
ComponentOne

MaskedTextBox for WPF

Copyright © 2012 ComponentOne LLC. All rights reserved.

Corporate Headquarters

ComponentOne LLC

201 South Highland Avenue

3rd Floor

Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com

Web site: <http://www.componentone.com>

Sales

E-mail: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of ComponentOne LLC. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the original CD (or diskettes) are free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective CD (or disk) to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for a defective CD (or disk) by sending it and a check for \$25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. We are not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

This manual was produced using [ComponentOne Doc-To-Help™](#).

Table of Contents

ComponentOne MaskedTextBox for WPF Overview	1
Installing MaskedTextBox for WPF	1
Studio for WPF Setup Files.....	1
Using Maps Powered by Esri	2
System Requirements	3
Installing Demonstration Versions.....	4
Uninstalling MaskedTextBox for WPF.....	4
End-User License Agreement	4
Licensing FAQs	4
What is Licensing?.....	4
How does Licensing Work?.....	5
Common Scenarios	5
Troubleshooting.....	7
Technical Support	9
Redistributable Files.....	10
About this Documentation.....	10
XAML and XAML Namespaces.....	10
Creating a Microsoft Blend Project.....	11
Creating a .NET Project in Visual Studio	12
Creating an XAML Browser Application (XBAP) in Visual Studio	13
Adding the MaskedTextBox for WPF Components to a Blend Project	14
Adding the MaskedTextBox for WPF Components to a Visual Studio Project	14
Key Features	15
MaskedTextBox for WPF Quick Start.....	17
Step 1 of 4: Setting up the Application.....	17
Step 2 of 4: Customizing the Application	18
Step 3 of 4: Adding Code to the Application	18
Step 4 of 4: Running the Application.....	20
About C1MaskedTextBox.....	23
Basic Properties.....	23

Mask Formatting.....	23
Mask Elements	24
Literals.....	25
Prompts	25
Watermark	25
Layout and Appearance.....	26
Appearance Properties	26
Content Properties.....	26
Text Properties	26
Color Properties.....	27
Border Properties.....	27
Size Properties	27
ComponentOne ClearStyle Technology	28
How ClearStyle Works.....	28
ClearStyle Properties	28
Templates.....	28
XAML Elements.....	29
MaskedTextBox for WPF Samples	31
MaskedTextBox for WPF Task-Based Help	31
Setting the Value	31
Adding a Mask for Currency	32
Changing the Prompt Character	33
Changing Font Type and Size.....	33
Locking the Control from Editing.....	34

ComponentOne MaskedTextBox for WPF Overview

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

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



Getting Started

Get started with the following topics:

- [Key Features](#) (page 15)
- [Quick Start](#) (page 17)
- [Task-Based Help](#) (page 31)

Installing MaskedTextBox for WPF

The following sections provide helpful information on installing **ComponentOne MaskedTextBox 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 **ComponentOne MaskedTextBox 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 C1MaskedTextBox 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\\My Documents\ComponentOne Samples

Windows 7/Vista path: C:\Users\\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\

64-bit machine: C:\Program Files (x86)\ESRI SDKs\

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.

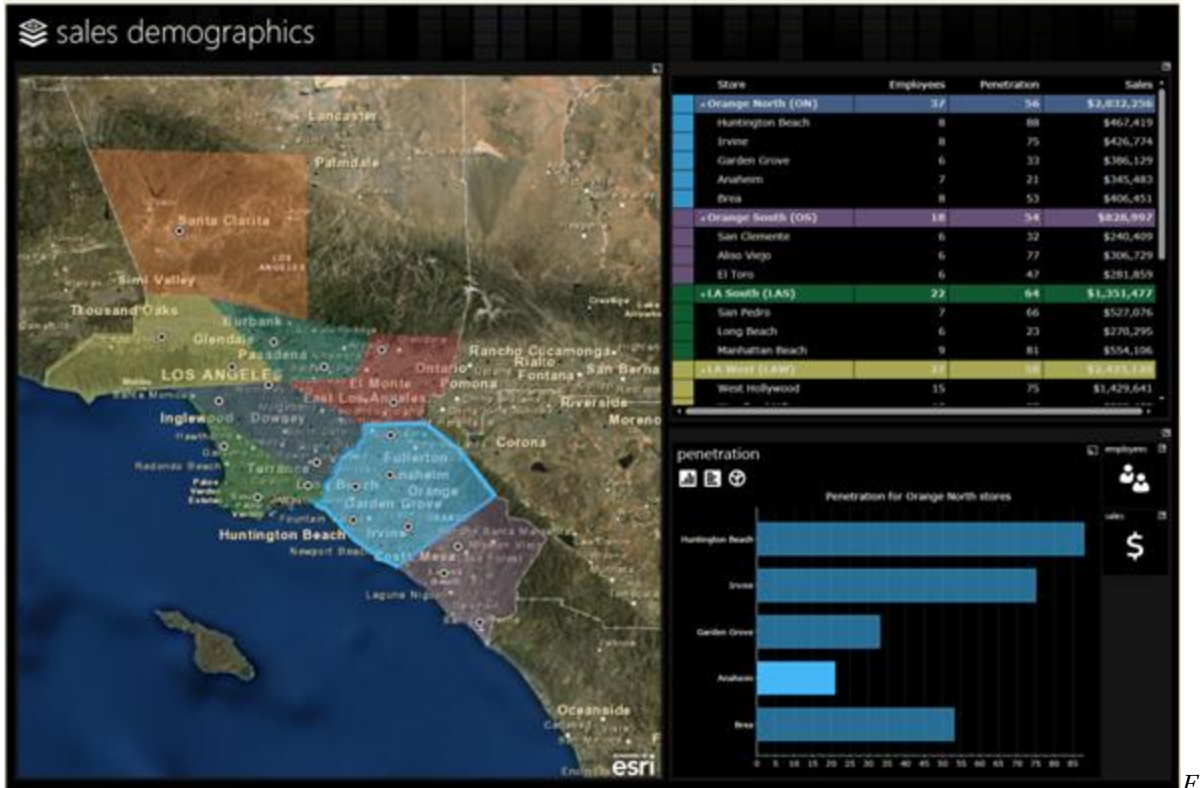
Using Maps Powered by Esri

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.



sri 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: **MaskedTextBox 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 **MaskedTextBox 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 MaskedTextBox 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 MaskedTextBox 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 **CILc.exe** utility and the licensed DLLs to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use **CILc.exe** to compile the **licenses.licx** file. The command line should look like this:
`cilc /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 MaskedTextBox 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

Microsoft, Windows, Windows 7, Windows Vista, Visual Studio, and Microsoft Expression are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Esri is a registered trademark of Environmental Systems Research Institute, Inc. (Esri) in the United States, the European Community, or certain other jurisdictions.

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 and 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
<code>xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"</code>	This is the default Windows Presentation Foundation namespace.
<code>xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"</code>	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 `CIMaskedTextBox` 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:MyMTB="http://schemas.componentone.com/winfx/2006/xaml"
```

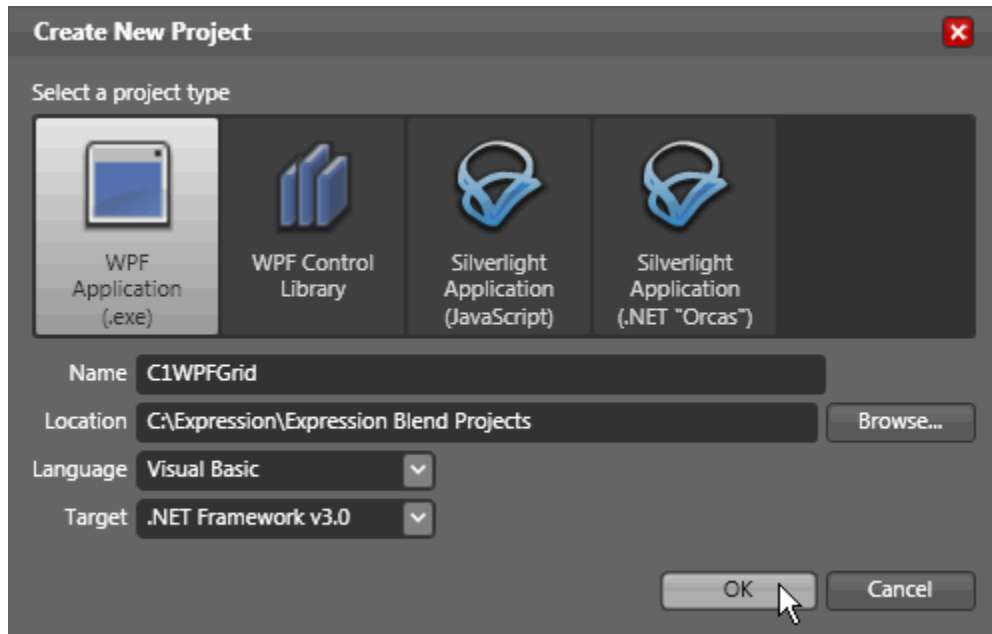
You can now use your custom namespace when assigning properties, methods, and events. For example, use the following XAML to add a border around the `MaskedTextBox`:

```
<MyMTB:CIMaskedTextBox Name="CIMaskedTextBox1" 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**.

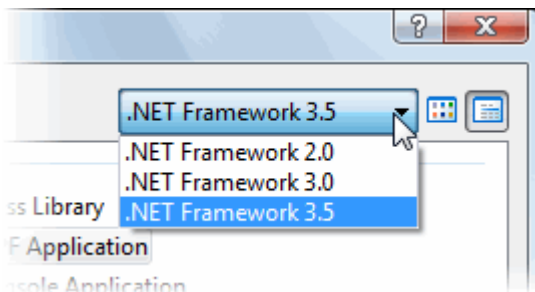


A new Blend project with a XAML window is created.

Creating a .NET Project in Visual Studio

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

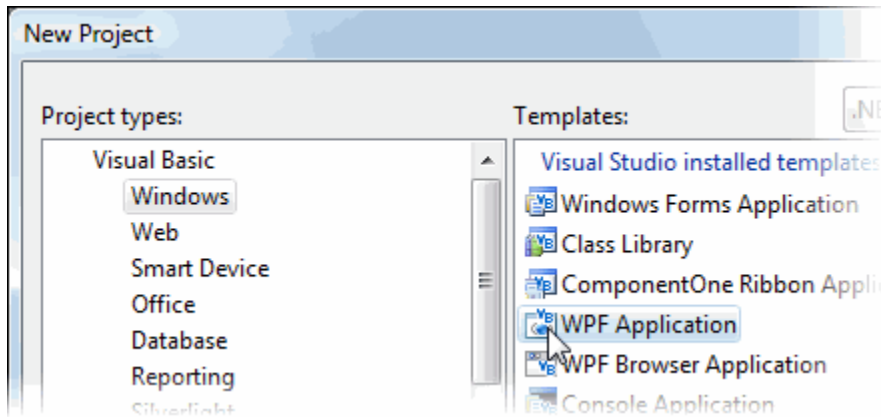
1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**.
The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



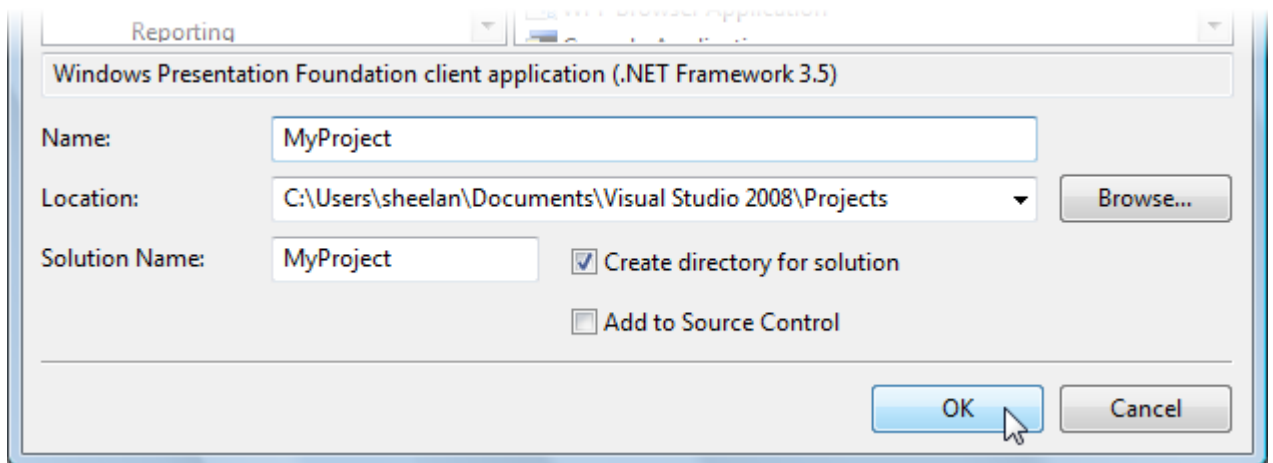
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

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**.



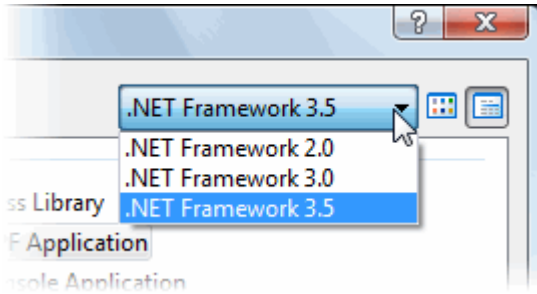
A new Microsoft Visual Studio .NET WPF project is created with a XAML file that will be used to define your user interface and commands in the application.

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 MaskedTextBox for WPF Components to a Blend Project

In order to use `C1MaskedTextBox` or another **ComponentOne MaskedTextBox 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 **MaskedTextBox 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 **C1MaskedTextBox**. The component will appear in the Toolbox below the **Asset Library** button.
4. Double-click the **C1MaskedTextBox** component in the Toolbox to add it to **Window1.xaml**.

Adding the MaskedTextBox for WPF Components to a Visual Studio Project

When you install **ComponentOne MaskedTextBox for WPF** the `C1MaskedTextBox` control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

ComponentOne MaskedTextBox for WPF provides the following control:

- C1MaskedTextBox

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

Manually Adding MaskedTextBox for WPF to the Toolbox

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

- C1MaskedTextBox

To manually add the C1MaskedTextBox 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 **MaskedTextBox for WPF** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1WPFMaskedTextBox**, 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 MaskedTextBox for WPF to the Window

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

1. Add the C1MaskedTextBox control to the Visual Studio Toolbox.
2. Double-click C1MaskedTextBox or drag the control onto the window.

Adding a Reference to the Assembly

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

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne MaskedTextBox 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 **MaskedTextBox for WPF** assembly visible to the project.

Key Features

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

- **Validate Data and Enhance Your UI**

The ComponentOne masked text box control (C1MaskedTextBox) provides a text box with a mask that automatically validates the input. The edit mask enhances the UI by preventing end-users from entering invalid characters into the control.

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

The masked text box control includes a Watermark property, which lets end-users know what type of information is expected.

MaskedTextBox for WPF Quick Start

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

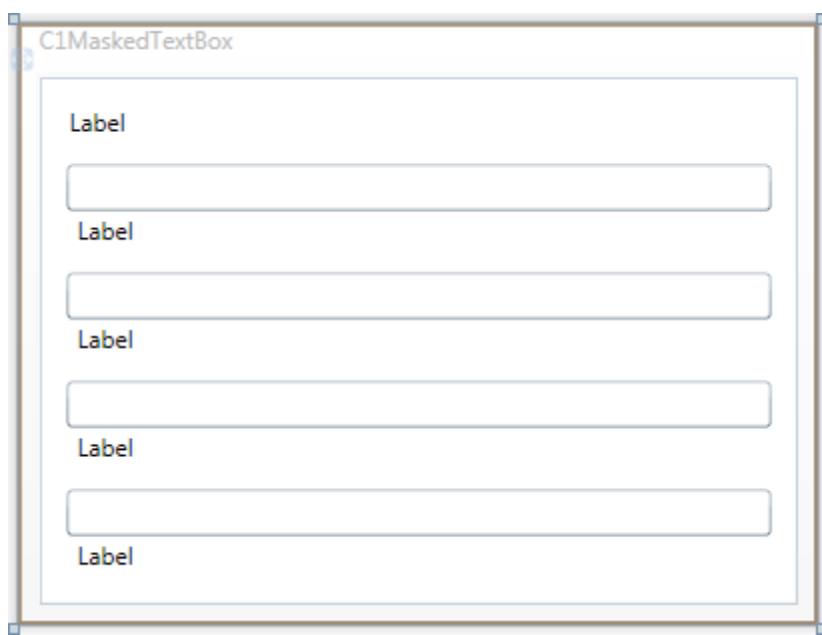
You will create a simple form using several **C1MaskedTextBox** controls that will demonstrate the difference between the **Text** and **Value** properties. The controls will include various masks and different appearance and behavior settings so that you can explore the possibilities of using **MaskedTextBox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

Step 1 of 4: Setting up the Application

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

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

1. Create a new WPF project in Visual Studio. For more information about creating a WPF project, see [Creating a .NET Project in Visual Studio](#) (page 12).
2. Resize the initial window by setting Window1's **Width** to "400".
3. Navigate to the Toolbox and double-click the **C1MaskedTextBox** icon to add the control to Window1. Repeat this step 3 more times to add a total of 4 **C1MaskedTextBox** controls.
4. In the Toolbox, double-click the **Label** icon to add the control to Window1. Repeat this step 4 more times to add a total of 5 standard **Label** controls.
5. Resize the controls and rearrange the controls on the window with the controls numbered smallest to largest from top to bottom alternating **Label** and **C1MaskedTextBox** controls. Your application should now appear similar to the following:



You've successfully created a WPF application and added **C1MaskedTextBox** controls to the application. In the next step you'll customize those controls and complete setting up the application.

Step 2 of 4: Customizing the Application

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

Complete the following steps:

1. In Design view, click once on the **Label1** control to select it, navigate to the Properties window, and set its **Content** property to "Employee Information".
2. Select each remaining **Label** control in turn, navigate to the Properties window, and set the following for each:
 - Delete the default "Label" text next to **Content** property.
 - Set the **FontSize** property to "9".

3. Switch to XAML view and customize **C1MaskedTextBox1** by adding `Watermark="Name"` to the `<c1:C1MaskedTextBox>` tag so it appears similar to the following:

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

This will add a watermark to the control.

4. Switch to XAML view and customize **C1MaskedTextBox2** by adding `Watermark="Employee ID"` `Mask="000-00-0000"` to the `<c1:C1MaskedTextBox>` tag so it appears similar to the following:

```
<c1:C1MaskedTextBox Margin="14,98,12,0" Name="C1MaskedTextBox2"
Height="23" VerticalAlignment="Top" Watermark="Employee ID" Mask="000-00-
0000" />
```

This will add a watermark and mask to the control.

5. Switch to XAML view and customize **C1MaskedTextBox3** by adding `Watermark="Hire Date"` `Mask="00/00/0000"` to the `<c1:C1MaskedTextBox>` tag so it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Margin="14,0,12,87"
Name="C1MaskedTextBox3" VerticalAlignment="Bottom" Watermark="Hire Date"
Mask="00/00/0000"/>
```

This will add a watermark and mask to the control.

6. Switch to XAML view and customize **C1MaskedTextBox4** by adding `Watermark="Phone Number"` `Mask="(999) 000-0000"` to the `<c1:C1MaskedTextBox>` tag so it appears similar to the following:

```
<my:C1MaskedTextBox Height="23" Margin="14,0,12,33"
Name="C1MaskedTextBox4" VerticalAlignment="Bottom" Watermark="Phone
Number" Mask="(999) 000-0000"/>
```

This will add a watermark and mask to the control.

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

Step 3 of 4: Adding Code to the Application

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

Complete the following steps:

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

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

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

- Visual Basic

```
Imports C1.WPF
```

- C#

```
using C1.WPF;
```

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

- Visual Basic

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

- C#

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

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

- Visual Basic

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

- C#

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

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

- Visual Basic

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

- C#

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

```

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

```

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

- Visual Basic

```

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

```

- C#

```

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

```

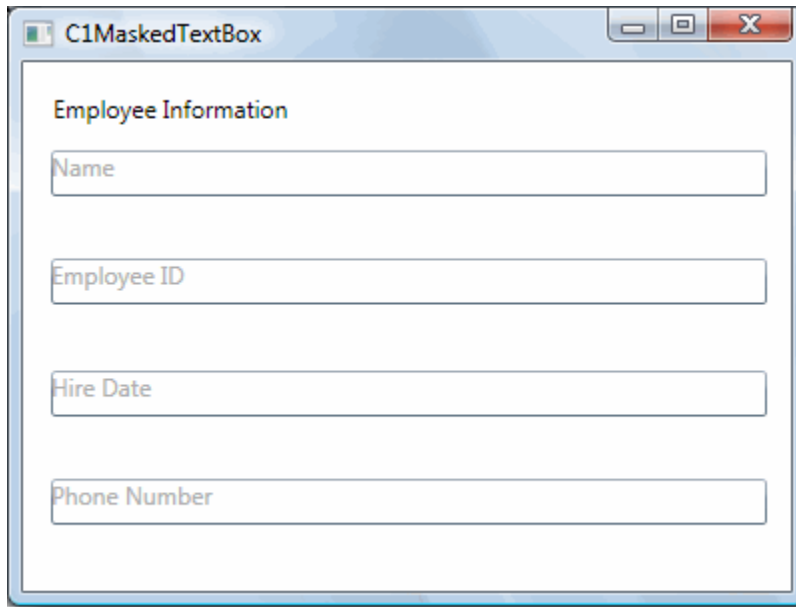
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 4 of 4: Running the Application

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

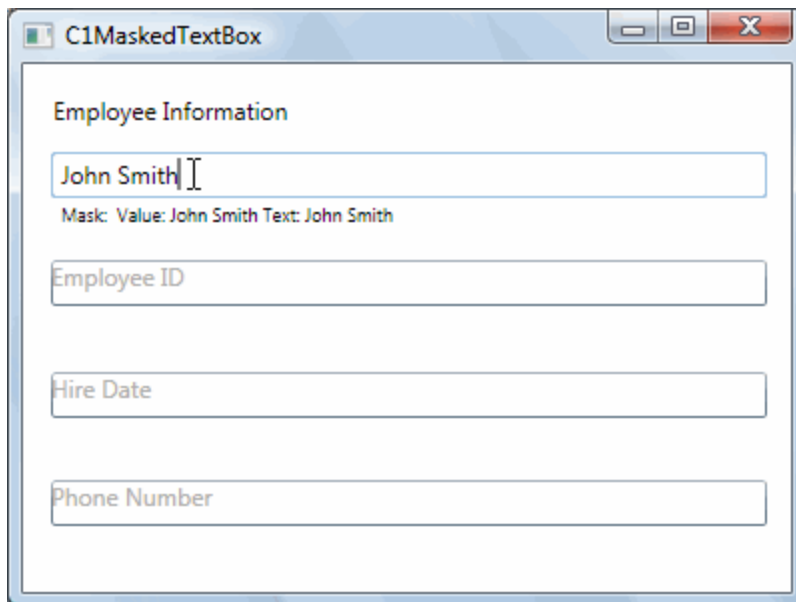
1. From the **Project** menu, select **Test Solution** to view how your application will appear at run time.

The application will appear similar to the following:



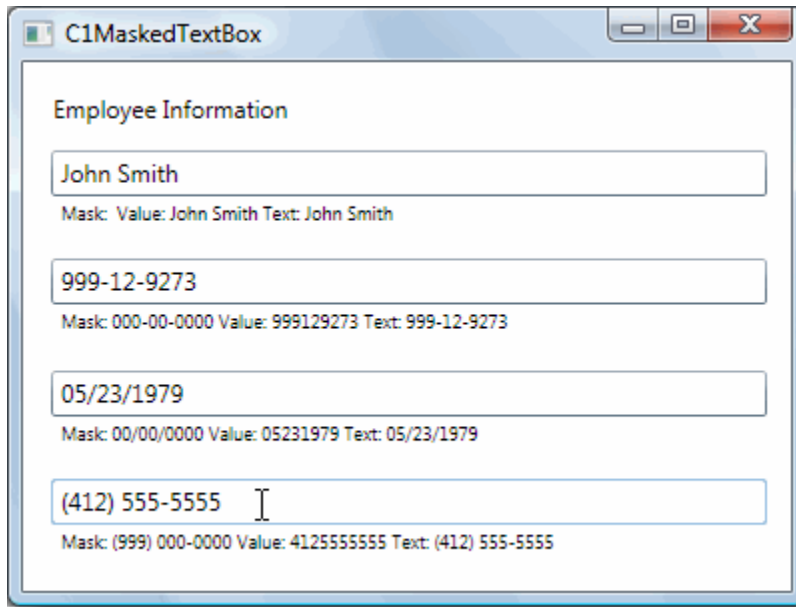
Notice the watermark that appears in each **C1MaskedTextBox** control.

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



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

3. Try entering a string in the second **C1MaskedTextBox** control. Notice that you cannot – that is because the Mask property was set to only accept numbers. Enter a numeric value instead – notice that this does work.
4. Enter numbers in each of the remaining controls. The application will appear similar to the following image:

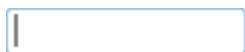


Notice that the Value property displayed under each C1MaskedTextBox control does not include literal characters, while the **Text** property does.

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

About C1MaskedTextBox

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



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

Basic Properties

ComponentOne MaskedTextBox for WPF includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see [Appearance Properties](#) (page 26) for more information about properties that control appearance.

The following properties let you customize the C1MaskedTextBox control:

Property	Description
Mask	Gets or sets the input mask to use at run time. See Mask Formatting (page 23) for more information.
PromptChar	Gets or sets the character used to show spaces where user is supposed to type.
Text	Gets or sets the text content of this element.
TextMaskFormat	Gets or sets a value that determines whether literals and prompt characters are included in the Value property.
Value	Gets the formatted content of the control as specified by the TextMaskFormat property.
Watermark	Gets or sets the content of the watermark.

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

Mask Formatting

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

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

The following table lists some example masks:

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

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

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

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

Mask Elements

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

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

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

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

Literals

In addition to the mask elements defined in the [Mask Formatting](#) (page 23) topic, other characters can be included in the mask. These characters are *literals*. Literals are non-mask elements that will appear as themselves within C1MaskedTextBox. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user.

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

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

Prompts

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

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

Watermark

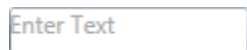
Using the Watermark property you can provide contextual clues of what value users should enter in a C1MaskedTextBox control. The watermark is displayed in the control while not text has been entered. To add a

watermark, add the text `Watermark="Watermark Text"` to the `<c1:C1MaskedTextBox>` tag in the XAML markup for any **C1MaskedTextBox** control.

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

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

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



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

Layout and Appearance

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

Appearance Properties

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

Content Properties

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

Property	Description
Mask	Gets or sets the input mask to use at run time. See Mask Formatting (page 23) for more information.
PromptChar	Gets or sets the character used to show spaces where user is supposed to type.
Watermark	Gets or sets the content of the watermark.

Text Properties

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

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

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

Color Properties

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

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

Border Properties

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

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

Size Properties

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

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

ComponentOne ClearStyle Technology

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

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

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

How ClearStyle Works

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

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

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

ClearStyle Properties

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

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

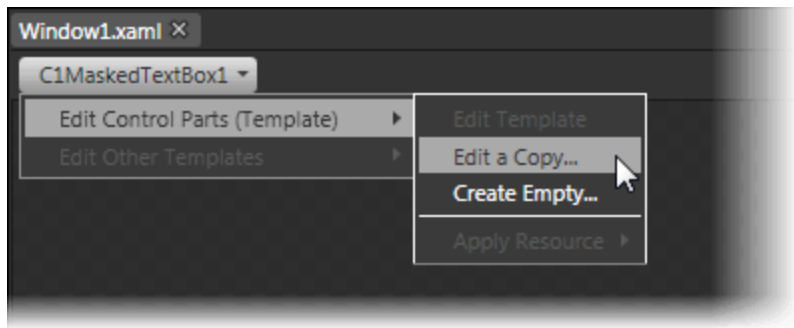
Templates

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

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

Accessing Templates

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



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

Note that you can use the [Template](#) property to customize the template.

XAML Elements

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

Included Auxiliary XAML Elements

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

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

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

MaskedTextBox for WPF Samples

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

C# Samples

The following C# sample is included:

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

MaskedTextBox for WPF Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1MaskedTextBox control in general. If you are unfamiliar with the **ComponentOne MaskedTextBox for WPF** product, please see the [MaskedTextBox for WPF Quick Start](#) (page 17) first.

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

Each task-based help topic also assumes that you have created a new WPF project. For additional information on this topic, see [Creating a .NET Project in Visual Studio](#) (page **Error! Bookmark not defined.**) or [Creating a Microsoft Blend Project](#).

Setting the Value

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

At Design Time

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

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

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

In XAML

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

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

In Code

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

- Visual Basic

```
C1MaskedTextBox1.Value = 123
```
- C#

```
c1MaskedTextBox1.Value = 123;
```

Run your project and observe:

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

Adding a Mask for Currency

You can easily add a mask for currency values using the Mask property. By default the C1MaskedTextBox control starts with its Mask not set but you can customize this at design time, in XAML, and in code. For more details about mask characters, see [Mask Elements](#) (page 24).

At Design Time

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

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

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

In XAML

For example, to set the Mask property add `Mask="$999,999.00"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="C1MaskedTextBox1" VerticalAlignment="Top"
Width="120" Mask="$999,999.00"></c1:C1MaskedTextBox>
```

In Code

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

- Visual Basic

```
C1MaskedTextBox1.Mask = "$999,999.00"
```
- C#

```
c1MaskedTextBox1.Mask = "$999,999.00";
```

Run your project and observe:

The mask will appear in the control:

Enter a number; notice that the mask is filled:

Changing the Prompt Character

The PromptChar property sets the characters that are used to prompt users in the C1MaskedTextBox control. By default the PromptChar property is set to an underline character ("_") but you can customize this at design time, in XAML, and in code. For more details about the PromptChar property, see [Prompts](#) (page 25).

At Design Time

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

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

In XAML

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

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

In Code

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

- Visual Basic

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

- C#

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

Run your project and observe:

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



Changing Font Type and Size

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

At Design Time

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

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

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

In XAML

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

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

In Code

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

- Visual Basic

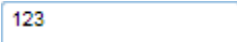
```
C1MaskedTextBox1.FontSize = 10
C1MaskedTextBox1.FontFamily = New System.Windows.Media.FontFamily("Arial")
```

- C#

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

Run your project and observe:

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



Locking the Control from Editing

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

At Design Time

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

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

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

In XAML

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

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

In Code

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

- Visual Basic

```
C1MaskedTextBox1.IsReadOnly = True
```

- C#

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

Run your project and observe:

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

123