ComponentOne

Studio 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

Getting Started with Studio for WPF

Leveraging the full potential of Windows Presentation Foundation (WPF), **ComponentOne Studio® for WPF** offers everything from high-performing datagrids, charts and scheduler controls to slick layout and navigation controls. Accomplish more enterprise-level functionality in less time.

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:

Installing Studio for WPF

The following sections provide helpful information on installing **ComponentOne Studio 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\Version	Contains copies of all ComponentOne binaries (DLLs, EXEs). In addition, files from the Microsoft WPF Toolkit are also installed. For more information about the Microsoft WPF Toolkit, see CodePlex. The C1.WPF.dll and WPFToolkit.dll assemblies are required for deployment.
AssemblyName\XAML	Contains the full XAML definitions of WPF 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 Viewer help to the **C:\Program Files\ComponentOne\Studio for WPF\HelpViewer** directory.

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 organized in subfolders by assemby and language.

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.

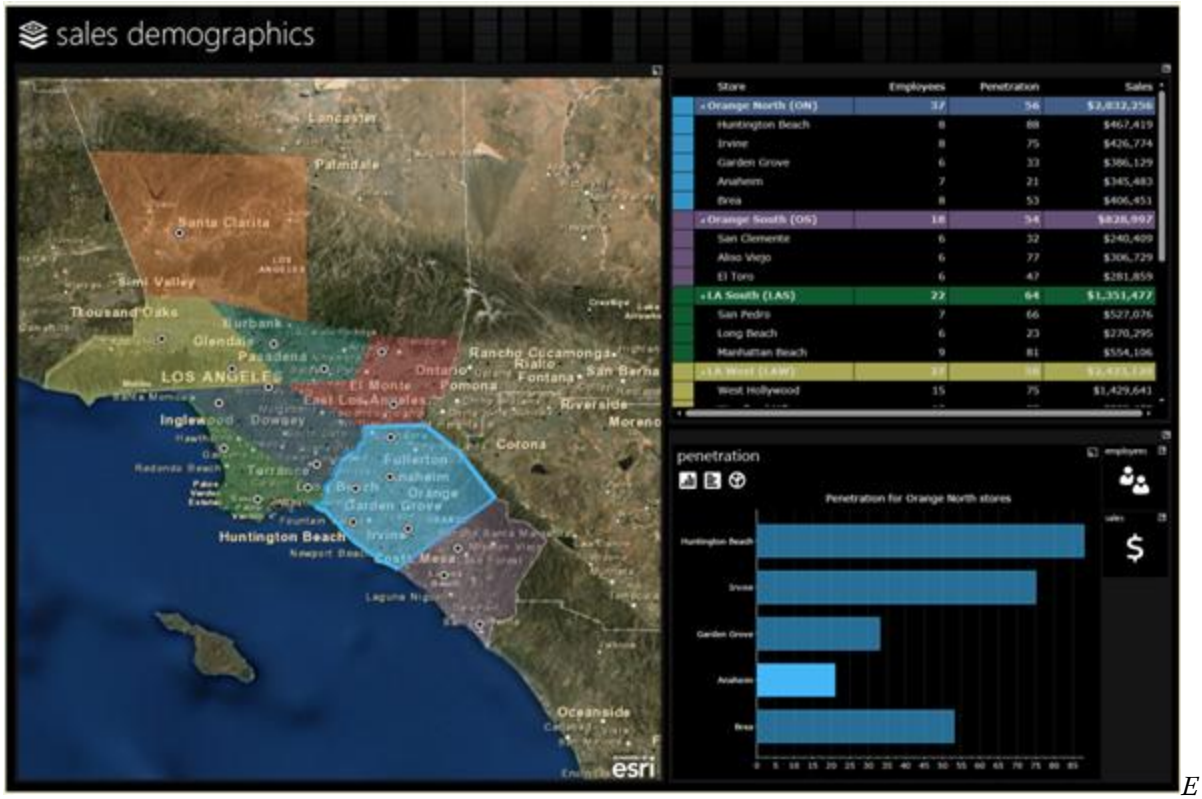
Using Maps Powered by Esri

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 or later
	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:	Studio for WPF includes design-time support for Expression Blend.

Installing Demonstration Versions

If you wish to try **ComponentOne Studio 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 Studio for WPF

To uninstall ComponentOne **Studio for WPF**:

1. Open the **Control Panel** and select **Add or Remove Programs** or **Programs and Features**.
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** or **Programs and Features**.
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 Studio for WPF is developed and published by GrapeCity, Inc. 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.*ProductName*.*v*.dll

Where *ProductName* is the name of the product (note, not included in C1.WPF.dll) and *v* is the optional version number (for example 4 for .NET 4.0; .NET 3.5 files do not include a version number).

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

Contact Us

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

Corporate Headquarters

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 • USA
1-800-858-2739 • 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 **Studio for WPF** 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** and the namespace is **http://schemas.componentone.com/winfx/2006/xaml**. 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:My="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 **C1Book** with a border:

```
<My:C1Book Name="C1Book1" 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 WPF Project in Visual Studio

To create a new WPF project in Visual Studio, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio, 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.

Adding the Studio for WPF Components to a Blend Project

In order to use a **ComponentOne Studio for WPF** component in the Design workspace of Blend, you must first add a reference to the **Studio for WPF** assemblyof that component and then add the component from Blend's **Asset Library**.

To add a reference to the assembly:

1. Select **Project | Add Reference**.
2. Browse to find the **C1.WPF** assembly installed with **Studio for WPF** that you wish to add (for example, **C1.WPF.DataGrid.dll**).

   Note: See Studio for WPF Setup Files (page 1) for installation directories.

3. Select the **C1.WPF** DLL file (for example, **C1.WPF.DataGrid.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 required **C1.WPF** assembly, click the **Asset Library** button (it appears like a sideways chevron) in the Blend Toolbox. The **Asset Library** appears.
2. Click the **Controls** drop-down arrow and select **All**.
3. Select the control. It will appear in the Toolbox below the **Asset Library** button.
4. Double-click the component in the Toolbox to add it to the window.

Adding the Studio for WPF Components to a Visual Studio Project

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

ComponentOne Studio for WPF provides several controls. To use a **Studio for WPF** panel or control, add it to the window or add a reference to the required **C1.WPF** assembly to your project (for example, **C1.WPF.DataGrid.dll**).

Manually Adding Studio for WPF to the Toolbox

When you install **Studio for WPF**, several **Studio for WPF** controls and panels will appear in the Visual Studio Toolbox customization dialog box.

To manually add the **Studio for WPF** controls 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 **Studio for WPF** components appear on their own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1WPF**, 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 that you wish to add. Note that there may be more than one component for each namespace.

Adding Studio for WPF controls to the Window

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

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

Adding a Reference to the Assembly

To add a reference to the **Studio for WPF** assemblies, complete the following steps:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the desired **ComponentOne Studio for WPF** assembly from the list on the **.NET** tab or on the **Browse** tab, browse to find the **C1.WPF.*Product*.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
Imports C1.WPF.Product
```

   This makes the objects defined in the **Studio for WPF** assembly visible to the project. Note that in the steps above, replace "Product" with the correct assembly name, for example "Extended" or "DataGrid".

Localization

With Studio for WPF and Silverlight, you can now broaden your global audience with built-in localization support for 20+ international languages. This means that all UI strings baked into ComponentOne WPF and Silverlight controls can be automatically translated into these languages (aside from English):

1. Arabic (ar)
2. Czech (cs)
3. Danish (da)
4. Dutch (nl)
5. German (de)
6. Greek (el)
7. Spanish (es)
8. Finnish (fi)
9. French (fr)
10. Hebrew (he)
11. Italian (it)
12. Japanese (ja)
13. Norwegian (no)
14. Polish (pl)
15. Portuguese (pt)
16. Russian (ru)
17. Slovak (sk)
18. Swedish (sv)
19. General Chinese (zh)
20. Traditional Chinese (zh-Hant) - Taiwan, Hong Kong

