
ComponentOne

Book 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 Book for WPF Overview	1
Installing Book for WPF	1
Studio for WPF Setup Files.....	1
Using Maps Powered by Esri	2
System Requirements	3
Installing Demonstration Versions.....	4
Uninstalling Book for WPF.....	4
End-User License Agreement	4
Licensing FAQs	4
What is Licensing?.....	5
How does Licensing Work?.....	5
Common Scenarios	6
Troubleshooting.....	8
Technical Support	9
Redistributable Files.....	10
About this Documentation.....	10
XAML and XAML Namespaces.....	11
Creating a Microsoft Blend Project.....	11
Creating a .NET Project in Visual Studio	12
Adding the Book for WPF Components to a Blend Project	13
Adding the Book for WPF Components to a Visual Studio Project.....	14
Key Features.....	15
Book for WPF Quick Start	17
Step 1 of 3: Creating the Book Application.....	17
Step 2 of 3: Adding Content to the Book Control.....	18
Step 3 of 3: Running the Book Application	21
Working with Book for WPF	25
XBAP Support	25
Basic Properties.....	25
Basic Events	26

Book Zones	26
Page Fold Size.....	28
Page Fold Visibility	29
Page Turning Options	29
First Page Display	30
Page Shadows	31
Book Navigation	31
Book for WPF Layout and Appearance	32
Book for WPF Appearance Properties.....	32
Color Properties.....	32
Alignment Properties.....	32
Border Properties.....	32
Size Properties	33
Book Templates.....	33
Page Templates	34
Book Styles.....	34
Book Template Parts.....	35
Book Visual States.....	35
Book for WPF Samples.....	37
Book for WPF Task-Based Help	37
Creating a Book.....	37
Adding Items to a Book	39
Clearing Items in a Book.....	40
Displaying the First Page on the Right	40
Setting the Initial Page	41
Navigating the Book with Code.....	41

ComponentOne Book for WPF Overview

Present information using a familiar book metaphor with **C1Book**, a page-turning book control for innovative WPF navigation. With the **C1Book** control you can present **UIElement** objects as if they were pages in a regular paper book. You can see two elements at a time, add shadows, turn pages with the mouse, and more with **ComponentOne Book™ for WPF**.

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 37)

Installing Book for WPF

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

Studio for WPF Setup Files

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

Bin Contains copies of all ComponentOne binaries (DLLs, EXEs). For **Component Book 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
- C1.WPF.Extended.dll
- C1.WPF.Extended.Expression.Design.dll
- C1.WPF.Extended.VisualStudio.Design.dll
- C1.WPF.Extended.Expression.Design.4.dll
- C1.WPF.Extended.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 C1Book 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\\<version number>

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

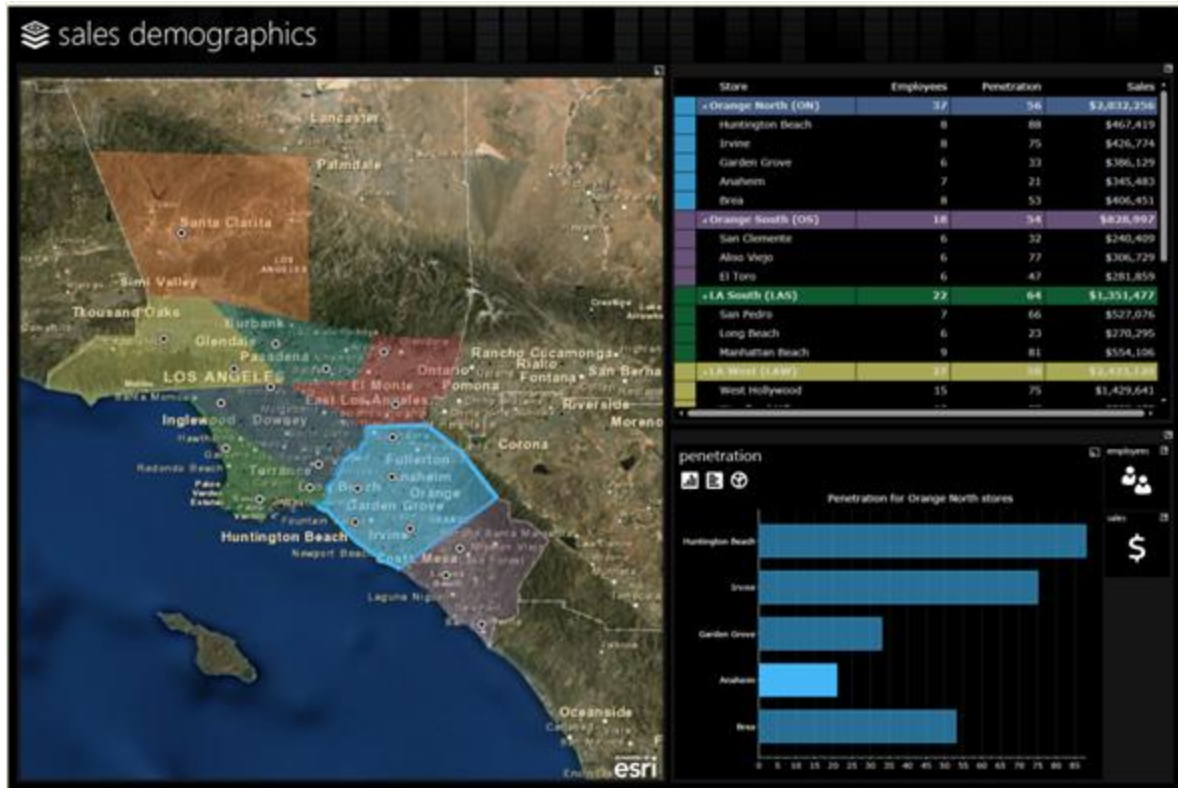
Easily transform GIS data into business intelligence with controls for Silverlight, WPF, and Windows Phone powered by Esri® software.

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.



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:**

Book for WPF includes design-time support for Expression
Blend.

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

Installing Demonstration Versions

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

To uninstall **ComponentOne Book 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 Book 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 Book 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
- C1.WPF.Extended.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 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 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
<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 C1Book 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 **C1.WPF**. This is a unified namespace; once this is in the project, all ComponentOne WPF controls found in your references will be accessible through XAML (and IntelliSense). Note that you still need to add references to the assemblies for each control you need to use.

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

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

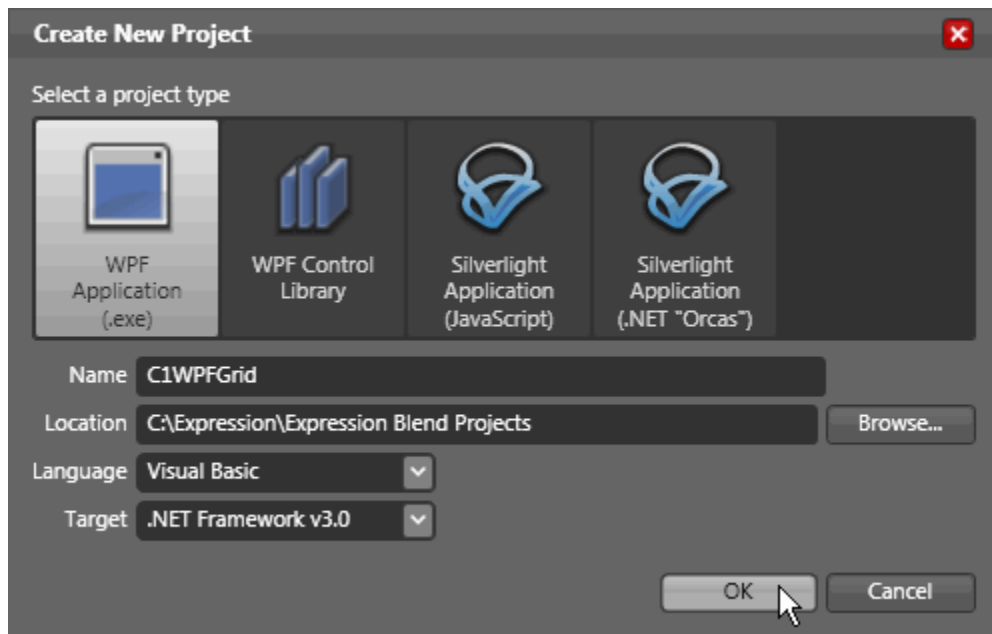
You can now use your custom namespace when assigning properties, methods, and events. For example, use the following XAML to add a border around the Book:

```
<MyNB: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**.

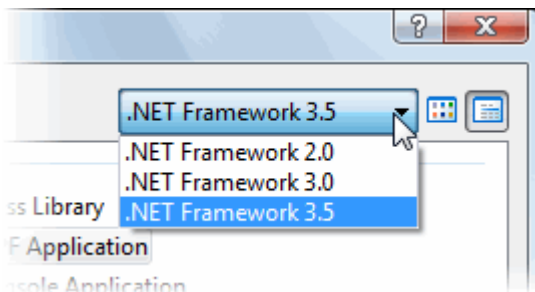


A new Blend project with a XAML window is created.

Creating a .NET Project in Visual Studio

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

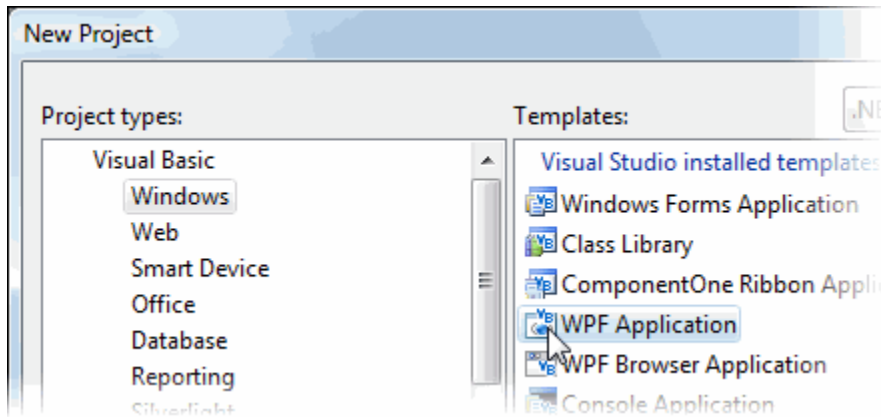
1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**.
The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



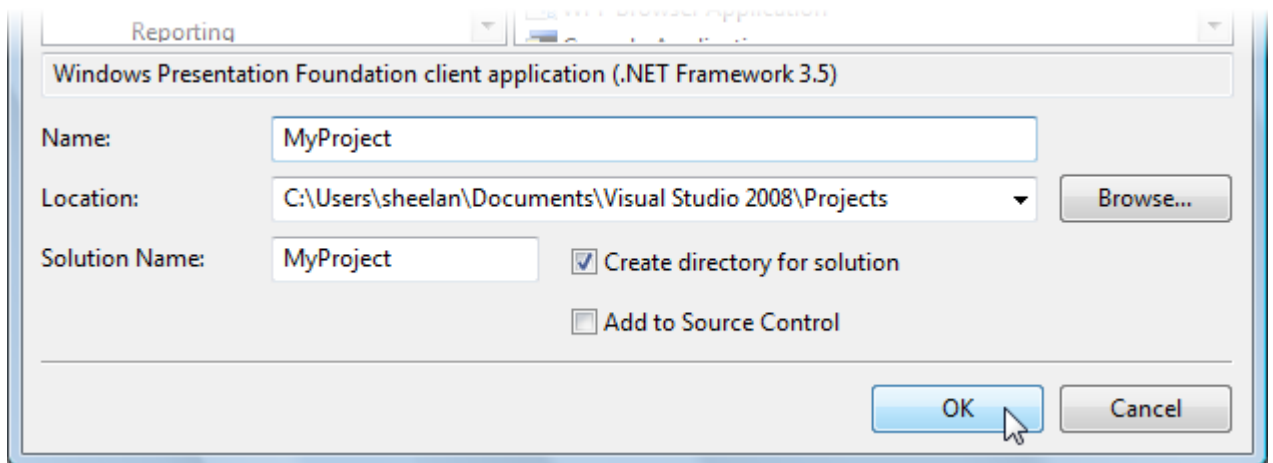
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

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.

Adding the Book for WPF Components to a Blend Project

In order to use C1Book or another **ComponentOne Book for WPF** component in the Design workspace of Blend, you must first add a reference to the **C1.WPF.Extended** 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.Extended.dll** assembly installed with **Book 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.Extended.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.Extended.dll** 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 **C1Book**. The component will appear in the Toolbox below the **Asset Library** button.
4. Double-click the **C1Book** component in the Toolbox to add it to **Window1.xaml**.

Adding the Book for WPF Components to a Visual Studio Project

When you install **ComponentOne Book for WPF** the C1Book control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

ComponentOne Book for WPF provides the following control:

- C1Book

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

Manually Adding Book for WPF to the Toolbox

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

- C1Book

To manually add the C1Book 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 **Book for WPF** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1Book**, 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.Extended** namespace. Note that there may be more than one component for each namespace.

Adding Book for WPF to the Window

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

1. Add the C1Book control to the Visual Studio Toolbox.
2. Double-click C1Book or drag the control onto the window.

Adding a Reference to the Assembly

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

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

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

Key Features

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

- **Familiar Book Metaphor**

C1Book enables you to present information innovatively using a familiar mental model – that of a book. But **Book for WPF** is not a typical static book, it's dynamic and interactive, and it takes the familiar metaphor further using WPF.

- **Real Book-like Visuals**

C1Book enables you to customize the look and feel of the book pages; for example, show page folds and display inner and outer shadows. It not only looks like a book, but can be interacted with like a book.

- **Flexible Data Binding**

C1Book is an **ItemsControl**, so you can bind it to any data source. Each item in the data source can be a **UIElement** or a generic object that gets converted into a **UIElement** using templates.

- **Custom Styles for Book Pages and Cover**

C1Book supports different templates for odd and even pages, and it is possible to define custom pages like the cover.

Book for WPF Quick Start

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

You'll create a WPF application using a **C1Book** control that contains a variety of content, you'll add a **C1Book** control to the application, customize and add content to the **Book**, and observe some of the run-time interaction possible with **Book for WPF**.

Step 1 of 3: Creating the Book Application

In this step you'll create a WPF application using **Book for WPF**. When you add a **C1Book** control to your application, you'll have a complete, functional book-like interface that you can add images, controls, and other elements to. To set up your project and add a **C1Book** control to your application, complete the following steps:

1. Create a new WPF project in Visual Studio. For more information about creating a WPF project, see [Creating a .NET Project in Visual Studio](#) (page 12).
2. In the Solution Explorer, right-click the **References** item and choose **Add Reference**. Select the **C1.WPF**, **C1.WPF.Extended**, and **WPFToolkit** assemblies and click **OK** to add references to your project.
3. Navigate to the Visual Studio Toolbox, and double-click the **C1Book** icon to add the control to the window.
4. Resize the window and reposition the **C1Book** control in the window.
5. Click once on the **C1Book** control in Design view, navigate to the Properties window, and set the following properties:
 - Set **Width** to "450" and **Height** to "300".
 - Set **HorizontalAlignment** and **VerticalAlignment** to **Center** to center the control in the panel.
 - Check the **IsFirstPageOnTheRight** check box to set the first page to appear on the right side.
 - Set the **TurnInterval** property to 600 to increase the time taken for the page turn animation.

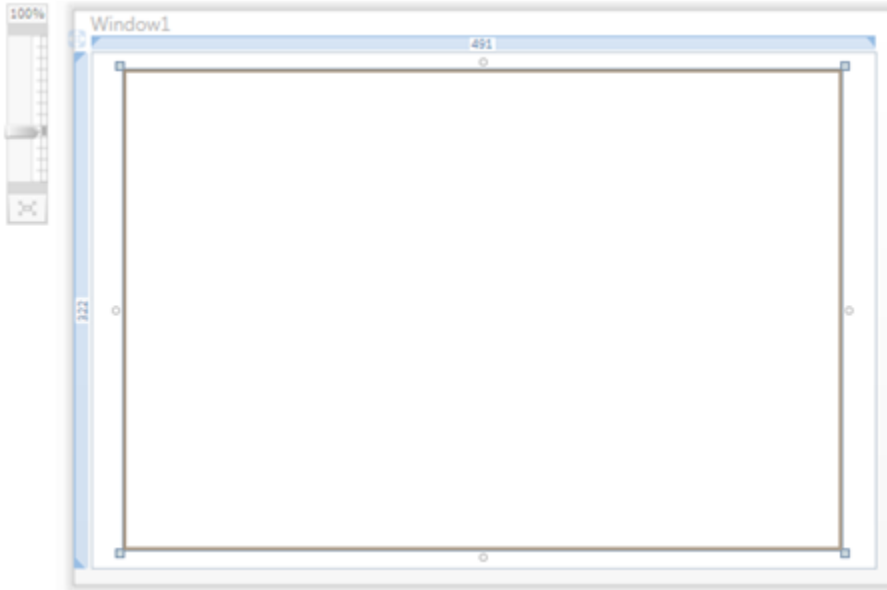
The XAML will appear similar to the following:

```
<c1:C1Book Name="C1Book1" Width="450" Height="300"
VerticalAlignment="Center" HorizontalAlignment="Center"
IsFirstPageOnTheRight="True" TurnInterval="600" />
```

6. Update the **C1Book** control's markup in XAML view, so that it includes a closing tag as in the following:

```
<c1:C1Book Name="C1Book1" Width="450" Height="300"
VerticalAlignment="Center" HorizontalAlignment="Center"
IsFirstPageOnTheRight="True" TurnInterval="600"></c1:C1Book>
```

The page's Design view should now look similar to the following image (with the **C1Book** control selected on the form):



You've successfully set up your application's user interface, but the `C1Book` control currently contains no content. In the next step you'll add content to the `C1Book` control, and then you'll add code to your application to add functionality to the control.

Step 2 of 3: Adding Content to the Book Control

In this step you'll add content to the `C1Book` control in design-time, XAML markup, and code. You'll add standard Microsoft controls and content to create a virtual book with several pages that can be turned. To customize your project and add content to the `C1Book` control in your application, complete the following steps:

1. Click once on the `C1Book` control to select it.
2. Navigate to the Toolbox and double-click the **TextBlock** control to add it to the project.
3. In XAML view move the **TextBlock**'s tag so it is within the `C1Book` control's tags.
4. Select the **TextBlock** in Design view, navigate to the Properties window, and set the following properties:
 - **Text** to "Hello World!"
 - **HorizontalAlignment** to **Center**
 - **VerticalAlignment** to **Center**
5. Switch to XAML view and add two button controls in the markup just after the **TextBlock**. The markup will appear like the following:

```
<c1:C1Book Name="C1Book1" Width="450" Height="300"
VerticalAlignment="Center" HorizontalAlignment="Center"
IsFirstPageOnTheRight="True" TurnInterval="600">
  <TextBox Height="23" HorizontalAlignment="Center" Margin="10,0,0,102"
Name="TextBox1" VerticalAlignment="Center" Width="120">Hello
World!</TextBox>
  <Button x:Name="Button1" Content="Last" Height="100" Width="100"
Click="Button1_Click"/>
  <Button x:Name="Button2" Content="Next" Width="150" Height="150"
Click="Button2_Click"/>
</c1:C1Book>
```

This will give the buttons names so they are accessible in code, size the controls, and add event handlers that you will add code for in the next steps.

6. In Design view, double-click the window to switch to Code view.
7. In Code view, add the following import statements to the top of the page:

- Visual Basic

```
Imports Cl.WPF
Imports Cl.WPF.Extended
```

- C#

```
using Cl.WPF;
using Cl.WPF.Extended;
```

8. Add the following code just after the page's constructor to add handlers to the **Click** events:

- Visual Basic

```
Private Sub Button1_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    Me.ClBook1.CurrentPage = Me.clbook1.CurrentPage - 1
End Sub
Private Sub Button2_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    Me.ClBook1.CurrentPage = Me.clbook1.CurrentPage + 2
End Sub
```

- C#

```
private void Button1_Click(object sender,
System.Windows.RoutedEventArgs e)
{
    this.clBook1.CurrentPage = this.clBook1.CurrentPage - 1;
}
private void Button2_Click(object sender,
System.Windows.RoutedEventArgs e)
{
    this.clBook1.CurrentPage = this.clBook1.CurrentPage + 1;
}
```

The buttons will now allow the user to navigate to the last or next page at run time.

9. Add code to the **Window_Loaded** event so that it appears similar to the following:

- Visual Basic

```
Private Sub Window1_Loaded(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles MyBase.Loaded
    Dim txt1 As New TextBlock
    txt1.VerticalAlignment = VerticalAlignment.Center
    txt1.HorizontalAlignment = HorizontalAlignment.Center
    txt1.Text = "The End."
    ClBook1.Items.Add(txt1)
End Sub
```

- C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    TextBlock txt1 = new TextBlock();
    txt1.VerticalAlignment = VerticalAlignment.Center;
    txt1.HorizontalAlignment = HorizontalAlignment.Center;
    txt1.Text = "The End.";
    clBook1.Items.Add(txt1);
}
```

```
}
```

This will add a **TextBlock** to the C1Book control in code.

10. Save your project and return to XAML view.

11. In XAML view, add the following markup just after the `<Button x:Name="Button2"/>` tag:

- XAML to add

```
<Grid x:Name="checkers" Background="White" ShowGridLines="True">
  <Grid.RowDefinitions>
    <RowDefinition Height=".25*" />
    <RowDefinition Height=".25*" />
    <RowDefinition Height=".25*" />
    <RowDefinition Height=".25*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width=".25*" />
    <ColumnDefinition Width=".25*" />
    <ColumnDefinition Width=".25*" />
    <ColumnDefinition Width=".25*" />
  </Grid.ColumnDefinitions>
  <Rectangle Fill="Red" Grid.Row="0" Grid.Column="0" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5" />
  <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5" />
  <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5" />
  <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5" />
  <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5" />
  <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5" />
  <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5" />
  <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5" />
</Grid>
<Grid x:Name="checkers2" Background="White" ShowGridLines="True">
  <Grid.RowDefinitions>
    <RowDefinition Height=".25*" />
    <RowDefinition Height=".25*" />
    <RowDefinition Height=".25*" />
    <RowDefinition Height=".25*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width=".25*" />
    <ColumnDefinition Width=".25*" />
    <ColumnDefinition Width=".25*" />
    <ColumnDefinition Width=".25*" />
  </Grid.ColumnDefinitions>
  <Rectangle Fill="Red" Grid.Row="0" Grid.Column="0" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5" />
  <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5" />
  <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5" />
  <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5" />
```

```

<Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5" />
<Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5" />
<Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5" />
<Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5" />
<Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5" />
<Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5" />
<Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5" />
<Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5" />
<Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5" />
<Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5" />
</Grid>

```

This markup will add two Grids with multiple Rectangle elements. This markup demonstrates how you can add multiple controls to a page of the C1Book control as long as the controls are all in one panel, such as a Grid or StackPanel.

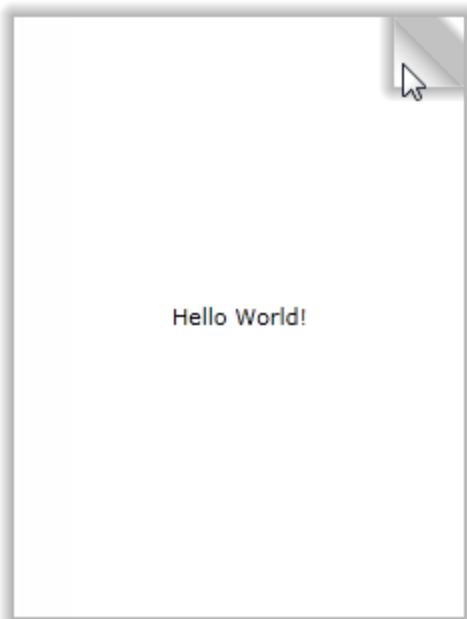
In this step you completed adding content to the C1Book control. In the next step you'll run the application and observe run-time interactions.

Step 3 of 3: Running the Book 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 **Book for WPF**'s run-time behavior, complete the following steps:

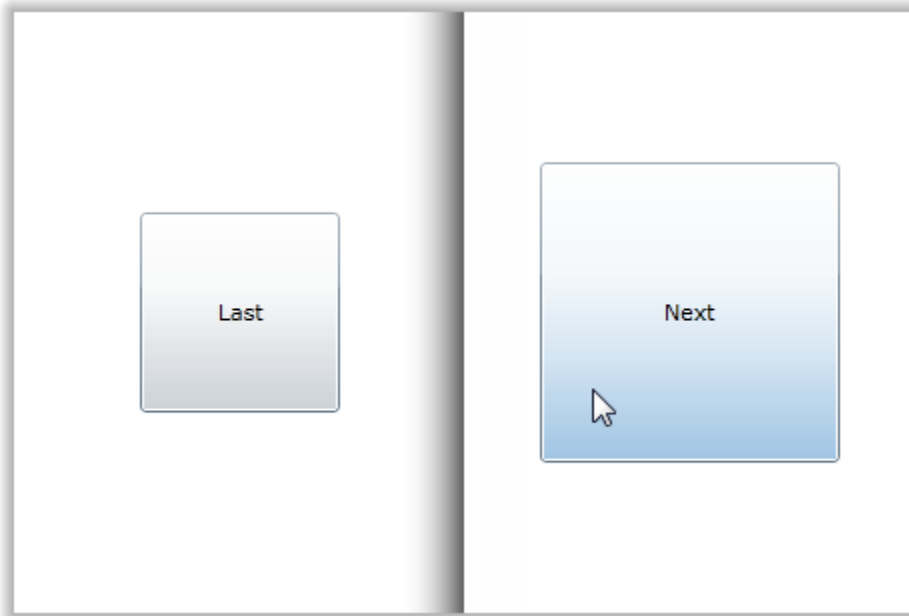
1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.

The application will appear similar to the following:



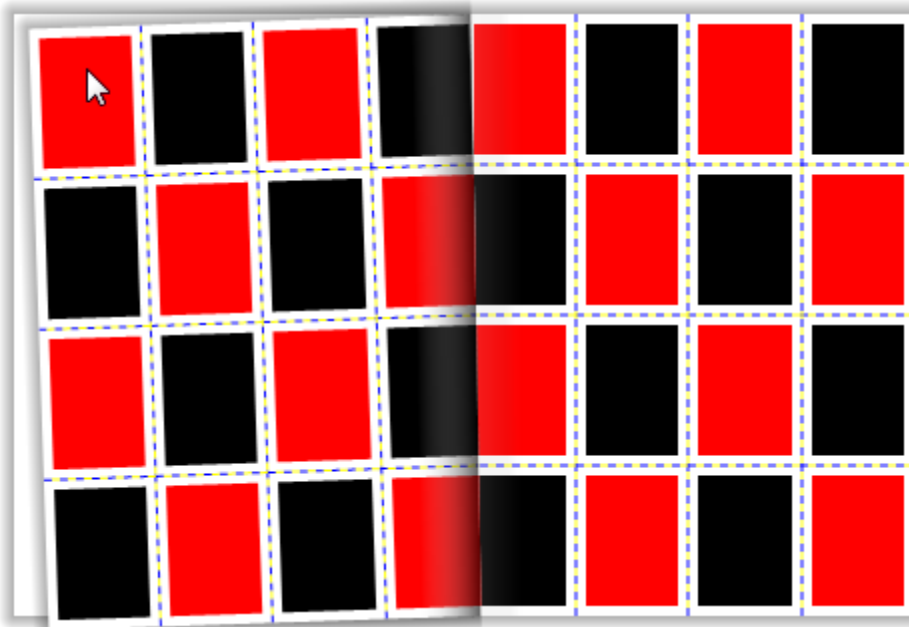
You set the `IsFirstPageOnTheRight` property so that only one page is initially visible. Notice that when you hover over the lower or upper-right corner of the C1Book control the page folds back slightly to prompt you to turn the page; see [Book Zones](#) (page 26) for more information.

2. Click the upper-right corner of the page and notice that the page turns and the second and third pages are visible:



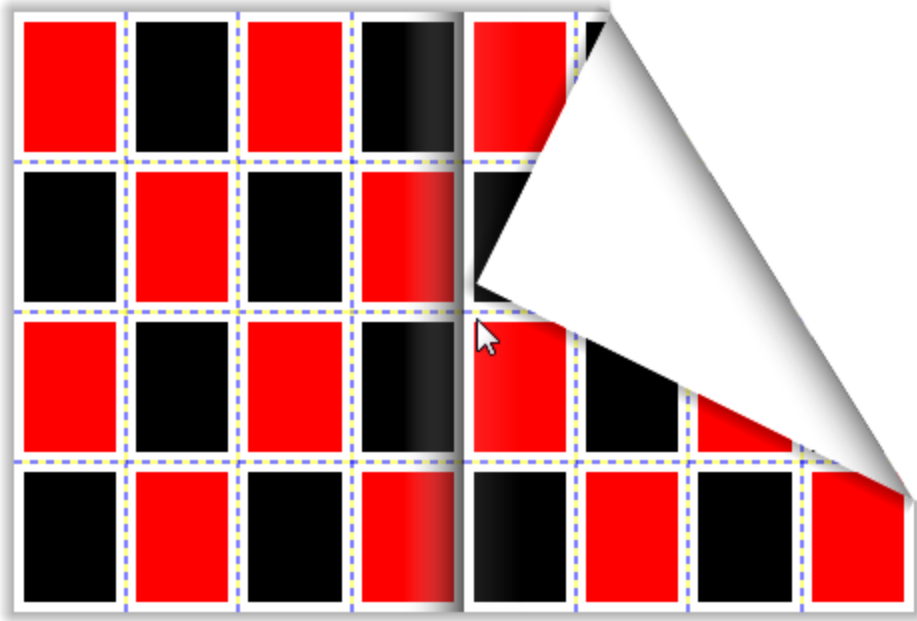
You customized the time it takes for the page to turn by setting the TurnInterval property.

3. Click the **Next** button. The next pages are displayed:

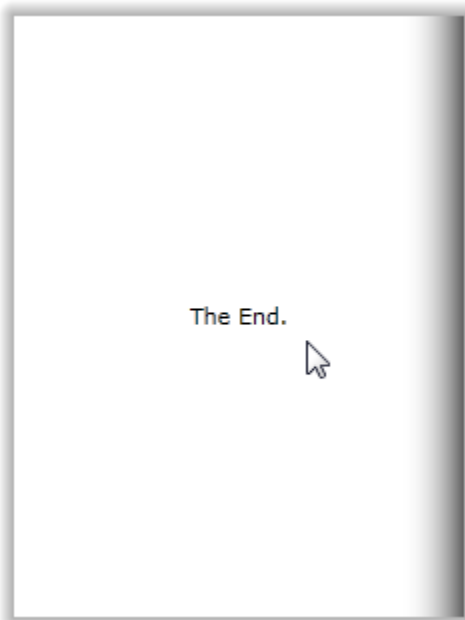


Note that you can return to the previous page by clicking the left top or bottom corner.

4. Click and drag the top-right page corner to the left to turn to the next page:



Notice that the last page contains the **TextBlock** content that was added in code:

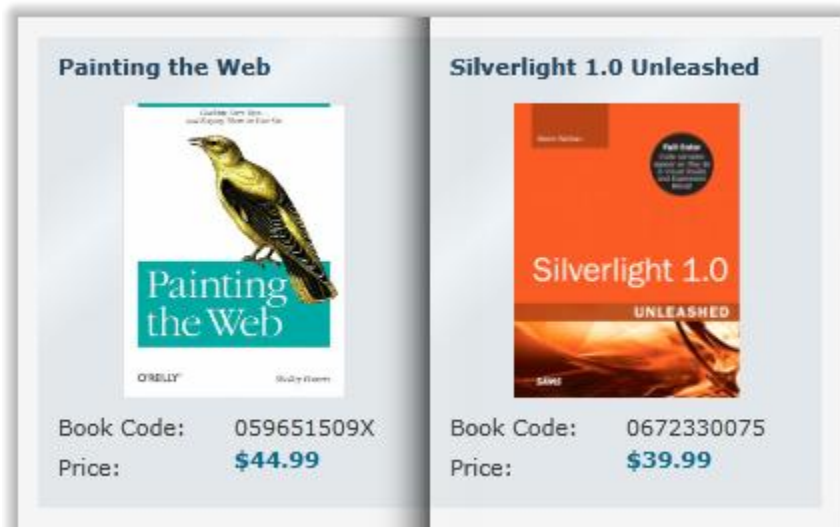


Congratulations! You've completed the **Book for WPF** quick start and created a simple WPF application, added and customized a **Book for WPF** control, and viewed some of the run-time capabilities of the control.

Working with Book for WPF

ComponentOne Book for WPF includes the C1Book control, a simple book control that acts as a container, allowing you to add controls, images, and more in a familiar book format. When you add the C1Book control to a XAML window, it exists as a container, similar to a panel, that can be customized and include added content.

The control's interface looks similar to the following image:



XBAP Support

While most **Studio for WPF** products are fully functional in an XAML Browser Application (XBAP), **ComponentOne Book for WPF** does not include XBAP functionality in a partial trust environment. **Book for WPF** uses shader effects for the shadows in the book display and shader effects are unfortunately not allowed in a WPF partial trust environment.

To run an application as full trust, you can open the application's properties, and select **This is a full trust application** from the **Security** tab. For full instructions, see <http://blogs.microsoft.co.il/blogs/maxim/archive/2008/03/05/wpf-xbap-as-full-trust-application.aspx>.

Basic Properties

ComponentOne Book 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 [Book for WPF Appearance Properties](#) (page 32) for more information about properties that control appearance.

The following properties let you customize the C1Book control:

Property	Description
CurrentPage	Gets or sets the current page that is displayed.
CurrentZone	Gets the zone of the book under the mouse. See Book Zones (page 26) for more information.
FoldSize	Gets or sets the size of the fold (in pixels). See Page Fold

	Size (page 28) for more information.
IsFirstPageOnTheRight	Gets or sets whether the first page of the book is displayed in the right side of the book. See First Page Display (page 30) for more information.
LeftPageTemplate	Gets or sets the template of the page shown in the left side of the book.
PageFoldAction	Gets or sets the action that will raise the turn animation. See Page Turning Options (page 29) for more information.
RightPageTemplate	Gets or sets the template of the page shown in the right side of the book.
ShowInnerShadows	Gets or sets whether the inner shadows are shown. See Page Shadows (page 31) for more information.
ShowOuterShadows	Gets or sets whether the outer shadows are shown. See Page Shadows (page 31) for more information.
ShowPageFold	Gets or sets whether the fold is displayed. See Page Fold Visibility (page 29) for more information.
TurnInterval	Gets or sets the amount (in milliseconds) of time of the turn animation.

Basic Events

ComponentOne Book for WPF includes several events that allow you to set interaction and customize the control. Some of the more important events are listed below.

The following events let you customize the C1Book control:




Event	Description
CurrentPageChanged	Fires when the CurrentPage property changes.
CurrentZoneChanged	Occurs when current zone changed. For more information, see Book Zones (page 26).
DragPageFinished	Occurs when page ends of being dragged.
DragPageStarted	Occurs when the page starts to be dragged.
IsMouseOverChanged	Event raised when the IsMouseOver property has changed.


Book Zones

The C1Book control includes several zones. These zones let you customize what happens when users interact with various sections of the control. You can use the CurrentZone property to get the user's current zone and you can use the CurrentZoneChanged event to customize when happens when users move to a different zone.

There are six separate zones in the C1Book control. For an illustration of each zone, note the mouse's position in each of the images in the following table:

Zone	Description	Example
------	-------------	---------

<p>Out</p>	<p>Specifies the zone outside the borders of the book.</p>	
<p>BottomLeft</p>	<p>Specifies bottom left fold zone.</p>	
<p>TopLeft</p>	<p>Specifies top left fold zone.</p>	

Center	Specifies the center of the book (no fold zone).	
TopRight	Specifies top right fold zone.	
BottomRight	Specifies bottom left fold zone.	

Page Fold Size

One way to customize the appearance of the book as users flip pages is by setting the size of the page fold using the `FoldSize` property. Page folds, which appear when users mouse over certain [book zones](#) (page 26), serve as a cue to users that a page can be turned.

When you set the `FoldSize` property you will be setting the size of all of the page folds – this includes the right top and bottom folds and the left top and bottom folds. So for example, when `FoldSize` is **40**, the bottom left and bottom right folds appear similar to the following image:



If set to a higher number, the folds will appear more prominent. When FoldSize is **80**, the bottom left and bottom right folds appear similar to the following image:



Page Fold Visibility

By default, users will see a page fold when their mouse is over certain [book zones](#) (page 26). If you choose to, however, you can change the page fold visibility. You can set the ShowPageFold property to any of the values in the following table to determine how users interact with the C1Book control:

Value	Description
OnMouseOver	The fold will be visible when the user drags the mouse over the edge of the page. This is the default setting.
Never	The fold will not be visible.
Always	The fold will always be visible.

Page Turning Options

By default, when users click once on a page fold the book will progress to the previous or next page. You can customize how pages turn on page click using the PageFoldAction property. For example you can set

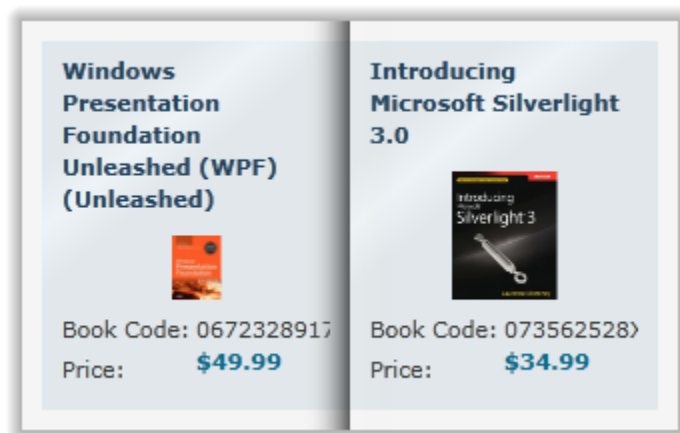
PageFoldAction so that users must double-click on the page fold to turn the page, or you can prevent page turning on mouse click altogether, requiring that users perform a drag-and-drop operation on the page fold to turn a page.

You can set the PageFoldAction property to any of the values in the following table to determine how users interact with the C1Book control:

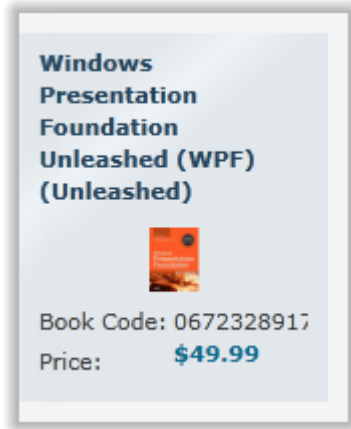
Value	Description
TurnPageOnClick	Turn the page when the user clicks the page fold.
TurnPageOnDoubleClick	Turn the page when the user double clicks the page fold.
None	Turn page when user drags the page fold across the book.

First Page Display

By default, the first page in the C1Book control is displayed on the left hand side. This makes it appear as if the book is open:



If you choose, however, you can change the display of the first page to appear on the right by setting the IsFirstPageOnTheRight property to **True**. When the first page is set to display on the right side, it will appear similar to a cover, as if the book is closed:



Page Shadows

The C1Book control includes shadows that give the control a three-dimensional appearance. The control includes inner and outer shadows. Inner shadows appear in the center of the book, and behind the right page when turning. Outer shadows appear around the outside of the control and behind the left page when turning. For example, the following image illustrates inner and outer shadows:



If you do not want shadows to be displayed, you can change their visibility by setting the `ShowInnerShadows` and `ShowOuterShadows` properties to **False**.

Book Navigation

At run time users can navigate through the C1Book control using the mouse. Users can click in one of the [book zones](#) (page 26) or can perform a drag-and-drop operation to turn the page. The C1Book control includes navigation-related methods, properties, and events to make it easier for you to determine what page a user is currently viewing, and to set the application's actions as users navigate through a book.

The `CurrentPage` property gets or sets the page that is currently displayed at run time. Note that when you turn a page, the page displayed on the left of the two-page spread will be the `CurrentPage`. Page numbering begins with **0**

and page 0 is always displayed on the left. So if `IsFirstPageOnTheRight` property is set to **True**, the first initial page of the book displayed on the right side will be page 1 with a hidden page 0 on the left side.

You can set the displayed page using the `CurrentPage` property, but you can also use the `TurnPage` method to change the current page at run time. The `TurnPage` method turns to book pages forward or back one page.

You can use the `CurrentPageChanged` event to specify actions that happen when the current page is changed. You can also use the `DragPageStarted` and `DragPageFinished` events to specify actions to take when the user turns the page using a drag-and-drop operation.

Book for WPF Layout and Appearance

The following topics detail how to customize the `C1Book` 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 lay out the control and to customize the control's actions.

Book for WPF Appearance Properties

ComponentOne Book for WPF includes several properties that allow you to customize the appearance of the control. You can change the color, border, and height of the control. The following topics describe some of these appearance properties.

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.

Alignment Properties

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

Property	Description
HorizontalAlignment	Gets or sets the horizontal alignment characteristics applied to this element when it is composed within a parent element, such as a panel or items control. This is a dependency property.
VerticalAlignment	Gets or sets the vertical alignment characteristics applied to this element when it is composed within a parent element such as a panel or items control. 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 control:

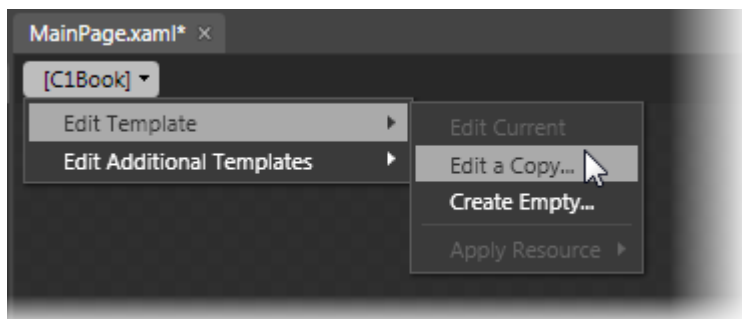
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.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

Book 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 Book 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 C1Book 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.

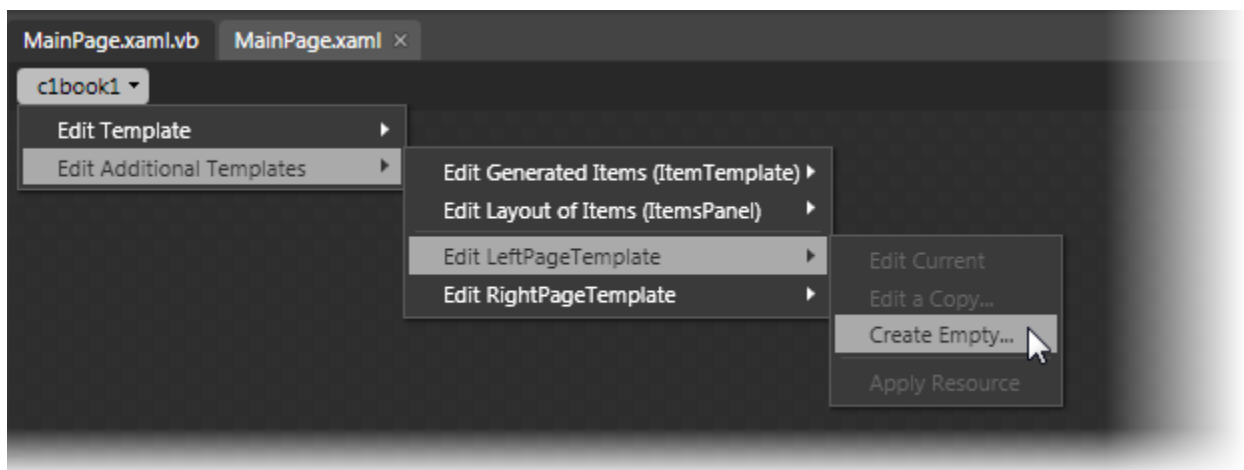
Note that you can use the [Template](#) property to customize the template.

Page Templates

ComponentOne Book for WPF's book control includes two page templates: the `LeftPageTemplate` and the `RightPageTemplate`. These templates control the appearance and layout of the left and right pages of the book. These templates function as master pages -- items that you add to these page templates will appear on every page that template effects. This is useful, for example, if you want to add a watermark or company logo to every page without adding it to every single page in the book individually.

Accessing Page Templates

You can access page templates in Microsoft Expression Blend by selecting the `C1Book` control and, in the menu, selecting **Edit Additional Templates**. Choose **Edit LeftPageTemplate** or **Edit RightPageTemplate** and select **Create Empty** to create a new blank template.



The **Create ControlTemplate Resource** dialog box will appear allowing you to name the template and determine where to define the template. By default, the template will appear blank with an empty Grid control:

```
<ControlTemplate x:Key="NewLeftPageTemplate">
    <Grid/>
</ControlTemplate>
```

You can customize the template as you would any other **ControlTemplate**.

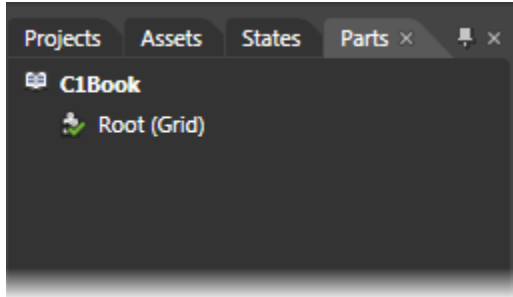
Book Styles

ComponentOne Book for WPF's `C1Book` control provides several style properties that you can use to change the appearance of the control. Some of the included styles are described in the table below:

Style	Description
FontStyle	Gets or sets the font style. This is a dependency property.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

Book Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the C1Book control to select it and choose **Object | Edit Template | Edit a Copy**). Once you've created a new template, the parts of the template will appear in the **Parts** window:



Note that you may have to select the **ControlTemplate** for its parts to be visible in the **Parts** window.

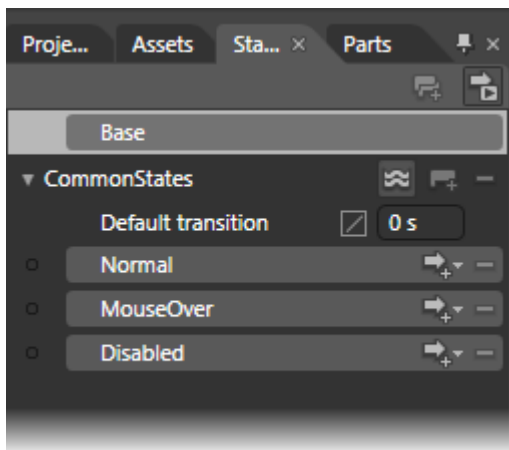
In the Parts window, you can double-click any element to create that part in the template. Once you have done so, the part will appear in the template and the element's icon in the **Parts** pane will change to indicate selection.

Template parts available in the C1Book control include:

Name	Type	Description
Root	FrameworkElement	Provides a framework of common APIs for objects that participate in WPF layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in WPF.

Book Visual States

In Microsoft Expression Blend, you can add custom states and state groups to define a different appearance for each state of your user control – for example, the visual state of the control could change on mouse over. You can view and edit visual states by creating a new template and [adding a new template part](#) (page 35). Once you've done so the available visual states for that part will be visible in the **Visual States** window:



Common states include **Normal** for the normal appearance of the item, **MouseOver** for the item on mouse over, and **Disabled** for when the item is not enabled. Focus states include **Unfocused** for when the item is not in focus and **Focused** when the item is in focus.

Book 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 Book page in the ControlExplorer sample demonstrates how to add content to and customize the C1Book control.

Book for WPF Task-Based Help

The following task-based help topics assume that you are familiar with Visual Studio and Expression Blend and know how to use the C1Book control in general. If you are unfamiliar with the **ComponentOne Book for WPF** product, please see the [Book for WPF Quick Start](#) (page 17) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Book for WPF** product. Most task-based help topics also assume that you have created a new WPF project and added a C1Book control to the project – for information about creating the control, see [Creating a Book](#) (page 37).

Creating a Book

You can easily create a C1Book control at design time, in XAML, and in code. Note that if you create a C1Book control as in the following steps, it will appear as an empty container. You will need to add items to the control for it to appear as a book at run time. For an example, see [Adding Items to a Book](#) (page 39).

At Design Time

To create a C1Book control, complete the following steps:

1. Click once on the design surface of the Window to select it.
2. Double-click the **C1Book** icon in the Toolbox to add the control to the panel. The C1Book control will now exist in your application.
3. If you choose, you can customize the control by selecting it and setting properties in the Properties window.

In XAML

To create a C1Book control using XAML markup, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select the **C1.WPF.dll** and **C1.WPF.Extended.dll** assemblies, and click **OK**.
2. Add a XAML namespace to your project by adding `xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"` to the initial `<Window>` tag. It will appear similar to the following:

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
x:Class="Window1" Title="Window1" Height="230" Width="348">
```

3. Add a `<c1:C1Book>` tag to your project within the `<Grid>` tag to create a C1Book control. The markup will appear similar to the following:

```
<Grid>
  <c1:C1Book x:Name="C1Book1" Height="300" Width="450"/>
</c1:C1Book>
</Grid>
```

This markup will create an empty C1Book control named "C1Book1" and set the control's size.

In Code

To create a C1Book control in code, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select the **C1.WPF.dll** and **C1.WPF.Extended.dll** assemblies, and click **OK**.
2. In XAML view, give the initial grid in the window a name, by updating the tag so it appears similar to the following:

```
<Grid x:Name="LayoutRoot">
```

3. Right-click within the **Window1.xaml** window and select **View Code** to switch to Code view.
4. Add the following import statements to the top of the page:

- Visual Basic

```
Imports C1.WPF
Imports C1.WPF.Extended
```

- C#

```
using C1.WPF;
using C1.WPF.Extended;
```

5. Add code to the page's constructor to create the C1Book control. It will look similar to the following:

- Visual Basic

```
Public Sub New()
  InitializeComponent()
  Dim clbook1 as New C1Book
  clbook1.Height = 300
  clbook1.Width = 450
  LayoutRoot.Children.Add(clbook1)
End Sub
```

- C#

```
public MainPage()
{
  InitializeComponent();
  C1Book clbook1 = new C1Book();
  clbook1.Height = 300;
  clbook1.Width = 450;
  LayoutRoot.Children.Add(clbook1);
}
```

This code will create an empty C1Book control named "clbook1", set the control's size, and add the control to the page.

What You've Accomplished

You've created a `C1Book` control. Note that when you create a `C1Book` control as in the above steps, it will appear as an empty container. You will need to add items to the control for it to appear as a book at run time. For an example, see [Adding Items to a Book](#) (page 39).

Adding Items to a Book

You can add any sort of arbitrary content to a `C1Book` control. This includes text, images, layout panels, and other standard and 3rd-party controls. In this example, you'll add a `TextBlock` control to a `C1Book` control, but you can customize the steps to add other types of content instead.

At Design Time

To add a `TextBlock` control to the book, complete the following steps:

1. Click the `C1Book` control once to select it.
2. Navigate to the Toolbox, and double-click the `TextBlock` item to add the control to the `C1Book` control.
3. If you choose, you can customize the `C1Book` and `TextBlock` controls by selecting each control and setting properties in the Properties window. For example, set the `TextBlock`'s `Text` property to "Hello World!".

In XAML

For example, to add a `TextBlock` control to the book add `<TextBlock Text="Hello World!"/>` within the `<c1:C1Book>` tag so that it appears similar to the following:

```
<c1:C1Book x:Name="C1Book1" Height="300" Width="450">
  <TextBlock Text="Hello World!"/>
</c1:C1Book>
```

In Code

For example, to add a `TextBlock` control to the book, add code to the page's constructor so it appears like the following:

- Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim txt1 as New TextBlock
    txt1.Text = "Hello World!"
    C1Book1.Items.Add(txt1)
End Sub
```

- C#

```
public MainPage()
{
    InitializeComponent();
    TextBlock txt1 = new TextBlock();
    txt1.Text = "Hello World!";
    c1Book1.Items.Add(txt1);
}
```

What You've Accomplished

You've added a control to the `C1Book` control. Run the application and observe that the `TextBlock` control has been added to the `C1Book` control. You can similarly add other content and controls.

Clearing Items in a Book

You may choose to allow users to clear all items from the C1Book control at run time, or you may need to clear the items collection when binding and then rebinding the control to another data source.

For example, to clear the book's content add the following code to your project:

- Visual Basic

```
Me.C1Book1.Items.Clear()
```
- C#

```
this.c1Book1.Items.Clear();
```

What You've Accomplished

The control's content will be cleared. If you run the application, you will observe that the book is blank.

Displaying the First Page on the Right

The `IsFirstPageOnTheRight` property gets or sets if the first page of the book is displayed on the right or the left side. See [First Page Display](#) (page 30) for more information. By default the C1Book control starts with the first page displayed on the left and two pages displayed, but you can customize this by setting the `IsFirstPageOnTheRight` property at design time, in XAML, and in code.

At Design Time

To set the `IsFirstPageOnTheRight` property at design time, complete the following steps:

1. Click the C1Book control once to select it.
2. Navigate to the Properties window and check the check box next to the **IsFirstPageOnTheRight** item.

In XAML

For example, to set the `IsFirstPageOnTheRight` property, add `IsFirstPageOnTheRight="True"` to the `<c1:C1Book>` tag so that it appears similar to the following:

```
<c1:C1Book x:Name="C1Book1" Height="300" Width="450"
IsFirstPageOnTheRight="True">
```

In Code

For example, to set the `IsFirstPageOnTheRight` property, add the following code to your project in the page's constructor:

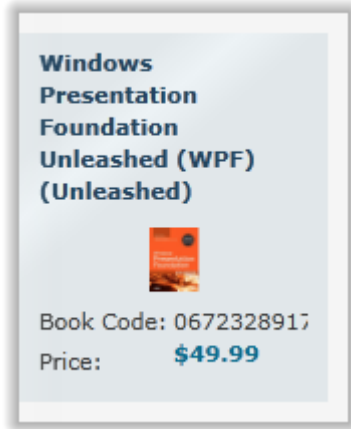
- Visual Basic

```
Me.C1Book1.IsFirstPageOnTheRight = True
```
- C#

```
this.c1Book1.IsFirstPageOnTheRight = true;
```

What You've Accomplished

You've set the first page to appear on the right. If you run the application, the first page will appear as a single page, like the book's cover:



Setting the Initial Page

The `CurrentPage` property gets or sets the value of the `C1Book` control's current page. By default the `C1Book` control starts with the first page displayed but you can customize this by setting the `CurrentPage` property at design time, in XAML, and in code.

At Design Time

To set the `CurrentPage` property to **3** at design time, complete the following steps:

1. Click the `C1Book` control once to select it.
2. Navigate to the Properties window and click in the text box next to the **CurrentPage** item.
3. Enter a number, for example "3", for the displayed initial page.

In XAML

For example, to set the `CurrentPage` property to **3**, add `CurrentPage="3"` to the `<c1:C1Book>` tag so that it appears similar to the following:

```
<c1:C1Book x:Name="C1Book1" Height="300" Width="450" CurrentPage="3">
```

In Code

For example, to set the `CurrentPage` property to **3**, add the following code to your project:

- Visual Basic

```
Me.C1Book1.CurrentPage = 3
```
- C#

```
this.c1Book1.CurrentPage = 3;
```

What You've Accomplished

You've changed the book's initial starting page. If you run the application, the initial page that appears will be page 3.

Navigating the Book with Code

You can set the displayed page using the `CurrentPage` property, but you can also use the `TurnPage` method to change the current page at run time. For more information, see [Book Navigation](#) (page 31). In this topic you'll add two buttons to your application, one that will turn to the previous page and one that will turn to the next page of the book.

To add additional navigation to your book, complete the following steps:

1. Navigate to the Toolbox and double-click the **Button** item twice to add two **Button** controls to your application.
2. Select **Button1**, navigate to the Properties window and set the **Content** property to "<".
3. Select **Button2**, navigate to the Properties window and set the **Content** property to ">".
4. Resize and reposition the buttons in the Window. Place the **Button1** button to the left of the book, and the **Button2** button to the right of the book.
5. Double-click **Button1** to create the **Button_Click** event handler and switch to Code view.
6. Return to Design view and repeat the previous step with **Button2** so each button has a **Click** event specified.

The XAML markup will appear similar to the following:

```
<Button HorizontalAlignment="Right" Margin="0,43,12,0" Name="Button1"
Width="28" Height="23" VerticalAlignment="Top">&gt;</Button>
<Button Height="23" HorizontalAlignment="Left" Margin="12,43,0,0"
Name="Button2" VerticalAlignment="Top" Width="28">&lt;</Button>
```

7. Switch to Code view and add the following import statements to the top of the page:

- Visual Basic

```
Imports C1.WPF
Imports C1.WPF.Extended
```

- C#

```
using C1.WPF;
using C1.WPF.Extended;
```

8. Add code to the **Click** event handlers so they look like the following:

- Visual Basic

```
Private Sub Button1_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    Me.C1Book1.TurnPage(True)
End Sub

Private Sub Button2_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    Me.C1Book1.TurnPage(False)
End Sub
```

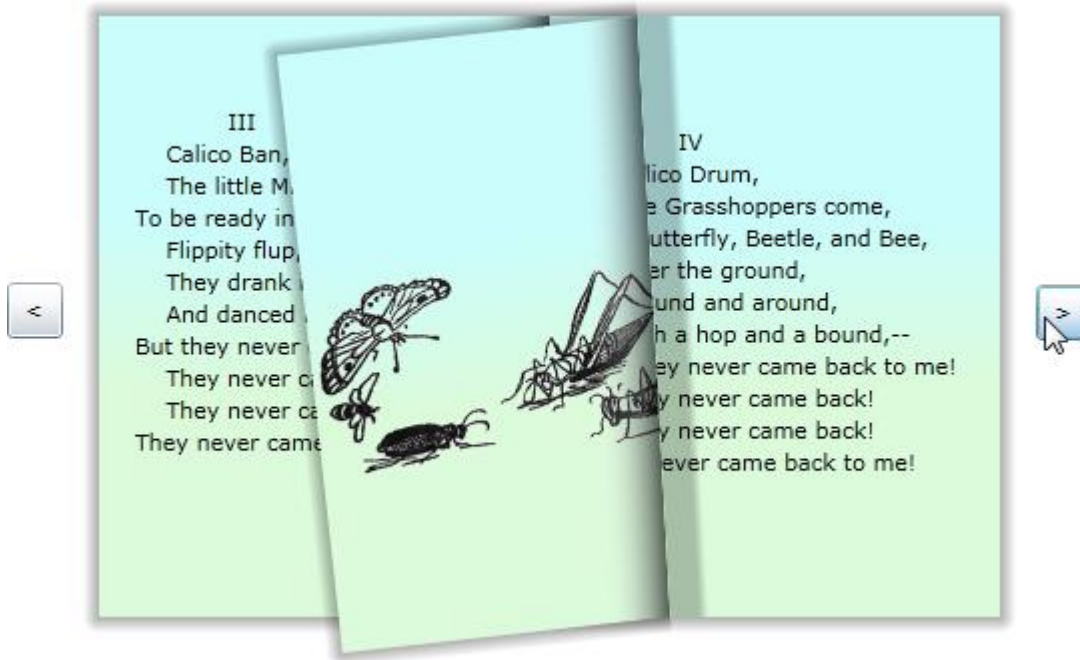
- C#

```
public MainPage()
{
    private void button1_Click(object sender,
System.Windows.RoutedEventArgs e)
    {
        this.c1Book1.TurnPage(true);
    }
    private void button2_Click(object sender,
System.Windows.RoutedEventArgs e)
    {
        this.c1Book1.TurnPage(false);
    }
}}
```

This code will turn the book a page forward or back depending on the button clicked.

What You've Accomplished

You've customized navigation in the book. To view the book's navigation, run the application and click the right button. Notice that the page turns to the next page with a page turning animation:



Click the left button and observe that the book returns to the previous page.