalAlignment="Top"
Width="111.01" ShowButtons="False" />
```

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

To lock the C1NumericBox control from run-time editing, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties window, 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 Height="21.96" HorizontalAlignment="Right"
Margin="0,16,105,0" Name="C1NumericBox1" VerticalAlignment="Top"
Width="111.01" IsReadOnly="True" />
```

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: