ComponentOne

ReportViewer for Silverlight

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third Floor

Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com

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

Sales

E-mail: sales@componentone.com

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

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

Table of Contents

ComponentOne ReportViewer for Silverlight Overview 1

Silverlight**SilverlightSilverlightSilverlight**SilverlightSilverlightHelp with ComponentOne Studio for Silverlight 1

Key Features 3

ReportViewer for Silverlight Quick Start 4

Step 1 of 3: Creating the C1ReportViewer Application 4

Step 2 of 3: Adding Content to the C1ReportViewer Control 5

Step 3 of 3: Running the C1ReportViewer Application 6

Working with ReportViewer for Silverlight 7

ReportViewer Elements 8

Basic Properties 8

Basic Events 9

ReportViewer for Silverlight Layout and Appearance 9

ReportViewer for Silverlight Appearance Properties 10

Color Properties 10

Alignment Properties 10

Border Properties 10

Size Properties 10

ReportViewer Templates 11

C1ReportViewer Styles and Templates 11

C1ReportViewer Visual States 12

Run-Time Interaction 12

ReportViewer Content Area 12

ReportViewer Toolbar 14

ReportViewer for Silverlight Task-Based Help 15

Adding C1ReportViewer to the Application 15

Loading Documents into C1ReportViewer 16

Loading Documents from Application Resources 16

Loading Documents from Files on the Client Machine 17

Loading Documents from Files on the Server 18
Creating and Loading Reports Dynamically 22
Hiding the Toolbar 23
Customizing the Toolbar 23

ComponentOne ReportViewer for Silverlight Overview

Add report viewing capabilities to your Silverlight applications. **ComponentOne ReportViewer for Silverlight™** can display HTML and PDF-based reports from virtually any report service, including Microsoft SQL Server Reporting Services and **C1Report**. This powerful viewer allows users to see, search, zoom, select, print and save the reports to local files.

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

Getting Started

Get started with the following topics:

- Key Features (page 3)
- Quick Start (page 4)
- Run-Time Interaction (page 12)

Silverlight**SilverlightSilverlightSilverlight**SilverlightSilverlightHelp with ComponentOne Studio for Silverlight

Getting Started

For information on installing **ComponentOne Studio for Silverlight**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for Silverlight.

What's New

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

Key Features

Add report viewing capabilities to your Silverlight applications. **ComponentOne ReportViewer for Silverlight™** can display HTML and PDF-based reports from virtually any report service, including Microsoft SQL Server Reporting Services and **C1Report**. This powerful viewer allows users to see, search, zoom, select, print and save the reports to local files.

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

- **View Reports from Multiple Sources**

  The **C1ReportViewer** control is engine-agnostic, because it supports the most common document formats: HTML and PDF. Use **C1ReportViewer** to display reports from virtually any report generator such as **C1Report**, Microsoft Reporting Services, Active Reports, Crystal or any other report provider capable of generating HTML or PDF output.

- **Load and Save Documents**

  In addition to viewing generated reports, you can also use **C1ReportViewer** to load arbitrary PDF and HTML documents. Users can also save files that were loaded.

- **Printing Support**

  C1ReportViewer allows users to print the current document in its entirety or print a selection of pages. Or print directly from code using the **PrintDocument** method.

- **Find Text**

  Users can perform text searches within the document. As matches are found they are brought into view, and users can navigate through search results in a quick and intuitive manner.

- **Multiple View Modes**

  C1ReportViewer features multiple viewing modes so you can view documents at any scale. Users can set the zoom level to fit the page into view. View just 1 page or view multiple pages side by side.

- **Page Customization**

  Specify page properties such as page size and margin thickness. You can even design a page template to provide custom headers and footers that are not generated as part of the report.

- **Customize the Toolbar**

  ReportViewer includes a default toolbar for quick development. Creating a custom toolbar for **ReportViewer** is very simple because each button in the default toolbar has a corresponding command in the control.

- **Silverlight Toolkit Themes**

  Add style to your UI with built-in support for the most popular Microsoft Silverlight Toolkit themes, including ExpressionDark, ExpressionLight, WhistlerBlue, RainerOrange, ShinyBlue, and BureauBlack.

ReportViewer for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne ReportViewer for Silverlight**. In this quick start you'll create a simple project using a C1ReportViewer control. You'll create a new Silverlight application, add the C1ReportViewer control to your application, add a PDF file that will be displayed in the C1ReportViewer control, and observe some of the run-time interaction possible with **ReportViewer for Silverlight**.

Step 1 of 3: Creating the C1ReportViewer Application

In this step you'll create a Silverlight application using **ReportViewer for Silverlight**. When you add a C1ReportViewer control to your application, you'll have a complete, functional document viewer interface that you can display PDF and HTML files in. To set up your project and add a C1ReportViewer control to your application, complete the following steps:

1. From the Visual Studio **File** menu select **New** and choose **Project**.
2. In the **New Project** dialog box choose a language and Silverlight **Application** in the left-side menu, choose **.NET Framework 4** in the **Framework** drop-down list, and enter a name for the project. You may also need to select Silverlight 4 as the Silverlight version number. In this example the application will be named "QuickStart". If you name the project something else, in later steps you may need to change references to "QuickStart" with the name of your project.
3. In the Solution Explorer, right-click the project name and choose **Add Reference**. In the **Add Reference** dialog box, locate and select the **C1.Silverlight** and **C1.Silverlight.ReportViewer** assemblies and click **OK** to add references to your project.
4. Open the XAML view of the MainPage.xaml file; in this quick start you'll add the C1ReportViewer control using XAML markup.
5. Add the XAML namespace to the UserControl tag with the following markup: `xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"`.

   The namespaces will now appear similar to the following:

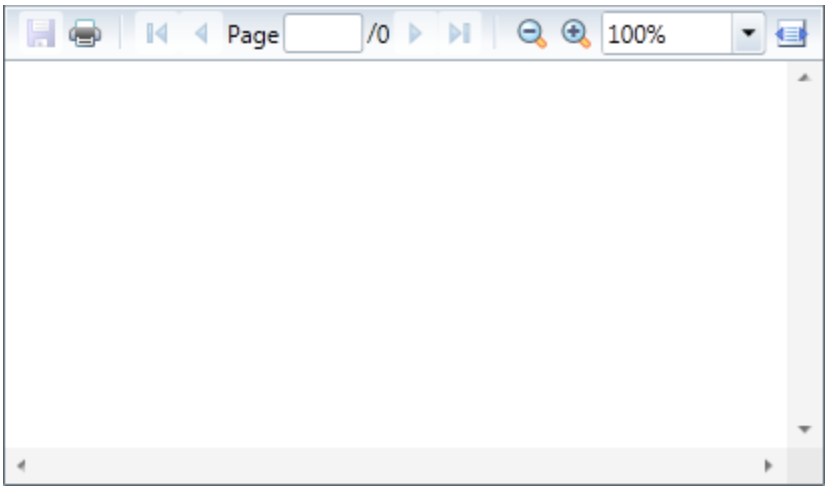
   ```
   <UserControl x:Class="QuickStart.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
   mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
   ```

   This is a unified namespace that will enable you to work with most ComponentOne WPF or Silverlight controls without adding multiple namespaces.
6. Add the `<c1:C1ReportViewer x:Name="C1ReportViewer1" />` tag within the Grid tags on the page to add the C1ReportViewer control to the application.

   The XAML will appear similar to the following:

   ```
   <Grid x:Name="LayoutRoot" Background="White">
       <c1:C1ReportViewer x:Name="C1ReportViewer1" />
   </Grid>
   ```

   This will add a C1ReportViewer control named "C1ReportViewer1" to the application. If you run the application now, it will appear similar to the following image:



You've successfully set up your application's user interface, but if you run your application now you'll see that the C1ReportViewer control currently contains no content. In the next steps you'll add content to the C1ReportViewer control, and then you'll observe some of the run-time interactions possible with the control.

Step 2 of 3: Adding Content to the C1ReportViewer Control

In the previous step you created a Silverlight application and added the C1ReportViewer control to your project. In this step you'll add PDF content to the C1ReportViewer control. Note that in this step you will add a PDF file that is included with the **ComponentOne Studio for Silverlight** samples, which are by default installed in the **Documents** or **MyDocuments** folder in the **ComponentOne Samples\Studio for Silverlight 4.0\C1.Silverlight.ReportViewer\PdfViewerSamples** directory. If you choose, you can instead use another PDF file and adapt the steps. To customize your project and add a PDF file to the C1ReportViewer control in your application, complete the following steps:

1. Navigate to the Solution Explorer, right-click the project name, and select **Add** | **Existing Item**.
2. In the **Add Existing Item** dialog box, locate the **C1XapOptimizer.pdf** file included in the **ControlExplorer** sample. In the file type drop-down box, you may need to choose **All Files** to view the PDF file. Note that if you choose, you can instead pick another PDF file to use.
3. In the Solution Explorer, click the PDF file you just added to the application. In the Properties window, set its **BuildAction** property to **Resource** and confirm that the **Copy to Output Directory** item is set to **Do not Copy**.
4. Switch to Code view by right-clicking the page and selecting **View Code**. In the next steps you'll add XAML markup to your application to add content to the drop-down box.
5. Add the following imports statement at the top of the page:
   - Visual Basic
     ```
     Imports C1.Silverlight.ReportViewer
     ```
   - C#
     ```
     using C1.Silverlight.ReportViewer;
     ```
6. Add the following code to the main class :
   - Visual Basic
     ```
     Public Sub New()
         InitializeComponent()
     ```

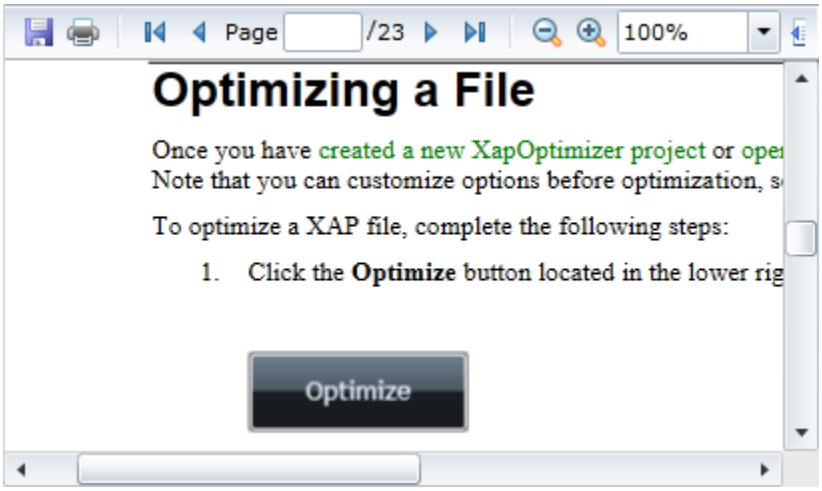
```
    Dim resource = Application.GetResourceStream(New
Uri("QuickStart;component/C1XapOptimizer.pdf", UriKind.Relative))
Me.C1ReportViewer1.LoadDocument(resource.Stream)
End Sub
```

- C#

```
public MainPage()
{
    InitializeComponent();
    var resource = Application.GetResourceStream(new
Uri("QuickStart;component/C1XapOptimizer.pdf", UriKind.Relative));
    this.C1ReportViewer1.LoadDocument(resource.Stream);
}
```

This code adds a stream and loads the stream into the C1ReportViewer control. Note that if you named the application differently, you will need to replace "QuickStart" with the name of your project. If you added a different PDF file, replace "C1XapOptimizer.pdf " with the name of your file.

If you run the application now, it will appear in the content window within the C1ReportViewer control:



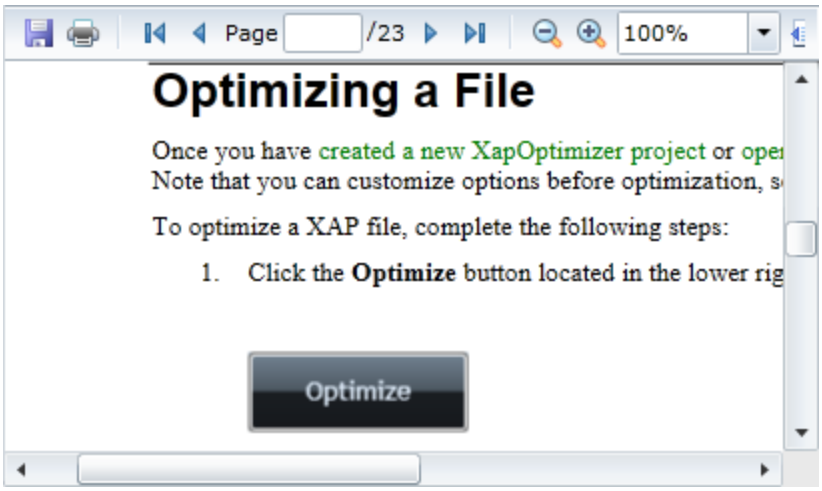
In this step you added content to the C1ReportViewer control. In the next step you'll view some of the run-time interactions possible in the control.

Step 3 of 3: Running the C1ReportViewer Application

Now that you've created a Silverlight application and added content to the C1ReportViewer control, the only thing left to do is run your application. To run your application and observe **ReportViewer for Silverlight**'s run-time behavior, complete the following steps:

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

   The application will appear similar to the following:



The C1ReportViewer control appears as a toolbar and content area. Notice that the PDF file you added appears in the content area of the control.

2. In the toolbar, click the **Next Page** arrow button to move to the next page of the PDF file. You can return to the previous page by clicking the **Previous Page** arrow button. You can also navigate to the first or last page of the document using the **First Page** and **Last Page** buttons.
3. Click the **Zoom Out** button to view more of the PDF in the window. Note that you can also choose a zoom level by clicking the **Zoom** drop-down box.
4. Click the **Fit Width** button to automatically fit the width of the PDF file to the size of the viewer's content window. Other options include **OnePage** to view the entire page in the available space and **TwoPage** to view two pages of the document in the available space.
5. Click in the **Search** text box and enter text to search for – for example "Sales". Notice that the document scrolls to the next instance of that word and that the word is highlighted in the PDF file. The toolbar also displays the number of instances of that word or phrase. You can click the **Find Previous** and **Find Next** buttons to navigate to the previous or next instance of the word.
6. Click the **Save** button. In the **Save** As dialog box enter a name for the file and click the **Save** button to save the file to a location of your choice. If you choose, you can click the **Print** button in the toolbar to print the file.

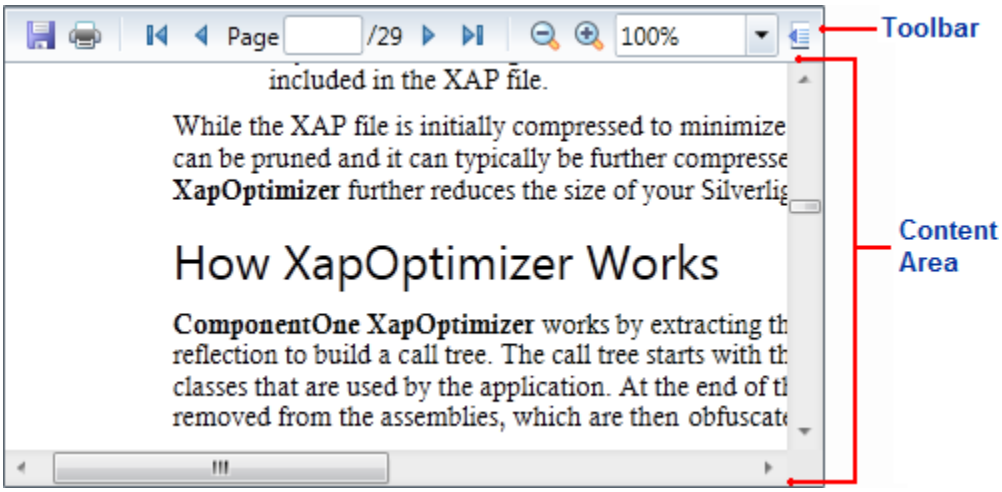
Congratulations! You've completed the **ReportViewer for Silverlight** quick start and created a simple Silverlight application, added and customized a **ReportViewer for Silverlight** control, and viewed some of the run-time capabilities of the control.

Working with ReportViewer for Silverlight

ComponentOne ReportViewer for Silverlight includes the C1ReportViewer control, a simple viewer that allows you to load and view HTML and PDF files. When you add the C1ReportViewer control to a XAML window it exists as a fully functional input control that can be customized and include loaded content.

ReportViewer Elements

The C1ReportViewer control consists of two parts: a toolbar and a content area. The image below identifies the toolbar and content area:



Any HTML content or PDF that you load into the C1ReportViewer control will be viewed in the content area. The toolbar allows users to manipulate the content at run time, for example to print or zoom in or out the content. For more information about the content area and toolbar, see the ReportViewer Content Area (page 12) and ReportViewer Toolbar (page 14) topics.

ComponentOne ReportViewer for Silverlight also includes the C1ReportViewerToolbar control which consists of just the toolbar element.

Basic Properties

ComponentOne ReportViewer for Silverlight includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see ReportViewer for Silverlight Appearance Properties (page 10) for more information about properties that control appearance.

The following properties let you customize the C1ReportViewer control:

Property	Description
FindText	Gets or sets the text that will be searched by FindNext and FindPrevious methods.
HorizontalScrollBarVisibility	Gets or sets a value that indicates whether a horizontal ScrollBar should be displayed.
PageCount	Gets the current number of display pages for the content hosted by the C1ReportViewer.
PageMargin	Gets or sets the margin of the page used for displaying and printing HTML documents.
PageNumber	Gets the page number for the currently displayed page.
PageSeparation	Gets or sets the separation between pages.

PageSize	Gets or sets the size of the page used for displaying and printing HTML documents.
PageTemplate	Gets or sets the DataTemplate used to display pages.
SelectionBackground	Gets or sets a Brush for this C1ReportViewer control's selection.
ToolbarStyle	Gets or set the style applied to this C1ReportViewer control's toolbar.
ToolbarVisibility	Gets or sets the Visibility of this C1ReportViewer toolbar.
VerticalScrollBarVisibility	Gets or sets a value that indicates whether a horizontal ScrollBar should be displayed.
ViewMode	Gets or sets the **ViewMode** for this C1ReportViewer.
ViewportHeight	Gets a value that contains the vertical size of the viewable content.
ViewportWidth	Gets a value that contains the horizontal size of the viewable content.
Zoom	Gets or sets the document zoom.

Basic Events

ComponentOne ReportViewer for Silverlight includes 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 C1ReportViewer control:

Event	Description
FindCountChanged	Event raised when the **FindCount** property has changed.
FindNumberChanged	Event raised when the **FindNumber** property has changed.
FindTextChanged	Event raised when the **FindText** property has changed.
IsFlowingChanged	Event raised when the **IsFlowing** property has changed.
PageCountChanged	Event raised when the **PageCount** property has changed.
PageNumberChanged	Event raised when the **PageNumber** property has changed.
RequestNavigate	Fired when a link from a document is clicked.
ViewModeChanged	Event raised when the **ViewMode** property has changed.
ZoomChanged	Event raised when the **Zoom** property has changed.

ReportViewer for Silverlight Layout and Appearance

The following topics detail how to customize the C1ReportViewer control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of Silverlight's XAML-based styling. You can also use templates to format and lay out the control and to customize the control's actions.

ReportViewer for Silverlight Appearance Properties

ComponentOne ReportViewer for Silverlight 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.
SelectionBackground	Gets or sets a Brush for this C1ReportViewer control's selection.
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.

ReportViewer Templates

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

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1ReportViewer control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.

Once you've created a new template, the template will appear in the **Objects and Timeline** window. Note that you can use the Template property to customize the 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.

Additional Templates

In addition to the default template, the C1ReportViewer control includes a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1ReportViewer control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty**.

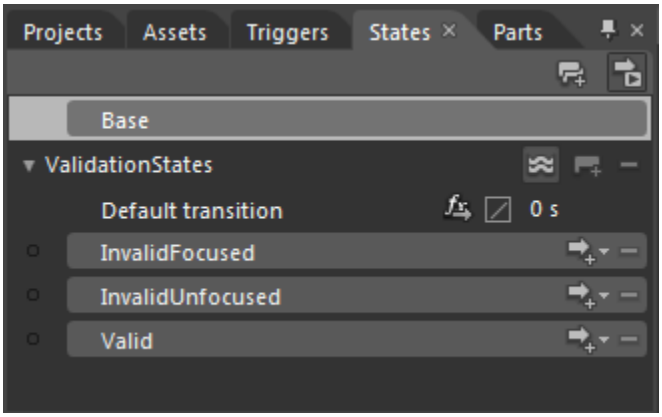
C1ReportViewer Styles and Templates

ComponentOne ReportViewer for Silverlight's C1ReportViewer 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
FocusVisualStyle	Gets or sets a property that enables customization of appearance, effects, or other style characteristics that will apply to this element when it captures keyboard focus. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
PageTemplate	Gets or sets the DataTemplate used to display pages.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.
ToolbarStyle	Gets or set the style applied to this C1ReportViewer control's toolbar.

C1ReportViewer 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. Once you've done so the available visual states for that part will be visible in the **States** window:



Run-Time Interaction

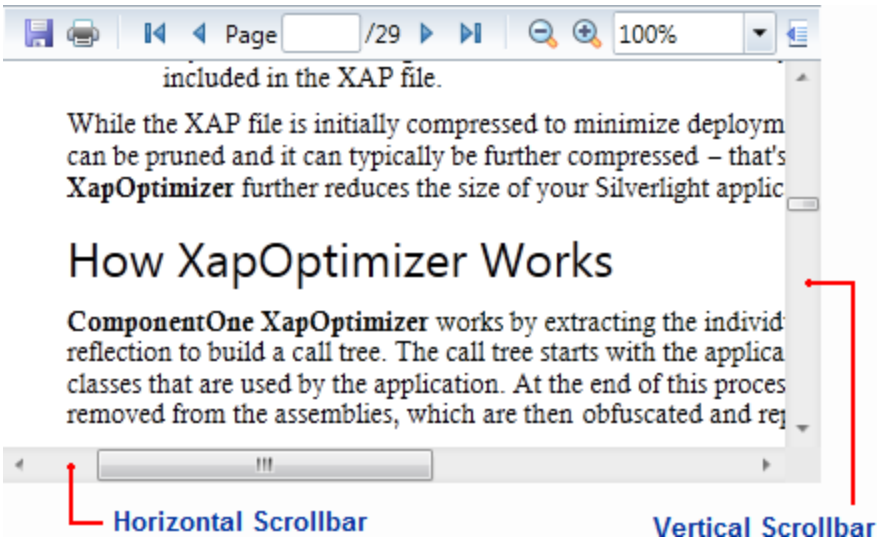
Users can interact with items in the toolbar and content area of the C1ReportViewer control at run time. Users can move and drag content in the content area or use the toolbar to manipulate the document displayed in the content area.

ReportViewer Content Area

At run time, users can manipulate content in the content area, scrolling, selecting, and copying content from the C1ReportViewer control.

Scrolling Content

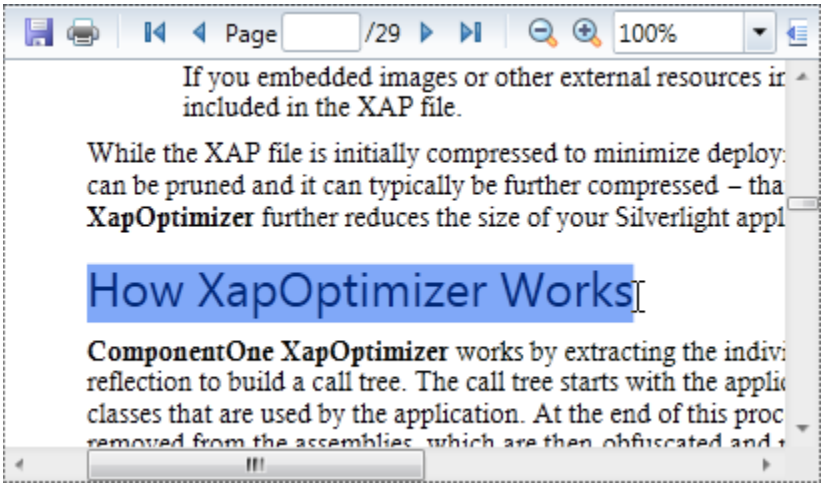
When the content of the control is taller and viewer than the viewing area of the control's content area, scrollbars appear to allow users to move to different areas of the document:



You can scroll through the content area using the arrow buttons, moving the scrollbar thumb buttons, with the keyboard arrow buttons, or with the mouse scroll wheel.

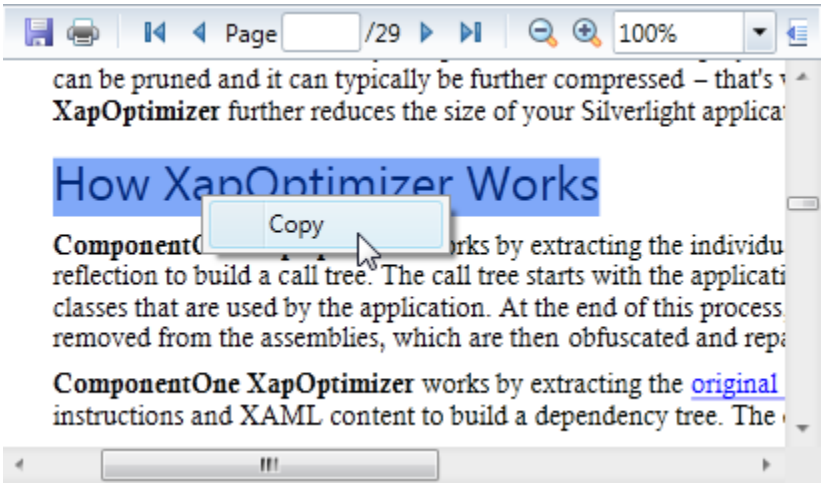
Selecting Content

You can select content using by clicking and dragging the mouse cursor over the content you want to select. When content is selected, it will appear highlight. For example, the words "How XapOptimizer Works" are selected in the image below:



Copying Content

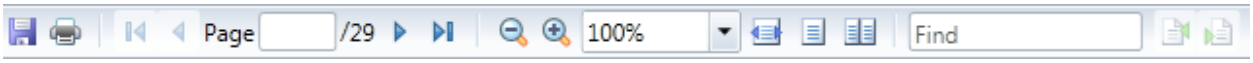
The C1ReportViewer control includes a context menu that allows you to copy content. First select the content that you want to copy and then right-click the document. A context menu will appear; by selecting **Copy** in the context menu you can copy the content:



You can also copy selected content using the keyboard by using the **CTRL + C** key combination.

ReportViewer Toolbar

At run time, users can use the toolbar to manipulate the document displayed in the content area. The toolbar appears similar to the following image by default:



Note that some items in the toolbar are not active or visible by default. For example, the **Previous Page** button is not active when on the first page of the document. The following options are included in the toolbar:

Image	Name	Description
	Save	Save the current document to the local file system.
	Print	Prints the current document.
	First Page	Navigates to the first page in the document.
	Previous Page	Navigates to the previous page in the document.
Page 3 /29	Page	Navigates to a specific page entered in the text box.
	Next Page	Navigates to the next page in the document.
	Last Page	Navigates to the last page in the document.
	Zoom Out	Zooms out of the document.
	Zoom In	Zooms into the document.
100%	Zoom	Zooms to the value selected.
	Fit Width	Fits the width of the document to the size of the viewport.
	One Page	Fits the size of the document to the size of the viewport so

		that an entire page is displayed.
	Two Pages	Displays two pages side-by-side.
	Find	Searches the document for text entered in the box as it is typed.
	Find Previous	Navigates to the previous instance of the searched text.
	Find Next	Navigates to the next instance of the searched text.

Each of the toolbar interactions can be also be performed programmatically so that you can easily replace the built-in toolbar with your own custom toolbar. If you choose to create a custom toolbar, you can hide the built-in toolbar using the ToolbarVisibility property.

ReportViewer for Silverlight 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 C1ReportViewer control in general. If you are unfamiliar with the **ComponentOne ReportViewer for Silverlight** product, please see the ReportViewer for Silverlight Quick Start (page 4) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne ReportViewer for Silverlight** product. Most task-based help topics also assume that you have created a new Silverlight project and added a C1ReportViewer control to the project – for information about creating the control, see Adding C1ReportViewer to the Application (page 15).

Adding C1ReportViewer to the Application

Complete the following steps to add a C1ReportViewer control to your application:

1. From the Visual Studio **File** menu select **New** and choose **Project**.
2. In the **New Project** dialog box choose a language in the left-side menu, choose **.NET Framework 4** in the **Framework** drop-down list, and enter a name for the project.
3. In the Solution Explorer, right-click the project name and choose **Add Reference**. In the **Add Reference** dialog box, locate and select the following assemblies and click **OK** to add references to your project:
   - C1.Silverlight
   - C1.Silverlight.ReportViewer
   - C1.Silverlight.RichTextBox
   - C1.Silverlight.Zip
4. Open the XAML view of the MainPage.xaml file and add the XAML namespace to the UserControl tag with the following markup:
   `xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"`.

   The namespaces will now appear similar to the following:

```
<UserControl x:Class="QuickStart.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
```

```
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="400">
```

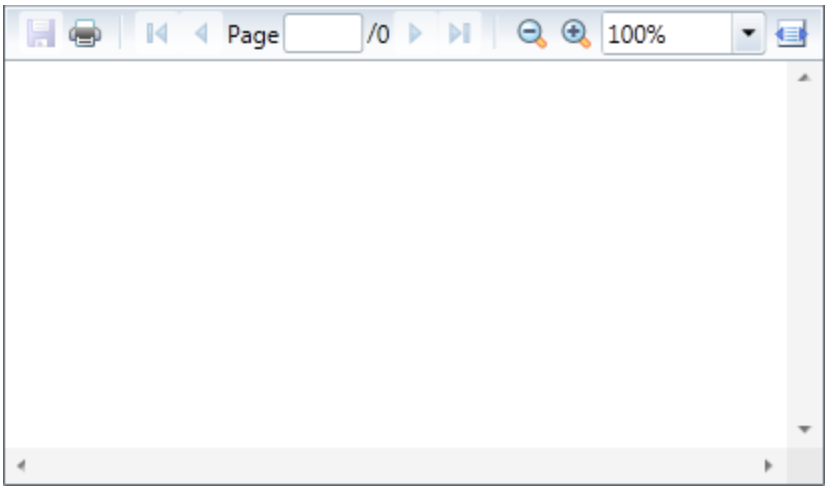
This is a unified namespace that will enable you to work with most ComponentOne WPF or Silverlight controls without adding multiple namespaces.

5. Add the `<c1:C1ReportViewer x:Name="C1ReportViewer1" />` tag within the Grid tags on the page to add the C1ReportViewer control to the application.

   The XAML will appear similar to the following:

```
<Grid x:Name="LayoutRoot" Background="White">
    <c1:C1ReportViewer x:Name="C1ReportViewer1" />
</Grid>
```

   This will add a C1ReportViewer control named "C1ReportViewer1" to the application. If you run the application now, it will appear similar to the following image:



You've successfully set up your application's user interface, but if you run your application now you'll see that the C1ReportViewer control currently contains no content. See the Loading Documents into C1ReportViewer (page 16) topic for options for loading content.

Note: If the **C1ReportViewer** control was installed to the Visual Studio Toolbox, simply dragging the control onto a page will automatically perform all the steps above.

Loading Documents into C1ReportViewer

If you run the application after adding a C1ReportViewer control to your application, you'll see an empty C1ReportViewer on the page. The next step is to invoke the LoadDocument method to add some content to the control. The LoadDocument method allows you to load content from **Stream** objects (which may contain PDF, HTML, or MHTML documents), or from **strings** (which may contain HTML or MHTML documents).

Loading Documents from Application Resources

You can easily load a document from an application resource. For example, complete the following steps:

1. Navigate to the Solution Explorer, right-click the project name, and select **Add** | **Existing Item**.

2. In the **Add Existing Item** dialog box, locate a PDF file. In the file type drop-down box, you may need to choose **All Files** to view the PDF file. Note that if you choose, you can instead pick another PDF file to use.
3. In the Solution Explorer, click the PDF file you just added to the application (in this example, we'll assume the file is named **resource.pdf**). In the Properties window, set its **BuildAction** property to **Resource** and confirm that the **Copy to Output Directory** item is set to **Do not Copy**.
4. Switch to Code view by right-clicking the page and selecting **View Code**. In the next steps you'll add XAML markup to your application to add content to the drop-down box.
5. Add the following imports statement at the top of the page:
   - Visual Basic
     ```
     Imports C1.Silverlight.ReportViewer
     ```
   - C#
     ```
     using C1.Silverlight.ReportViewer;
     ```
6. Add the following code to the main class:
   - Visual Basic
     ```
     Public Sub New()
         InitializeComponent()
         Dim resource = Application.GetResourceStream(New
     Uri("AppName;component/resource.pdf", UriKind.Relative))
     Me.C1ReportViewer1.LoadDocument(resource.Stream)
     End Sub
     ```
   - C#
     ```
     public MainPage()
     {
         InitializeComponent();
         var uri = new Uri("/AppName;component/resource.pdf",
     UriKind.Relative);
         var resource = Application.GetResourceStream(uri);
         this.C1ReportViewer1.LoadDocument(resource.Stream);
     }
     ```

This code adds a stream and loads the stream into the C1ReportViewer control. Note that if you named the application differently, you will need to replace "AppName" with the name of your project. If you added a different PDF file, replace "resource.pdf" with the name of your file.

Loading Documents from Files on the Client Machine

In this example, you'll set up the application so that users can load a file from the local file system. You will add a button to the application and then add code to choose and open a file at run time. Because this example uses the **OpenFileDialog** control this code must be executed in a **Button Click** event. Note that this topic assumes you have added a C1ReportViewer control named "C1ReportViewer1" to your application.

Complete the following steps.

1. Open the MainPage.xaml file in your application, and open XAML view.
2. Add the following markup to add a button control to the application.
   ```
   <Button Content="Load File" Height="23" Name="Button1" Width="70"
   Click="Button1_Click" />
   ```
3. Right-click the page and select **View Code**. In Code View you'll add code to initialize the button you added in the previous step.
4. Add the following imports statement at the top of the page:

- Visual Basic
```
Imports C1.Silverlight.ReportViewer
```
- C#
```
using C1.Silverlight.ReportViewer;
```

5. Add the following **Button_Click** event handler code:

- Visual Basic
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles Button1.Click
    Dim dialog = New OpenFileDialog()
    dialog.Filter = "PDF files|*.pdf|HTML files|*.html;*.mhtml"
    If dialog.ShowDialog() = True Then
        Using fileStream = dialog.File.OpenRead()
            Try
                C1ReportViewer1.LoadDocument(fileStream)
            Catch ex As Exception
                MessageBox.Show("Failed to load document.")
            End Try
        End Using
    End If
End Sub
```
- C#
```
private void Button1_Click(System.Object sender,
System.Windows.RoutedEventArgs e)
{
   dynamic dialog = new OpenFileDialog();
   dialog.Filter = "PDF files|*.pdf|HTML files|*.html;*.mhtml";
   if (dialog.ShowDialog() == true) {
         using (fileStream == dialog.File.OpenRead()) {
               try {
                     C1ReportViewer1.LoadDocument(fileStream);
               } catch (Exception ex) {
                     MessageBox.Show("Failed to load document.");
               }
         }
   }
}
```

This code initializes a dialog box to be opened when the button is clicked. In the dialog box users can select a file to open in the C1ReportViewer control. Notice how, in the code above, the **LoadDocument** method allows you to load PDF and HTML content.

6. Run the application.
7. In the running application, click the **Load File** button. Notice that a dialog box appears, allowing you to choose a PDF or HTML file of your choice.
8. Locate and select a PDF file on your local machine to open and then click the **Open** button. The dialog box will close and the file you selected will be loaded into the C1ReportViewer control.

Loading Documents from Files on the Server

A common usage scenario for the **C1ReportViewer** control is to have a report server (such as **C1Report** or Microsoft SQL Server Reporting Services) generate reports on a schedule, and deploy them to the file system on the server. Your Silverlight or WPF applications can then get these files from the server and display them to the user with very little overhead.

This scenario is illustrated in the **C1ReportViewerQuickstart** sample.

After adding the **C1ReportViewer** to your application, you should add a Silverlight-enabled WCF service to the server project. This service will provide the Silverlight client with the list of reports available and with the actual document streams for each report.

For example, the following is a typical implementation of a report provider Web service:

- Visual Basic

```
<ServiceContract([Namespace] := "")> _
<AspNetCompatibilityRequirements(RequirementsMode :=
AspNetCompatibilityRequirementsMode.Allowed)> _
Public Class ReportingService
      <OperationContract> _
      Public Function GetReportList() As String()
      Dim path__1 =
Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory, "Resources")
      Return Directory.GetFiles(path__1, "*.pdf")
      End Function
      <OperationContract>
Public Function GetReportStream(reportName As String) As Byte()
      ' get file name
      Dim path__1 =
Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory, "Resources")
      reportName = Path.Combine(path__1, reportName)

      ' load file into stream
      Dim ms = New MemoryStream()
      Dim buff = New Byte(63999) {}
      Using sr = New FileStream(reportName, FileMode.Open)
            While True
                  Dim read As Integer = sr.Read(buff, 0, buff.Length)
                  ms.Write(buff, 0, read)
                  If read = 0 Then
                        Exit While
                  End If
            End While
      End Using
End Function
' return byte stream
Return ms.ToArray()
End Class
```

- C#

```
[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements(RequirementsMode =
         AspNetCompatibilityRequirementsMode.Allowed)]
public class ReportingService
{
  [OperationContract]
  public string[] GetReportList()
  {
    var path = Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory,
"Resources");
    return Directory.GetFiles(path, "*.pdf");
  }

  [OperationContract]
  public byte[] GetReportStream(string reportName)
  {
```

```
    // get file name
      var path =
Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory, "Resources");
      reportName = Path.Combine(path, reportName);

      // load file into stream
      var ms = new MemoryStream();
      var buff =new byte[64000];
      using (var sr = new FileStream(reportName, FileMode.Open))
      {
        for (; ; )
        {
          int read = sr.Read(buff, 0, buff.Length);
          ms.Write(buff, 0, read);
          if (read == 0) break;
        }
      }

    // return byte stream
    return ms.ToArray();
  }
}
```

As you can see, the code is very standard. The first method lists the reports available on the server so the Silverlight application can show a list of reports to the user, and the second method returns the byte stream that represents the selected report.

The client part of the application uses the service as follows:

- Visual Basic

```
Public Sub New()
      InitializeComponent()

      ' go get the list of reports available
      Dim svc = New ReportingServiceReference.ReportingServiceClient()
      AddHandler svc.GetReportListCompleted, AddressOf
svc_GetReportListCompleted
      svc.GetReportListAsync()
End Sub

' populate ComboBox with list of reports available on the server
Private Sub svc_GetReportListCompleted(sender As Object, e As
ReportingServiceReference.GetReportListCompletedEventArgs)
      _cmbReport.Items.Clear()
      For Each file As String In e.Result
            _cmbReport.Items.Add(Path.GetFileNameWithoutExtension(file))
      Next
      _cmbReport.IsEnabled = True
      _cmbReport.SelectedIndex = 0
End Sub

' show the report that was selected
Private Sub ReportType_Click(sender As Object, e As EventArgs)
      ' build report name
      Dim reportName As String = DirectCast(_cmbReport.SelectedItem,
String)
      reportName += If(_btnPDF.IsChecked.Value, ".pdf", ".mhtml")
```

```
      ' go get the stream
      Dim svc = New ReportingServiceReference.ReportingServiceClient()
      AddHandler svc.GetReportStreamCompleted, AddressOf
svc_GetReportStreamCompleted
      svc.GetReportStreamAsync(reportName)
End Sub

' display the report
Private Sub svc_GetReportStreamCompleted(sender As Object, e As
ReportingServiceReference.GetReportStreamCompletedEventArgs)
      Dim ms = New MemoryStream(e.Result)
      _reportViewer.LoadDocument(ms)
End Sub
```

- C#

```
public MainPage()
{
  InitializeComponent();

  // go get the list of reports available
  var svc = new ReportingServiceReference.ReportingServiceClient();
  svc.GetReportListCompleted += svc_GetReportListCompleted;
  svc.GetReportListAsync();
}

// populate ComboBox with list of reports available on the server
void svc_GetReportListCompleted(object sender,
  ReportingServiceReference.GetReportListCompletedEventArgs e)
{
  _cmbReport.Items.Clear();
  foreach (string file in e.Result)
  {
    _cmbReport.Items.Add(Path.GetFileNameWithoutExtension(file));
  }
  _cmbReport.IsEnabled = true;
  _cmbReport.SelectedIndex = 0;
}

// show the report that was selected
void ReportType_Click(object sender, EventArgs e)
{
  // build report name
  string reportName = (string)_cmbReport.SelectedItem;
    reportName += _btnPDF.IsChecked.Value ? ".pdf" : ".mhtml";

    // go get the stream
    var svc = new ReportingServiceReference.ReportingServiceClient();
    svc.GetReportStreamCompleted += svc_GetReportStreamCompleted;
    svc.GetReportStreamAsync(reportName);
  }

  // display the report
  void svc_GetReportStreamCompleted(object sender,
    ReportingServiceReference.GetReportStreamCompletedEventArgs e)
  {
    var ms = new MemoryStream(e.Result);
```

```
    _reportViewer.LoadDocument(ms);
  }
```

Creating and Loading Reports Dynamically

Another fairly common scenario is dynamic report creation. In this case, you would configure the report server to allow access from your application server, and would use a Web service similar to the one described in the Loading Documents from Files on the Server (page 18) topic to obtain the report stream directly from the report server (instead of loading it from a file).

The specific steps involved depend on the specific report server you are using. To dynamically obtain a PDF report from a Microsoft SQL Server Reporting Services server, for example, you would modify the Web service listed in the Loading Documents from Files on the Server (page 18) topic as follows:

- Visual Basic

```
<ServiceContract([Namespace] := "")> _
<AspNetCompatibilityRequirements(RequirementsMode :=
AspNetCompatibilityRequirementsMode.Allowed)> _
Public Class ReportsService
      <OperationContract> _
      Public Function GetReportStream(reportName As String) As Byte()
            Dim reportServer As String = "YOUR REPORT SERVER NAME"

            Dim url =
String.Format("http://{0}/ReportServer?/{1}&rs:Format=PDF", reportServer,
reportName.Replace(" "C, "+"C))

            Dim wc = New System.Net.WebClient()
            wc.UseDefaultCredentials = True

            Dim stream = wc.OpenRead(url)
            Dim ms = New MemoryStream()

            Dim buf = New Byte(64 * 1024 - 1) {}
            While True
                  Dim read As Integer = stream.Read(buf, 0, buf.Length)
                  If read = 0 Then
                        Exit While
                  End If
                  ms.Write(buf, 0, read)
            End While
            ms.Flush()

            Return ms.ToArray()
      End Function
End Class
```

- C#

```
[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements(RequirementsMode =
     AspNetCompatibilityRequirementsMode.Allowed)]
public class ReportsService
{
  [OperationContract]
  public byte[] GetReportStream(string reportName)
  {
    string reportServer = "YOUR REPORT SERVER NAME";
```

```
    var url = string.Format("http://{0}/ReportServer?/{1}&rs:Format=PDF",
      reportServer,
      reportName.Replace(' ', '+'));

    var wc = new System.Net.WebClient();
    wc.UseDefaultCredentials = true;

    var stream = wc.OpenRead(url);
    var ms = new MemoryStream();

    var buf = new byte[64 * 1024];
    for (; ; )
    {
      int read = stream.Read(buf, 0, buf.Length);
      if (read == 0) break;
      ms.Write(buf, 0, read);
    }
    ms.Flush();

    return ms.ToArray();
  }
}
```

As you can see, the difference is minimal. This approach still allows you to specify the report caching policy on the report server, so there is no loss of performance or scalability.

Hiding the Toolbar

If you choose to create a customized toolbar (for an example, see the Customizing the Toolbar (page 23) topic), you may need to hide the default toolbar available in the C1ReportViewer control. You can hide the built-in toolbar using the ToolbarVisibility property. For example:

- XAML
```
<c1:C1ReportViewer x:Name="C1ReportViewer1"
ToolbarVisibility="Collapsed"/>
```
- Visual Basic
```
Me.C1ReportViewer.ToolbarVisibility = Visibility.Collapsed
```
- C#
```
this.C1ReportViewer.ToolbarVisibility = Visibility.Collapsed;
```

Customizing the Toolbar

Creating a custom toolbar for **C1ReportViewer** is very simple. All buttons in the default toolbar have a corresponding Command in the control, so you can create a custom toolbar using only XAML. Here is some sample code using **C1Toolbar** to create a **C1ReportViewer** toolbar:

```
<Grid x:Name="LayoutRoot">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition />
    </Grid.RowDefinitions>
    <c1:C1ToolbarStrip>
        <c1:C1ToolbarButton
            Content="First"
            Command="{Binding FirstPageCommand,ElementName=reportViewer}" />
        <c1:C1ToolbarButton
```

```
            Content="Previous"
            Command="{Binding PreviousPageCommand,ElementName=reportViewer}"
/>
        <ContentPresenter
            Content="{Binding PageNumber,ElementName=reportViewer}" />
        <TextBlock Text="/"/>
        <ContentPresenter
            Content="{Binding PageCount,ElementName=reportViewer}" />
        <c1:C1ToolbarButton
            Content="Next"
            Command="{Binding NextPageCommand,ElementName=reportViewer}" />
        <c1:C1ToolbarButton
            Content="Last"
            Command="{Binding LastPageCommand,ElementName=reportViewer}" />
        <ComboBox
            SelectedItem="{Binding
Zoom,ElementName=reportViewer,Mode=TwoWay}">
            <sys:Double>0.5</sys:Double>
            <sys:Double>1</sys:Double>
            <sys:Double>1.5</sys:Double>
        </ComboBox>
    </c1:C1ToolbarStrip>
    <c1:C1ReportViewer
        x:Name="reportViewer"
        Grid.Row="1"
        ToolbarVisibility="Collapsed"/>
</Grid>
```

Note how all buttons bind the **Command** property to a command in **C1ReportViewer**. Also, you can easily bind to the **PageNumber** and **PageCount** properties to display the current page and total number of pages. Finally, a **ComboBox** is bound to the **Zoom** property allowing the user to control the zoom factor.

Several additional buttons can be customized using various commands. This is the list of commands:

Command	Description
SaveCommand	Saves the document.
PrintCommand	Prints the document.
FirstPageCommand	Navigates to the first page in the document.
PreviousPageCommand	Navigates to the previous page in the document.
NextPageCommand	Navigates to the next page in the document.
LastPageCommand	Navigates to the last page in the document.
DecreaseZoomCommand	Zooms out of the document.
IncreaseZoomCommand	Zooms into the document.
FindPreviousCommand	Finds the previous instance of the searched text.
FindNextCommand	Finds the next instance of the searched text.

C1PdfViewerToolbar's template also expects **ToggleButtons** with the following names:

Option	Description
FitWidth	Fits the width of the document to the size of the control.

OnePage	Displays one page.
TwoPages	Displays two pages side-by-side.

To use one of these ToggleButtons, for example TwoPages, inside a custom toolbar scenario, you would need to put the TwoPages ToggleButton inside the C1ReportViewer template. If you are making your own toolbar outside of the control, add a Button and set C1ReportViewer.ViewMode = ViewMode.TwoPages in the click handler.