PDFWebViewer.NET 1.0

Developer Guide

Version: 1.4 (updated for product version 1.0.6.0)
Date: March 8, 2011

Legal Note

Contents

List of Code Samples

List of Tables

List of Figures

1. Introduction

This developer guide will help you understand the PDFWebViewer.NET 1.0 ASP.NET control. It walks through the main concepts and illustrates them with code samples. This guide is not intended as a type reference. A full type reference is included in the download and can be viewed online at http://www.tallcomponents.com/pdfwebviewer1/help/.

1.1. Features

PDFWebViewer.NET is a 100% .NET (verifiable) solution for viewing PDF documents in a webpage. The product includes a complete set of ASP.NET controls that show PDF content and help users navigate and inspect PDF documents. No PDF content is downloaded to the client. This gives you complete access control and eliminates the need for browser plug-ins like Adobe Acrobat.

These are PDFWebViewer.NET's primary features:

- View PDF in a web browser without downloading the document
- AJAX enabled controls provide a smooth user experience
- Smooth zoom, pan and scrolling
- Full text searching
- Control access to PDF documents at page level
- Compatible with all major browsers including IE7, IE8, FireFox, Opera and Chrome.
- No browser plug-ins required
- 100% .NET component, suitable for shared hosting deployment
- Single assembly for easy deployment
- Highly customizable
- Efficient caching to minimize server load and optimize response times

1.2. Basic concepts

PDFWebViewer.NET provides a solution for viewing PDF documents in web pages. There are three major components in this solution.

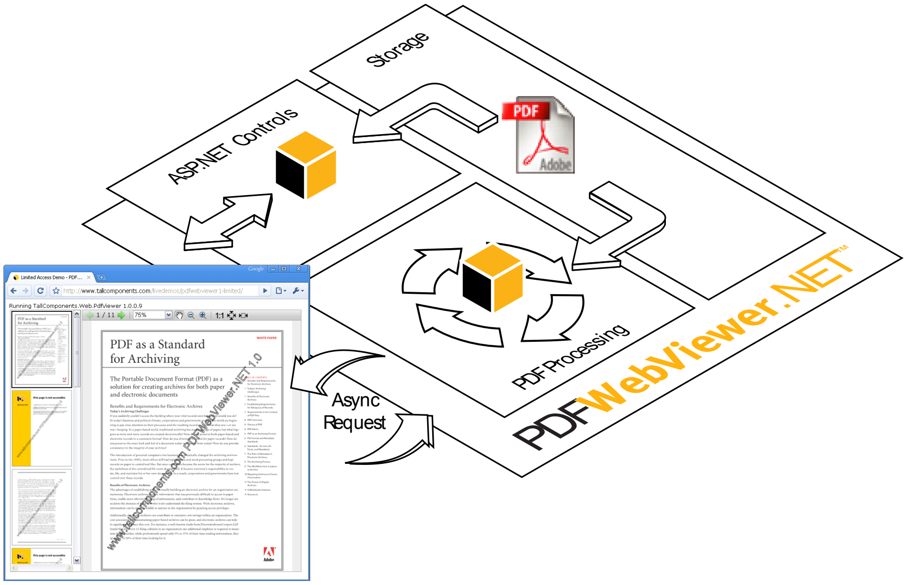


Figure 1-1 PdfWebViewer components

Storage is responsible for storing and retrieving PDF documents. Storage is discussed in detail in chapter 6.

PDF Processing renders the PDF content into images and provides information about documents. This is discussed in chapter 0.

The *ASP.NET Controls* display the rendered PDF content in an ASP.NET web page. The controls are discussed individually in chapter 3, 4 and 5.

To get started quickly, please see chapter 2 for a getting started guide and a walkthrough.

1.3. Conventions used in this document

Throughout this document, formatting is used to discern between different types of information.

A code sample is contained in between dashed yellow lines and uses a mono-space font.

```
// Code sample
```

Whenever a line in a code sample is too long to fit on the page, the character ↵ is inserted. When copy/pasting code from this document, make sure you remove that character and any whitespace following it.

Important side notes are formatted using a light-yellow background.

Note These blocks contain tips and warnings that may save you time and trouble.

1.4. Please send us feedback!

Please help us improve this document. If you have any questions, suggestions or find anything in this document is incorrect or missing then let us know at support@tallcomponents.com or send us a support request through our website at http://www.tallcomponents.com/question.aspx.

1.5. Online Resources

- Our web site:
  http://www.tallcomponents.com.
- PDFWebViewer.NET type reference online :
  http://www.tallcomponents.com/pdfwebviewer1/help/
- Microsoft Client Script Reference:
  http://msdn.microsoft.com/en-us/library/bb397536.aspx
- Microsoft ASP.NET Ajax for .NET 2.0 installer:
  http://www.microsoft.com/downloads/details.aspx?FamilyID=ca9d90fa-e8c9-42e3-aa19-08e2c027f5d6

1.6. Naming Convention

We try to adhere as much as possible to Microsoft's "Design Guidelines for Developing Class Libraries". The guideline can be found here:

http://msdn2.microsoft.com/en-us/library/ms229042.aspx

2. Getting started

2.1. Development environment

PDFWebViewer.NET has been compiled and tested with Microsoft Visual Studio.NET 2008 for Microsoft .NET Framework 2.0 and 4.0. PDFWebViewer.NET is a 100% managed .NET component and consists of just a single assembly: TallComponents.PDF.WebViewer.dll. PDFWebViewer.NET can be used with any ASP.NET application.

2.2. Requirements

PDFWebViewer.NET is compiled for version 2.0 and 4.0 of the Microsoft .NET framework. Both builds are included in the distribution.

2.2.1. Development environment

To develop applications using PDFWebViewer.NET any development environment for Microsoft .NET 2.0 or higher will do.

2.2.2. Microsoft .NET ASP.NET Ajax Extensions

To provide a good user experience, all web controls that are part of this product use Microsoft ASP.NET Ajax.

If your application targets the Microsoft .NET 3.5 platform, these extensions are included in the framework and no additional action is required.

If your application targets the Microsoft .NET 2.0 or 3.0 platform, you will need to install Microsoft ASP.NET Ajax 1.0. This extension is available for download free of charge from Microsoft at:

http://www.microsoft.com/downloads/details.aspx?FamilyID=ca9d90fa-e8c9-42e3-aa19-08e2c027f5d6

The dependency on Microsoft ASP.NET Ajax is limited to the `System.Web.Extensions` assembly. When deploying the application to a server, including that assembly in the /bin folder will also do. There is no need to install the Ajax extensions on the server or in the GAC (Global Assembly Cache), though you are free to do so if you wish.

2.3. Deployment

PDFWebViewer.NET is fully compatible with the XCopy deployment principle of .NET. This means that deploying PDFWebViewer.NET is as simple as copying the single assembly to the /bin folder of your web application. For .NET 2.0 and 3.0 sites, please also make sure the Microsoft Ajax Extensions are available (see section 2.2.1 for more information).

Although it is possible, there is no need to install PDFWebViewer.NET in the GAC (Global Assembly Cache).

2.3.1. Classic ASP

PDFWebViewer.NET does not support classic ASP.

2.4. Working with Visual Studio 2008

Throughout this document and all samples included with PDFWebViewer.NET we use Visual Studio 2008. The component will work with any development environment that can target the .NET 2.0 framework or newer.

2.4.1. IntelliSense

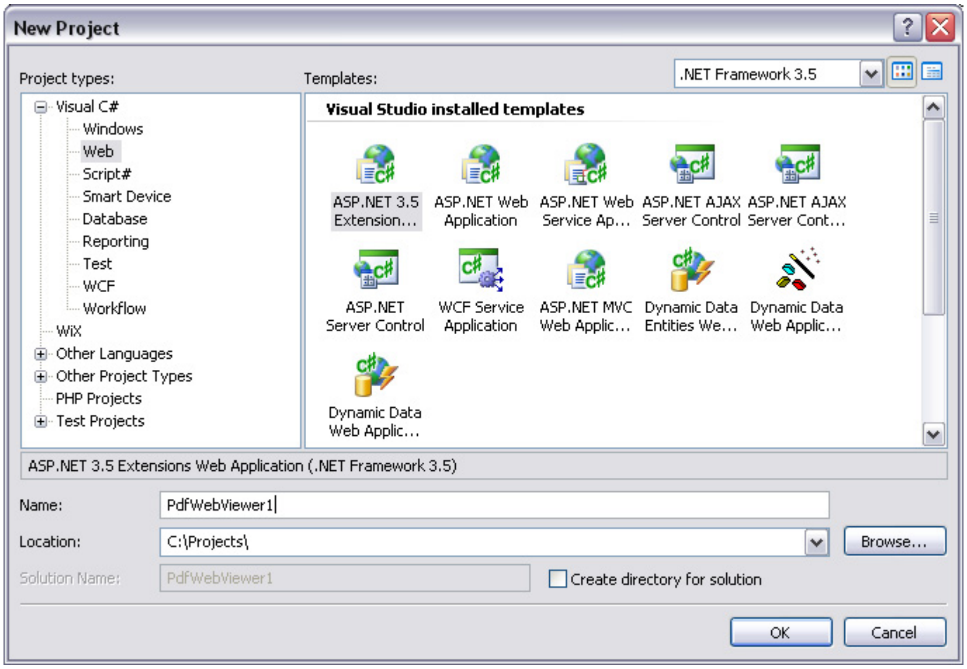
The PDFWebViewer.NET installation includes the XML documentation file TallComponents.PDF.WebViewer.xml, which contains all the documentation, found in the type reference. If this XML file is in the same folder as the assembly, Visual Studio provides IntelliSense documentation as you type in your code.

2.5. Walkthrough: A basic PDF viewer (C#)

This sample demonstrates how to setup a basic PDF viewer application with PDFWebViewer.NET. We will use Visual Studio 2008 to create a basic website project and setup a PdfViewer in the default page.

2.5.1. Start a new website

Start Visual Studio and select File › New › WebSite from the menu bar.



ASP.NET 3.5 Extensions Web Application (.NET Framework 3.5)

Name: PdfWebViewer1

Location: C:\Projects\

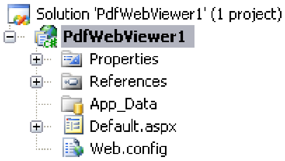
Solution Name: PdfWebViewer1

☐ Create directory for solution

Figure 2-1 Create a new ASP.NET 3.5 Project in Visual Studio 2008

Make sure the drop down menu at the top right reads “.NET Framework 3.5”. Then select “ASP.NET 3.5 Extensions Web Application” to create a new ASP.NET 3.5 website.

Enter the project name. In this sample we’ll assume the project is named “PdfWebViewer1”, but you’re free to choose any project name. You may also want to set the location for the project.



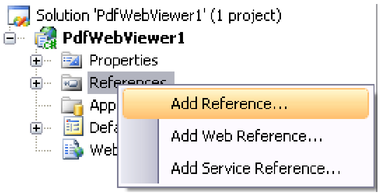
Solution 'PdfWebViewer1' (1 project)
- PdfWebViewer1
  - Properties
  - References
  - App_Data
  - Default.aspx
  - Web.config

Figure 2-2 The newly created project

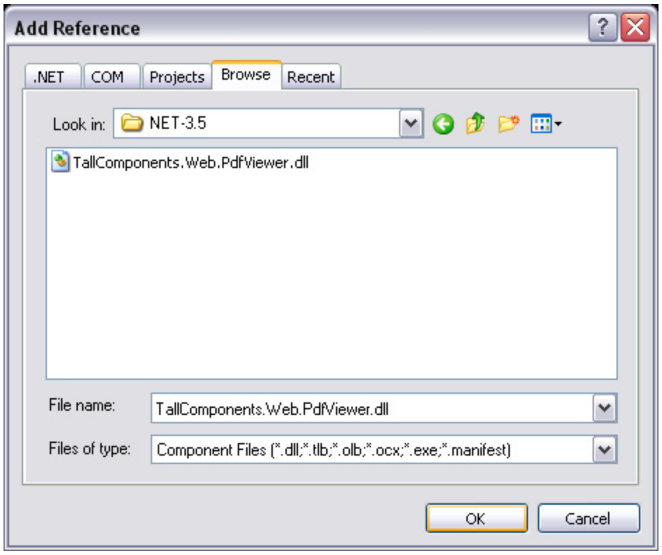
After clicking OK, Visual Studio will create an empty project that looks like figure 2-2.

2.5.2. Add reference to PDFWebViewer.NET

In order to use the controls and handlers in PDFWebViewer.NET we will have to reference the assembly TallComponents.Pdf.WebViewer.dll in the project. There are a couple of ways to do this. As demonstrated below you can right-click on the *References* folder in the project explorer and select *Add Reference...*



This will popup a dialog named *Add Reference*. Select the browse tab and browse to the location where you unpacked the downloaded component. Since we're building an ASP.NET 3.5 website, select the TallComponents.Pdf.WebViewer.dll file from the .NET-3.5 folder and click OK.



Look in: NET-3.5

TallComponents.Web.PdfViewer.dll

File name: TallComponents.Web.PdfViewer.dll

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

2.5.3. Add controls to the page

By default, the document default.aspx should now be open. If not, double click the file in the project explorer. Switch to source view and add the highlighted lines in the code sample below:

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="PdfWebViewer1._Default" %>
<%@ Register Assembly="TallComponents.Web.PdfViewer" TagPrefix="tc"
    Namespace="TallComponents.Web.Pdf" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>PDFWebViewer.NET Sample</title>
</head>
<body>
    <form id="form1" runat="server">
    <asp:ScriptManager runat="server" ID="ScriptManager1" />
    <div>
      <tc:PdfViewerToolbar runat="server" PdfViewerControlId="PdfViewer1"
          Width="800px" />
      <tc:PdfViewer runat="server" id="PdfViewer1" Width="800px"
          ServiceUrl="PdfToImage.ashx" />
    </div>
    </form>
</body>
</html>
```

Code sample 2-1 The code view of the page (default.aspx)

At the top of the page, the tag prefix *tc* is registered for use. Right after the <form> tag, an Ajax ScriptManager is added. This control is part of the ASP.NET Ajax extensions and is required when one or more Ajax controls are on the page.

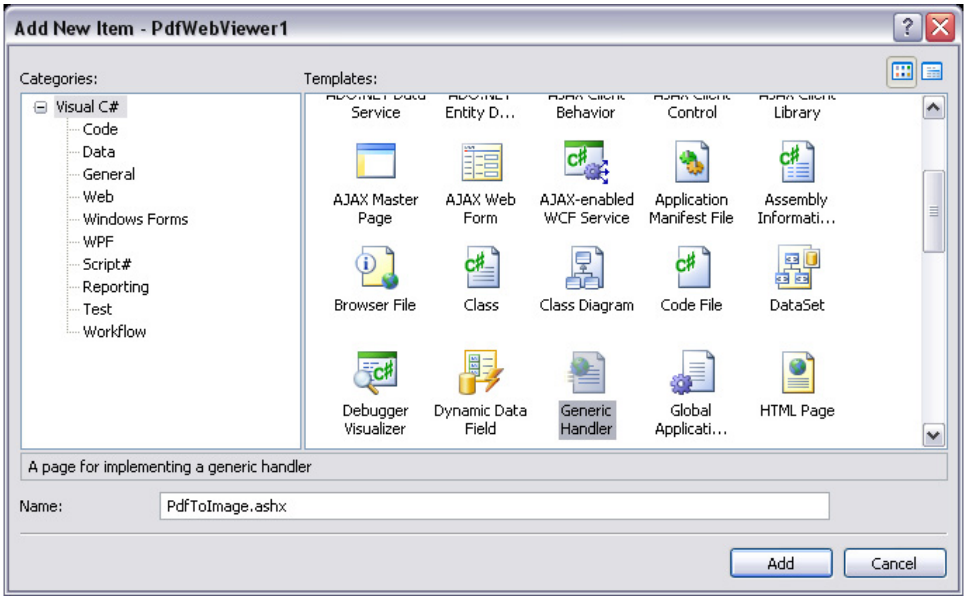
Next, the PdfViewerToolbar and PdfViewerControls are added. Note that the PdfViewerControlId refers to the ID of the viewer control. This required attribute enables the toolbar to hook up to the viewer.

The ServiceUrl attribute on the viewer is also required. This specified the URL at which the service used to render PDF (PdfToImageHandler) is located. We will configure that next.

2.5.4. Setup PDF To Image handler

To display PDF content in a web page, the PDF is converted to images. Everything needed to do this is included in the PdfToImageHandler class. The PdfViewer control will request images as needed. To do this the handler needs to be accessible by the control. There are a couple of ways to accomplish this, in this sample we'll add a generic handler (.ashx) file.

In Visual Studio, add a new item to the root folder of the project. From the dialog choose ‘Generic handler’ and enter the name for the handler as shown below:



A page for implementing a generic handler

Name: PdfToImage.ashx

This will generate a new handler and a code behind file. We don’t need the code behind since all the logic is implemented in PdfToImageHandler. Remove *PdfToImage.cs.ashx* from the project and replace the contents of *PdfToImage.ashx* with the following code:

```
<%@ WebHandler Class="TallComponents.Web.Pdf.PdfToImageHandler" %>
```

Code sample 2-2 Custom handler declaration in pageasimage.ashx

This line is all that is needed to declare the handler. It tells ASP.NET to use the PdfToImageHandler class tho handle requests for PdfToImage.ashx. Note that the name of the generic handler matches the URL set on the controls in the previous step.

2.5.5. Configure storage

The next step is to configure how the PDF documents are stored in this application. PDFWebViewer.NET comes with a provider that loads PDF documents from disk. This provider is called FileSystemStorageProvider.

The provider is configured in the web.config file. The next code sample demonstrates this.

```
<configuration>
   <configSections>
      <!-- other sections and groups for ASP.NET not shown here -->
      <section name="pdfStorage"
               type="TallComponents.Web.Storage.PdfStorageProviderSection,↵
                     TallComponents.Web.PdfViewer" />
   </configSections>
   <pdfStorage defaultProvider="filesystem">
      <providers>
         <clear />
         <add name="filesystem"
              type="TallComponents.Web.Storage.FileSystemStorageProvider,↵
                    TallComponents.Web.PdfViewer"
              basePath="~/documents"/>
      </providers>
   </pdfStorage>
     ...
</configuration>
```

Code sample 2-3 Storage configuration in web.config

The highlighted lines were added to the web config. First, a new section is declared named *pdfStorage*. This is the section used to configure storage providers for PDFWebViewer.NET.

Then, the section itself sets the default provider to a provider named “filesystem”. This provider is declared as being of type FileSystemStorageProvider. The provider is setup to use the folder *documents* within the web application as the base path to look for PDF documents.

2.5.6. Add PDF documents

Now that storage is configured, create the *documents* folder within the root of the website project and copy a PDF document named test.pdf into it. Your project should now look similar to this:

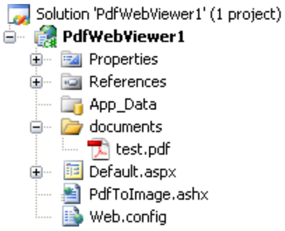


Figure 2-3 The completed project in the project explorer

2.5.7. Select document

The final step is to tell the viewer control what document to display. This can be accomplished by setting the StorageKey attribute on the control. To do this, open Default.aspx and edit the control declaration for the PdfViewer control to like this:

```
<tc:PdfViewer runat="server" id="PdfViewer1" Width="800px"
ServiceUrl="PdfToImage.ashx" StorageKey="test.pdf" />
```

Code sample 2-4 Set the document to display on the PdfViewer control

Note This sample uses the FileSystemStorageProvider that conveniently supports using the filename as a storage key.

2.5.8. Run website

Start the project. A web page will show up allowing you to view the PDF document you copied in.

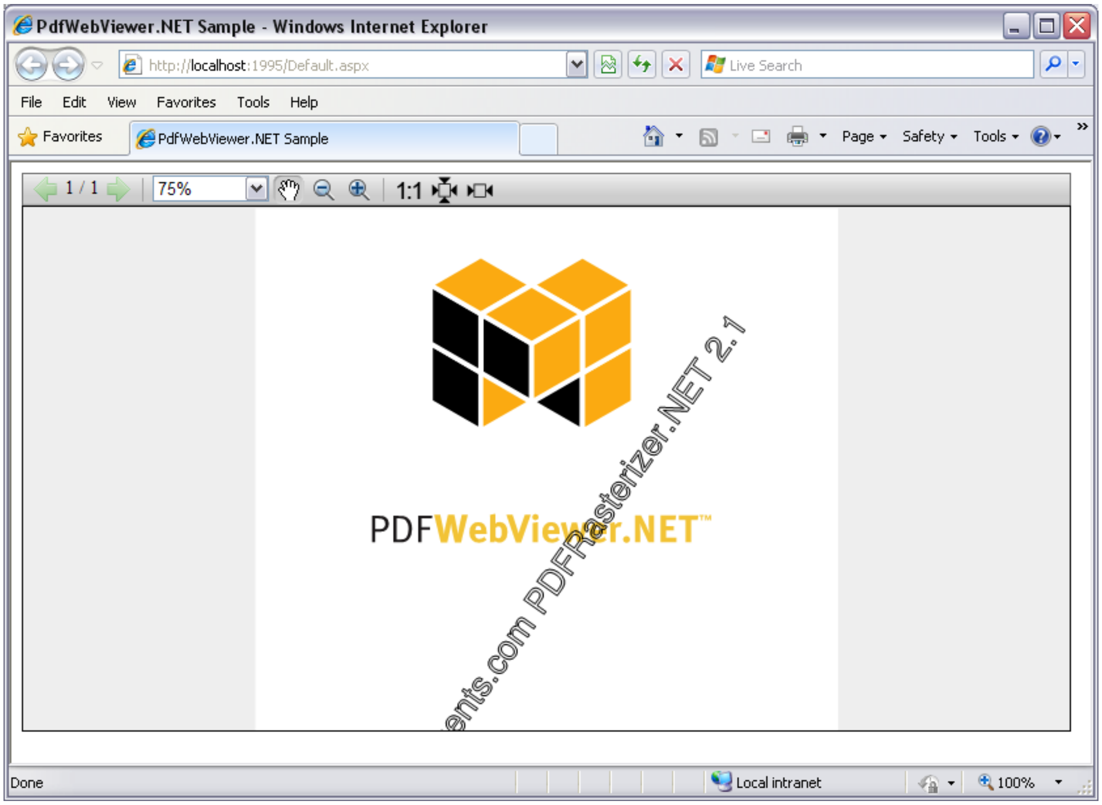


Figure 2-4 The sample application in Internet Explorer 8

3. PdfViewer

PdfViewer is the core control of PDFWebViewer.NET. It renders a view of a page in a PDF document. The control can be styled match the look of your site and you can interact with it using client script.

3.1. Using the PdfViewer control

The following code sample shows a minimal setup of a PdfViewer.

```
<tc:PdfViewer ID="pdfViewer" runat="server" ServiceUrl="pageasimage.ashx" />
```

Code sample 3-1 Minimal setup of PdfViewer.

The `ServiceUrl` property is required. It refers to the URL where the PageAsImageHandler service is registered. Please see chapter 7 for more information on this service.

3.2. Styling PdfViewer

PdfViewer can be styled to match the look of your application. The user interface of the PdfViewer is limited to the area the PDF is displayed in. The style of the area can be modified by specifying a size, a border or a background.

For example:

```
<tc:PdfViewer ID="pdfViewer" runat="server" Width="500" Height="500"
              ServiceUrl="pageasimage.ashx" BorderColor="Gray"
              BorderStyle="Solid" BorderWidth="1px"
              BackColor="#B9D1EA" />
```

Code sample 3-2 Viewer styling example.

This code sample assumes the `tc` prefix is registered either on the page or in the web.config file. Please see section 9.1 for more information on registering tag prefixes for ASP.NET.

This will give you a PdfViewer of 500 x 500 pixels with a 1 pixel solid gray border and a light blue background.

Property	Description
BackColor	Sets the background color.
BorderColor	The color for the outer border. Default is none. This value is ignored if BorderWidth and BorderStyle are not set.
BorderWidth	The width for the outer border. Default is 0px. This value is ignored if BorderColor and BorderStyle are not set.
BorderStyle	The style for the outer border. Default is none. This value is ignored if BorderColor and BorderWidth are not set.
CssClass	Sets the CSS class to assign to the toolbar. Use this to customize the look beyond what is possible through the control properties.
Height	The height of the viewer. Default is 400 pixels.
Width	The width of the viewer. Default is 400 pixels.
SearchResultCssClass	The CSS class applied to the rectangle that is used to highlight the search results. Default value is 'selection'.

Table 3-1 Style properties supported by PdfViewer.

Set the CssClass property to enable styling using CSS rules in the page or an external style sheet.

3.3. Enabling full-text search

PdfWebViewer.NET supports searching for text in PDF documents. To enable this, set the `SearchServiceUrl` property to the url of the SearchHandler service is registered.

```
<tc:PdfViewer ID="pdfViewer" runat="server" ServiceUrl="pageasimage.ashx"
SearchServiceUrl="search.ashx" />
```

Code sample 3-3 Minimal setup of PdfViewer with support for searching.

The `SearchServiceUrl` property is required if you want to use searching. Please see chapter 7 for more information on services and configuration.

3.4. Client script interaction

The PdfViewer control fully supports client script interaction. It provides information about the currently loaded document and its behavior can be controlled. The control will also report changes to its status through events.

Tip : The full type reference for PDFWebViewer.NET is available online at http://www.tallcomponents.com/pdfwebviewer1/help/

3.4.1. Getting information about the current document

The PdfViewer control provides information about the currently loaded document through client script. The document property provides an object that is similar to the TallComponents.Web.Pdf.Document object available for server programming. The code sample below demonstrates how to use this property.

```
<script type="text/javascript">
   function showDocInfo() {
      var viewer = $find('pdfViewer');
      if (viewer) {
         var doc = viewer.get_document();
         if (!doc.IsValid) {
            alert("No document loaded.");
         }
         else {
         alert(
            "Title: "+ doc.DocumentInfo.Title+
            "\nSubject: " + doc.DocumentInfo.Subject +
            "\nKeywords: " + doc.DocumentInfo.Keywords +
            "\nAuthor: " + doc.DocumentInfo.Author +
            "\nCreator: " + doc.DocumentInfo.Creator +
            "\n\nPages: " + doc.DocumentInfo.Pages.length);
         }
      }
   }
</script>
```

Code sample 3-4 Document information in client script (Upload & View demo)

Please look at the Upload & View demo to see this code in action.

3.4.2. Document object

The Document object provides essential information about a PdfDocument.

Note that if IsValid is false, the viewer is not displaying a document. This means all other properties of the document object are invalid.

Property	Type	Description
DocumentInfo	Object	Information about the PDF document (see DocumentInfo object)
FileName	String	The name of the file
IsValid	Boolean	Indicates if there is a valid document loaded. If false, none of the other properties of Document contain valid information.
Pages	Array	A list of pages (see Page object)
StorageKey	String	The unique storage key for the document

Table 3-2 Properties of the Document object in client script.

3.4.3. DocumentInfo object

The DocumentInfo object provides information about the PDF document.

Property	Type	Description
Author	String	The person who authored the PDF document.
Creator	String	The application that created the PDF document.
Keywords	String	Keywords of the PDF document.
Subject	String	The subject of the PDF document
Title	String	The title of the PDF document

Table 3-3 Properties of the DocumentInfo object in client script.

3.4.4. Page object

The Page object provides an indication of the size of a page.

Property	Type	Description
Width	Number	The width of the page in points (1 point = 1/72 inch).
Height	Number	The height of the page in points (1 point = 1/72 inch).

Table 3-4 Properties of the Page object in client script.

3.4.5. Status events

The PdfViewer control supports client script events that are fired when the control changes it's status. There are two events; *updating* and *updated*.

The *updating* event fires when the control starts updating the view. This usually means the control is about to make an out-of-band request to the server to fetch an image. Depending on bandwidth, caching and the complexity of the document this can take anywhere from a fraction of a second up to tens of seconds.

When the request completes and the view is updated, the *updated* event fires.

The main purpose of these events is to allow seamless integration of the PdfViewer control with other controls, like the PdfViewerToolbar or a custom application.

3.4.6. Searching

The PdfViewer control provides two methods that control full-text searching.

Method	Description
search	Starts or continues a search query
cancelSearch	Cancels a pending search query

The s*earch* method invokes the SearchHandler service using an asynchronous request. This request can take anywhere from a couple of milliseconds up to minutes to complete, depending on the document size and length. While a search request is pending, it may be cancelled using *cancelSearch*.

A search request starts at the current page and searches until it finds results for the query. The viewer jumps to the first result that is found highlights the text match using one or more rectangles overlayed on the page. You can set the CSS class of the rectangle using the searchResultCssClass property.

3.4.7. Search status events: searching and searchCompleted

If you are using full-text searching there are two events that help you keep track of what the PdfViewer control is doing; *searching* and *searchCompleted*.

The *searching* event fires when search query is started. The searchCompleted event fires when the request is completed or cancelled.

3.4.8. Handling events

Event handlers for the status events should have the following signature:

```
function myHandler(sender, args)
```

Code sample 3-5 Handler signature for status events.

When invoked, the `sender` parameter will reference to the viewer control. The `args` parameter is an empty instance of Sys.EventArgs (see also Sys.EventArgs.empty).

A handler for this event will typically query the viewer control for it's status.

The following code sample demonstrates how to use the status events.

```
// An event handler that makes the page red while the viewer is busy
function makePageRedWhenBusy( sender, args ) {
  var body = document.getElementsByTagName('body')[0];

  if( sender.get_isBusy() ) { // Check if the viewer is busy
    body.style.backgroundColor = "red";
  } else {
    body.style.backgroundColor = ""; // Restore background color
  }
}

// Find the PdfViewer control instance
var viewer = $find("pdfViewer");

// Register the event handlers
viewer.add_updating(makePageRedWhenBusy);
viewer.add_updated(makePageRedWhenBusy);

// Get initial status
makePageRedWhenBusy( viewer, null );
```

Code sample 3-6 Using the status events to update the user interface.

This sample demonstrates how to use the status events. The handler itself queries the viewer control to see if it is busy (i.e. loading a new image). If so, the background color of the body tag is set to red. This effectively makes the background of the whole page turn red. If the PdfViewer is not busy, the page color is restored to its default value.

Though this is not a very useful sample, it does clearly visualize the time between the start and finish of an update. This is the same mechanism the toolbar uses to display a busy image.

Next step in the sample code above is to use the $find shortcut method to find the PdfViewer control on the page. Note that the control id “pdfViewer” passed in to the $find method should match the client ID of the PdfViewer control in the page.

The event handler is registered with the control for both the updating and the updated event. Finally, the handler is invoked directly to get the initial status.

This last step is recommended because the control may be in the process of updating while the script is executing. This means the UI (in this case the page’s background color) does not correctly represent the control’s state.

4. PdfViewerToolbar

PdfViewerToolbar provides an easy to use and complete toolbar for the PdfViewer control. The control can be styled match the look of your site.

4.1. Using PdfViewerToolbar

The toolbar can be used out of the box and does not require any external files. The default appearance of the toolbar is neutral but can be customized.

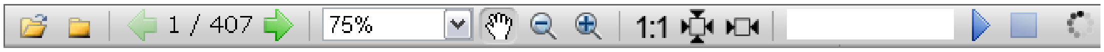


Figure 4-1 The default appearance for PdfViewerToolbar

The table below shows all the images used in the toolbar and provides a brief description.

Image	Description
📂	Triggers a file open action. Shown only when ShowFileButtons is true. This action requires a custom implementation.
📁	Triggers a file close action. Shown only when ShowFileButtons is true. This action requires a custom implementation.
⬅	Navigates to the previous page.
➡	Navigates to the next page.
✋	Switches the cursor mode to ‘pan’. The user can use the mouse to grab the page and drag it.
⊖	Switches the cursor mode to ‘zoom in’. The user can use the mouse to zoom in to a point on the page.
⊕	Switches the cursor mode to ‘zoom out’. The user can use the mouse to zoom around a point on the page.
1:1	Sets the zoom factor to 100%
✥	Enables automatic zooming to the page size.
↔	Enables automatic zooming to the page width.
▶	Starts searching. Shown only when ShowSearchBox is true.
■	Cancels a pending search request. Shown only when ShowSearchBox is true.
◌	Animated gif displayed when the viewer is busy updating.
|	Spacer image used to separate groups of buttons.
	The image that is repeated horizontally to fill the toolbar background. This image is only displayed when BackColor is not set.

Table 4-1 Toolbar buttons and images

4.1.1. Adding the PdfViewerToolbar control to a Page

The following code sample shows a minimal setup of a PdfViewer and a PdfViewerToolbar.

```
<tc:PdfViewerToolbar runat="server" PdfViewerControlId="pdfViewer" />
<tc:PdfViewer ID="pdfViewer" runat="server" ServiceUrl="pageasimage.ashx" />
```

Code sample 4-1 Minimal setup of PdfViewer and PdfViewerToolbar.

The `PdfViewerControlId` property is required and must match the ID of the PdfViewer. This setting ensures the PdfViewerToolbar is able to hook up to the PdfViewer so it can accuratly display the status of the viewer and send it commands when a button is clicked.

assumes the `tc` prefix is registered either on the page or in the web.config file. To register the prefix on the page include the following line at the top of the page.

```
<%@ Register TagPrefix="tc" Namespace="TallComponents.Web.Pdf"
Assembly="TallComponents.Web.PdfViewer" %>
```

Code sample 4-2 Register the tc tagprefix on the page

4.2. Styling PdfViewerToolbar

The PDFViewerToolbar supports some standard styling properties for ASP.NET controls. The table below lists the supported properties.

Property	Description
BackColor	Sets the background color. Setting BackColor disables the background image setting.
BorderColor	The color for the outer border. Default is none. This value is ignored if BorderWidth and BorderStyle are not set.
BorderWidth	The width for the outer border. Default is 0px. This value is ignored if BorderColor and BorderStyle are not set.
BorderStyle	The style for the outer border. Default is none. This value is ignored if BorderColor and BorderWidth are not set.
CssClass	Sets the CSS class to assign to the toolbar. Use this to cusotmize the look beyond what is possible through the control properties.
Height	The height of the toolbar. Default is 24 pixels.
Width	The width of the toolbar. Default is inherit, to make the toolbar fit the containing area.
Font.Size	The font size used in the toolbar. Default is 14 pixels.

Table 4-2 Toolbar styling properties.

Note When setting a border on a PdfViewerToolbar control this border increases the total width of the toolbar. This is a result of the way HTML and CSS layout work. To make sure the toolbar lines up with the viewer, either set a border on both controls or adjust the width of the viewer or the toolbar.

4.2.1. Using CSS to style the control

PdfViewerToolbar supports customization through CSS. The CustomStyleDemo website included with the product shows how to do this. The following code is an excerpt from default.aspx in this sample.

```
<tc:PdfViewerToolbar ID="toolbar" runat="server"
   PdfViewerControlId="pdfViewer"
   ToolbarBackgroundImageUrl="img/vista/back.gif"
   ...
   HoverButtonImageUrl="img/vista/hover.png"
   SelectedButtonImageUrl="img/vista/selected.png"
   Height="32px"
/>
```

Code sample 4-3 Excerpt from Default.aspx in CustomStyleDemo.

To keep the code sample short some of the properties have been omitted. In the code sample some custom images are set. Note that the height of the toolbar is changed to 32 pixels.

To make sure the toolbar buttons are still centered vertically an offset needs to be specified for these items. PdfViewerControl was designed using CSS so this can be done by adding a couple of style rules to the page.

The table below lists the CSS classes used by the control.

CSS Style	Description
.pdftbItem	Applied to each segment of the toolbar that is not a spacer. This includes all the buttons, the page number area and the zoom factor dropdown list.
.pdftbSpacer	Applied to spacers.

Table 4-3 Styles used by PdfViewerControl.

In the CustomStyleDemo the following rules are added at the top of the page to compensate for the increased height.

```
<style type="text/css">
      .pdftbItem { padding-top: 4px; }
      .pdftbSpacer { padding-top: 4px; }
</style>
```

Code sample 4-4 Using CSS to compensate for increased toolbar height.

The PdfViewerControl also supports the CssClass property. Setting this property will disable some of the styles applied directly to the markup. This allows full customization through CSS, for example custom background settings and applying different border colors to each side.

4.2.2. Custom image requirements

Image	Property	Description
📂	OpenImageUrl	Triggers a file open action.
📁	CloseImageUrl	Triggers a file close action.
⬅	PreviousImageUrl	Navigates to the previous page.
➡	NextImageUrl	Navigates to the next page.
✋	PanImageUrl	Switches the cursor mode to ‘pan’.
🔍	ZoomInImageUrl	Switches the cursor mode to ‘zoom in’.
🔍	ZoomOutImageUrl	Switches the cursor mode to ‘zoom out’. The user can use the mouse to zoom around a point on the page.

1:1	Zoom1on1ImageUrl	Sets the zoom factor to 100%
✥	ZoomToPageImageUrl	Enables automatic zooming to the page size.
⇹	ZoomToWidthImageUrl	Enables automatic zooming to the page width.
▶	SearchImageUrl	Starts searching.
■	CancelSearchImageUrl	Cancels a pending search request.
◌	BusyImageUrl	Animated gif displayed when the viewer is busy updating.
│	SpacerImageUrl	Spacer image used to separate groups of buttons.
	BackgroundImageUrl	The image that is repeated horizontally to fill the toolbar background. This image is only displayed when BackColor is not set.

Table 4-4 Toolbar buttons and images

The toolbar buttons can be customized by providing custom images. The toolbar is designed with images sized 24 x 24 pixels. The images should be transparent to allow the toolbar background to come through.

> **Note** 24-bit PNG images with alpha transparency may not work perfectly in all browsers, most notably Internet Explorer. IE 6 does not support PNG transparency at all. IE7 and IE8 may not display the disabled button state as expected when using alpha transparency.

Other image sizes should work but have not been thoroughly tested.

4.2.3. File buttons

There are two buttons on the toolbar that require custom actions to be implemented: the ‘open’ and ‘close’ button. These buttons are shown only when `ShowFileButtons` property is set to true. These buttons do not have a default implementation because the way the function is very application specific. For an example of how to implement these buttons, please refer to section 4.3.1.

4.2.4. Search box

If you want to allow full-text searching through the toolbar, set the `ShowSearchBox` property to true. This will show a text box and two additional buttons on the toolbar. Note that in order for search to work you also need to configure the SearchHandler service. See chapter 7 for more information.

4.3. Client script interaction

The PdfViewerToolbar fires an event in the browser when a button is clicked. This event can be handled in client script, which enables custom actions to be performed when a toolbar button is clicked.

4.3.1. buttonClick event

Event handlers for the buttonClick event should have the following signature:

```
function myHandler(sender, args)
```

Code sample 4-5 Handler signature for buttonClick event.

When invoked, the `sender` parameter will reference to the toolbar control. The `args` parameter is an instance of Sys.EventArgs. It will have a custom field, `command`, that holds the ID of the button that was clicked. The table below lists all the buttons and their command.

Image	Command	Description
📂	open	Triggers a file open action. Shown only when ShowFileButtons is true. This action requires a custom implementation.
📁	close	Triggers a file close action. Shown only when ShowFileButtons is true. This action requires a custom implementation.
⬅	prev	Navigates to the previous page.
➡	next	Navigates to the next page.
✋	pan	Switches the cursor mode to ‘pan’. The user can use the mouse to grab the page and drag it.
🔍	zoomIn	Switches the cursor mode to ‘zoom in’. The user can use the mouse to zoom in to a point on the page.
🔍	zoomOut	Switches the cursor mode to ‘zoom out’. The user can use the mouse to zoom around a point on the page.
1:1	zoom1on1	Sets the zoom factor to 100%
✥	zoomToPage	Enables automatic zooming to the page size.
⇔	zoomToWidth	Enables automatic zooming to the page width.
▶	search	Start searching, either by pressing enter in the search box or clicking the button. Shown only when ShowSearchBox is true.
■	searchCancel	Stop a pending search. Shown only when ShowSearchBox is true.

Table 4-5 Command names for toolbar buttons

Finally, the `args` parameter has a field named `suppressDefault`. This is set to `false` by default. Setting it to `true` will suppress the default action for the clicked button, allowing you to override the behavior of the toolbar.

The following code sample shows how to hook up a custom JavaScript function to handle the client-side buttonClick event.

```
<script type="text/javascript">
      Sys.Application.add_load(function() {
         $find('Toolbar').add_buttonClick(function(sender, args) {
            if (args.command == 'open') {
               $find('Modal1').show(450, 300);
            } else if (args.command == 'close') {
               window.location.href = window.location.href;
            }
         });
      });
   </script>
```

Code sample 4-6 Using the toolbar’s buttonClick event (Upload & View demo)

This snippet is an excerpt from the Upload & View sample included with the component download. It demonstrates how to register an event handler for the buttonClick event.

First, a handler is added to the Sys.Application.load event. This event fires when the document is loaded and all ASP.NET Ajax controls are ready.

The handler for the load event is an anonymous function. It uses the ASP.NET Ajax shortcut method $find to get the toolbar control named “Toolbar”. Then the actual handler for the buttonClick event is added to the toolbar through the add_buttonClick method.

The buttonClick handler then uses the `command` property to determine what button was clicked.

In this sample, the open command will show a modal dialog to allow users to upload a file. The close command will trigger a reload of the page which effectively clears the loaded document.

5. PdfThumbnailsList

PdfThumnailsList renders a list of thumbnails for each page in a document. This control is optimized to only download the thumbnails that are actually visible in the browser.

5.1. Features

The PdfThumbnailsList control is optimized to handle thumbnails intelligently. It will:

- Download thumbnails on demand to minimize server load on large documents
- Prioritize thumbnails loading; downloading thumbnails is postponed while the main viewer control is updating.
- Scale thumbnails dynamically to fit the size of the list

5.2. Styling the thumbnails list

The control can be styled using properties on the control and CSS declared in the page.

The table below lists the properties availble for styling the control.

Property	Description
BackColor	Sets the background color.
BorderColor	The color for the outer border. Default is none.
BorderWidth	The width for the outer border. Default is 0px.
BorderStyle	The style for the outer border. Default is none.
CssClass	The CSS class applied to the outer container for the thumbnails list.
Orientation	The direction for the PDF thumbnails list, either horizontal or vertical (default).
Height	The height of the list.
ThumbnailCssClass	The CSS class applied to all thumbnails.
SelectedThumbnailCssClass	The CSS class for the thumbnail that corresponds to the active page in the associated PdfViewer control.
Width	The width of the list.

5.3. Hover effect

Using CSS it's possible implement a hover effect on the thumbnails. Please consider the following snippet of CSS taken from the LibraryDemo included with the product.

```
.thumb { margin: 0px; padding:0px; }
<!-- Show a 1 pixel solid grey border around all thumbnails -->
.thumb img { border: solid 1px #999999; margin: 3px; }
<!-- Show a bright green border when the mouse hovers over it -->
.thumb:hover img { border: solid 2px #66ff33; margin: 2px; }
<!-- Show a black border on the active thumbnail -->
.active img { border: solid 2px black; margin: 2px; }
```

Code sample 5-1 CSS hover effect (from default.aspx in LibraryDemo)

The control is declared as follows:

```
<tc:PdfThumbnailsList
  ID="thumbnails"
  runat="server"
  PdfViewerControlId="pdfViewer"
  Width="600"
  Height="160"
  ServiceUrl="pageasimage.ashx"
  ThumbnailCssClass="thumb"
  SelectedThumbnailCssClass="active"
  Orientation="Horizontal" />
```

Code sample 5-2 Control declaration (from default.aspx in LibraryDemo)

The control will apply the CSS class `thumb` to all thumbnails and `active` to the thumbnail for the active page only.

These styles are used in the CSS rules shown in Code sample 2-1. The result is shown below:

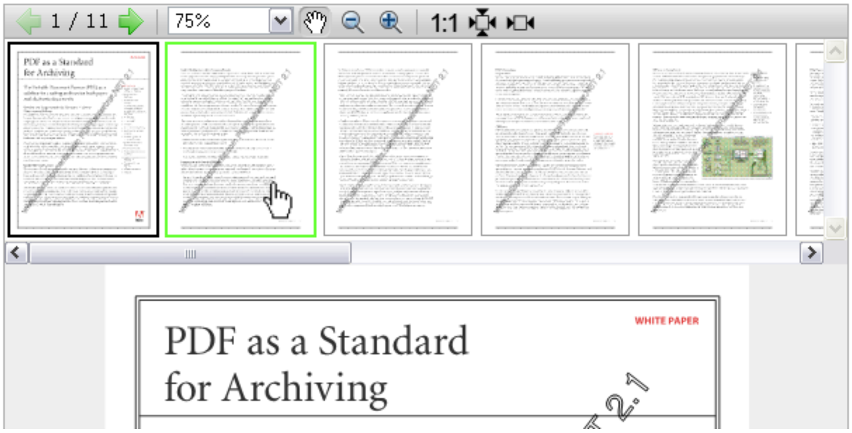


Figure 5-1 PdfThumbnailsList with CSS styling

6. Managing PDF documents

To display PDF content PDFWebViewer.NET need access to PDF documents. Included in the product is a basic storage provider that enables PDFWebViewer.NET to read files from disk.

This basic provider can be customized to integrate with your site. It is also possible to implement a custom solution that uses a different storage medium like a database.

6.1. Storage for PDF files

At the core of PDFWebViewer.NET is a service that converts PDF to images for viewing in a web browser using a set of web controls. The service and the controls need information about PDF documents. PDFWebViewer.NET provides the Document class that provides this information.

A Document instance is created from PDF file data retrieved from storage. There are however lots of ways an application can store PDF files.

PDFWebViewer.NET provides an extensible architecture that allows it to work with any ASP.NET application.

6.1.1. Using storage providers

PDFWebViewer.NET comes with ready-to-use storage managers. These managers conform to the ASP.NET provider model and are known as *storage providers*.

The storage providers share a common base class, `PdfStorageProvider`. This base class supports the following functionality:

Method	**Description**
GetDocument	Gets a single PDF document.
GetDocuments	Gets a list of all PDF documents managed by the provider.
GetPage	Gets a single page in a document.
StoreFile	Stores a file in the storage medium managed by the provider.

Table 6-1 Methods of PdfStorageProvider.

Please refer to the type reference for more details on these methods.

6.1.2. Accessing storage

Storage is accessed through the `PdfStorage` class. This class provides static properties to access all configured storage providers.

Property	**Description**
Provider	Gets the default storage provider.
Providers	Gets a collection of all configured storage providers.

Table 6-2 Static properties on PdfStorage that provide access to configured providers.

PdfStorage will take care of creating the providers based on the list of providers configured in the web.config conficuration file. Please see section 6.1.5 for more information on storage provider configuration.

6.1.3. Document and Page classes

PDFWebViewer.NET provides information about PDF documents. This information is available through the `Document`, `DocumentInfo` and `Page` classes.

The Document class represents a single PDF document. The table below lists the properties of the document class.

Property	Type	Description
DocumentInfo	DocumentInfo	Meta information contained in the document like subject and title.
FileName	String	The file name of the PDF document. This may be an empty string if the storage provider doesn't support file names.
Pages	PageCollection	The list of pages in the document.
StorageKey	String	The

Table 6-3 Properties of the Document class

DocumentInfo contains information specified in the PDF document.

Property	Type	Description
Author	String	The person who authored this document.
Creator	String	The application that created the document.
Keywords	String	The keywords for this document.
Subject	String	The subject of the document.
Title	String	The title of the document

Table 6-4 Properties of the DocumentInfo class

The Document.Pages property holds information about the number of pages (through the Pages.Count property) and the width and height og each page expressed in points.

Note Point is the basic unit of mesurement in a PDF document. 1 point equals 1/72 inch, which is approximatly 3.53 mm.

6.1.4. Storage keys

The storage providers and controls use *storage keys*. Storage keys are generated by the storage providers and can be determined through the Document.StorageKey property.

A storage key is an abstracted string value that uniquely references a document.

They exist to prevent sensitive information such as the full path of a document to be exposed to potentially malicious users.

For example, the FileSystemStorageProvider uses an MD5 hash of the path to a PDF document as the storage key.

6.1.5. Configuring a storage provider

Providers are configured through the configuration file web.config. The configuration requires two steps:

1. Register the pdfStorage section
2. Configure the storage provider

The following code sample demonstrates the changes that should be made to web.config.

```
 <configuration>
   <configSections>
      <!-- other sections and groups for ASP.NET not shown here -->
      <section name="pdfStorage"
               type="TallComponents.Web.Storage.PdfStorageProviderSection,↵
                     TallComponents.Web.PdfViewer" />
   </configSections>
   <pdfStorage defaultProvider="filesystem">
      <providers>
         <!-- clear all providers -->
         <clear />
         <!-- add a file system storage provider -->
         <add name="filesystem"
              type="TallComponents.Web.Storage.FileSystemStorageProvider,↵
                    TallComponents.Web.PdfViewer"
              basePath="~/documents"/>
      </providers>
   </pdfStorage>
     ...
</configuration>
```

Code sample 6-1 Storage configuration in web.config

This code sample lists the changes made to the configuration file in bold. This sample is limited to the relevant information only; an actual web.config file contains much more information.

At the top of the configuration file, the pdfStorage section is registered by adding a `<section>` declaration. This declaration is the same for all applications using PDFWebViewer.NET.

The pdfStorage section itself declares what provider to use by default and lists the available providers. In the code sample above a FileSystemStorage provider is configured as the default provider.

The `name` attribute is the name of the provider. This is required.

The `type` attribute is the fully qualified name of the provider.

The `basePath` attribute is specific to the FileSystemStorageProvider. It contains the path to folder where PDF documents are stored.

Note When referencing .NET types in a configuration file it is a best practice to use both the fully qualified name of the type and the name of the assembly that declares it. This speeds up processing of the configuration file and prevents problems due to types of the same name in different assemblies.

6.2. SessionStorageProvider

SessionStorageProvider is a lightweight provider that supports storing a single PDF document in session state. Due to the nature of session state this provider is limited to storing a single document per user effectively. It requires no configuration.

Note SessionStorageProvider provider is intended for testing and prototyping purposes. It should not be used in production environments.

6.3. FileSystemStorageProvider

FileSystemStorageProvider is a basic provider implementation that enables PDF documents to be read from disk.

6.3.1. Configuration settings

Setting	Description
basePath	The base path where documents are stored. This can be a virtual path like `~/Documents` or an absolute file system path like `C:\My Pdf Documents`.

Table 6-5 Configuration settings for FilesystemStorageProvider.

Note The folder pointed to by the basePath setting should be writable for the web server process if documents are being written to storage by the application (using the StoreFile method).

6.3.2. Customizing FileSystemStorageProvider

For some applications the functionality of the FileSystemStorageProvider is not sufficient. FileSystemStorageProvider can be customized by implementing a derived class.

Please refer to the type reference for for FileSystemStorageProvider for detailed information.

6.4. Implementing a custom provider

If you need to store PDF documents in a different storage medium like a database you should implement a custom storage provider. This section introduces the basic concepts.

In section 6.5 a complete sample demonstrates how to store PDF documents in a SQL Server database.

6.4.1. IPdfStorage interface

The IPdfStorage interface is the contract that all classes that manage PDF documents must implement. This is the core of the PDFWebViewer.NET extensible storage architecture.

Method	Description
GetDocument	Gets a single PDF document.
GetDocuments	Gets a list of all PDF documents managed by the provider.
GetPage	Gets a single page in a document.

Table 6-6 Methods of the IPdfStorageInterface.

6.4.2. PdfStorageProvider

The abstract base class for all storage providers in PdfStorageProvider. This class implements IPdfStorage and also provides support for adding documents to the underlaying storage.

6.5. Sample : Implement database storage (C#)

This sample demonstrates how to implement a provider that uses a database to store PDF files. Databases like Microsoft SQL Server are capable of storing large blobs of binary data in a single field. We will use this capability to build a provider that stores uploaded files in the database and makes these available for viewing.

This sample uses Linq-To-Sql to take care of retrieving and storing data in a SQL Server 2005 database but the general principals apply to other databases and data access technologies.

The complete code of this sample is included in the DatabaseStorageDemo, included with the project.

6.5.1. Start a new Web project

For this sample you will need an ASP.NET 3.5 website project. Add reference to the PDFWebViewer.NET assembly (TallComponents.Web.PdfViewer.dll) and System.Data.Linq.

See section 2.5.2 for more details on adding references to a Visual Studio project.

6.5.2. Create database and table

Create a new database named Sample.mdb in the App_Data folder.

Add a table named Files with the following fields

Field	Type	Description
ID	Int	Primary key Setup this field to be an Identity column.
fileName	Nvarchar(1024)	The name of the file.
Data	Image	The binary contents of the file.

Table 6-7 Structure of th File table in the sample database.

6.5.3. Setup data mapping

To represent the database rows in code we will create a class named DatabaseFile. To instruct Linq-To-Sql how the class maps to the database it is decorated with mapping attributes. The code sample below shows the class declaration.