21. Simplified Chinese (zh-Hans) - China

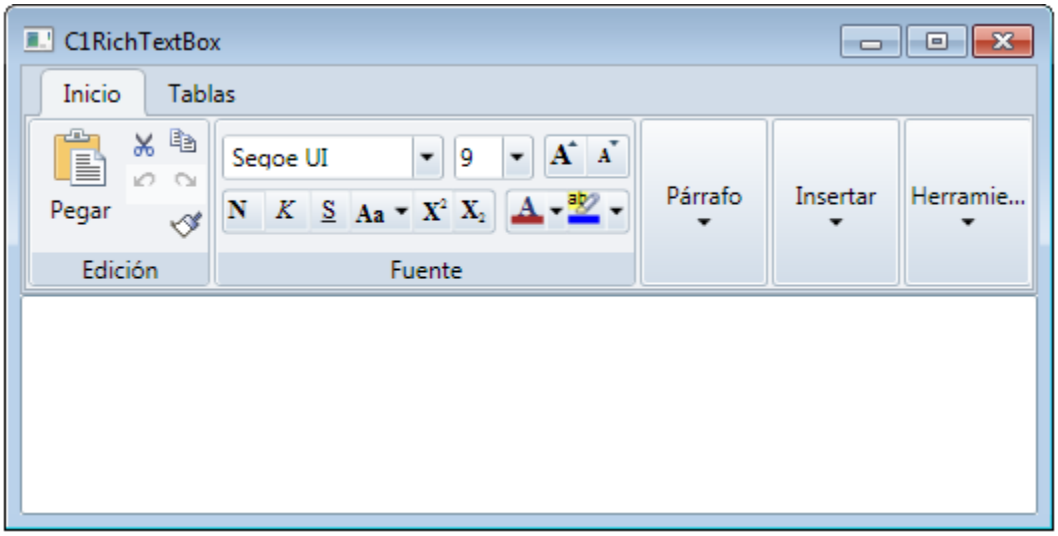
WPF Localization

When you localize a WPF application, you have several options. See here for best practices from Microsoft. When you use ComponentOne controls in your application, the localized resources are automatically included in your output based upon the published language you specify in your project's settings. They will come from their installed location under C:\Program Files\ComponentOne\Studio for WPF\bin, and do not need to be added manually to the project.

Change the UI Culture of your application to the desired culture. One way of doing this is on the application's current thread:

- Visual Basic
```
' set culture to Spanish ("es")
Thread.CurrentThread.CurrentUICulture = New CultureInfo("es")
```
- C#
```
// set culture to Spanish ("es")
Thread.CurrentThread.CurrentUICulture = new CultureInfo("es");
```

Some controls have many baked in UI strings that can benefit from quick localization including **C1Scheduler**, **C1DataGrid**, **C1DockControl**, **C1PdfViewer** and **C1RichTextBoxToolbar**.



Silverlight Localization

Localization for Silverlight applications is a bit different, as it has two extra steps.

First, you should add the desired localized resource files (.resx) to your project's Resources directory (Build Action: Embedded Resource). We provide the localized resource files for each ComponentOne assembly located at C:\Program Files\ComponentOne\Studio for Silverlight\Help\LocalizationResources.zip.

Second, you must unload your project and edit the <SupportedCultures> node in the project .csproj file. See here for more information.

```
<SupportedCultures>es,en</SupportedCultures>
```

The final step is the same in WPF; change the application thread's culture.

- Visual Basic
```
' set culture to Spanish ("es")
```

```
Thread.CurrentThread.CurrentUICulture = New CultureInfo("es")
```

- C#

```
// set culture to Spanish ("es")
Thread.CurrentThread.CurrentUICulture = new CultureInfo("es");
```

Check out a live demo showing every provided language in **Studio for Silverlight**.

This information just focuses on localizing C1 controls. You will likely have other UI strings that need to be localized too, so see here for more information about Silverlight localization from Microsoft.