```
namespace DatabaseStorageDemo.Model
{
   [Table( Name = "dbo.Files" )]
   public class DatabaseFile
   {
      [Column( Name = "ID", IsPrimaryKey = true, IsDbGenerated = true )]
      public int FileId { get; set; }
      [Column( Name = "fileName" )]
      public string FileName { get; set; }
      [Column( Name = "data" )]
      public byte[] Data { get; set; }
   }
}
```

Code sample 6-2 DatabaseFile class declaration (C#).

6.5.4. Connection string in web.config

In order to serve up PDF documents when requested to do so, the provider needs to access the database. For that, it needs a database connection string. The best place to setup connection strings is in web.config.

The following sample shows how a connection string for the sample database is declared in web.config.

```
<configuration>
  ...
  <connectionStrings>
    <clear/>
    <add name="default"
         connectionString="Data Source=.\SQLEXPRESS;↵
           AttachDbFilename=|DataDirectory|\Sample.mdf;↵
           Integrated Security=True;"/>
  </connectionStrings>
  ...
</configuration>
```

Code sample 6-3 Adding a ConnectionString to web.config.

6.5.5. Implementing a provider

Let's start implementing the provider. Create a new folder in the project named `Storage` and then create a new class named `DatabaseStorageProvider`. This class should inherit from `PdfStorageProvider`.

Through PdfStorageProvider, the provider inherits from `ProviderBase`, which has an overridable `Initialize` method. This method is invoked when the provider is created. All configuration options for the provider are passed in through the `config` parameter.

Since we need the connection string to connect to the database, we'll want a connection string specified in the configuration file. The following implementation checks if the setting is available en then stores the actual connection string for later use. It will also create a Linq-to-Sql DataContext object for reading from the database.

```
public class DatabaseStorageProvider : PdfStorageProvider
{
  public override void Initialize( string name, NameValueCollection config )
  {
    base.Initialize( name, config );

    if ( string.IsNullOrEmpty( config[ "connectionString" ] ) )
    {
       throw new ProviderException( "Missing required attribute↵
         'connectionString' on DatabaseStorageProvider." );
    }

    _connectionString = ConfigurationManager.ConnectionStrings[
           config[ "connectionString" ] ].ConnectionString;

    _context = new DataContext( _connectionString );
  }

  private DataContext _context;
  private string _connectionString;
}
```

Code sample 6-4 Initialize method for DatabaseStorageProvider.

6.5.6. Implementing storage specific methods

The code for the provider is not yet complete. Next we'll implement the storage specific methods.

Method	Description
GetDocument	Gets a single PDF document.
GetDocuments	Gets a list of all PDF documents managed by the provider.
StoreFile	Stores a file in the medium managed by the provider.

Table 6-8 Abstract methods of PdfStorageProvider.

The methods in Table 6-8 are abstract and must be implemented by our provider. Let's get started with the `GetDocuments` method.

```
public override IList<Document> GetDocuments()
{
  var query = from file in _context.GetTable<DatabaseFile>()
              orderby file.FileName
              select new Document( new MemoryStream( file.Data ),
                                   file.FileName,
                                   file.FileId.ToString() );

  return query.ToList();
}
```

Code sample 6-5 Implementation for the GetDocuments method.

This method uses a Linq query to get all files from the Files table in the sample database. The files are ordered by file name. For each row a new Document object is created.

Finally, the query is executed and the result is returned as list of documents. Note that this method should return an empty collection if no files were found.

Note The third argument supplied to constructor for Document is the storage key. In this sample we'll use the database key as the storage key. For security reasons you should obfuscate the storage key using some sort of hashing or encryption.

Next up is the `GetDocument` method. This method should return a single PDF document or `null` if the document is not available.

```
public override Document GetDocument( string key )
{
   Document result = null;
   int id = 0;
   if( Int32.TryParse( key, out id ) )
   {
      var query =
         from file in _context.GetTable<DatabaseFile>()
         where file.FileId == id
         select new Document( new MemoryStream( file.Data ),
                              file.FileName,
                              file.FileId.ToString() );

      result = query.FirstOrDefault();
   }

   return result;
}
```

Code sample 6-6 Implementation for the GetDocument method.

This method first tries to parse the supplied key into an integer value. If this succeeds, a database lookup is performed. When the query is executed either the first result is returned, or the default value for document (i.e. `null).`

Note GetDocument and GetDocuments should not throw exceptions. Return `null` or an empty collection respectively when an error occures.

Finally, the StoreFile method should write a new file to the database. This is a bit more complex.

In stead of using the shared data context used by the `GetDocument` and `GetDocuments` methods, this method creates it's own private context instead. This effectively isolates any changes made in this method from other requests.

Note A provider is created once per application domain. This means the same provider handles requests from multiple clients, possibly concurrently. Therefore you should take greate care to make sure the provider is safe for multithreading.

```
public override string StoreFile( string fileName, System.IO.Stream data )
{
  // Setup private data context
  var insertContext = new DataContext( _connectionString );
  var files = insertContext.GetTable<DatabaseFile>();

  // Create new file
  var newFile = new DatabaseFile()
    {
      FileName = fileName,
      Data = new byte[data.Length]
    };

  // Copy in file data
  if ( data.CanSeek )
  {
    data.Seek( 0, SeekOrigin.Begin );
  }
  data.Read( newFile.Data, 0, ( int )data.Length );

  // Schedule new file for insert
  files.InsertOnSubmit( newFile );

  // Insert the file
  insertContext.SubmitChanges();

  // Return the database ID as the storage key
  return newFile.FileId.ToString();
}
```

Code sample 6-7 Implementation of StoreFile

This completes the implementation of the DatabaseStorageProvider. Next we'll need to configure it.

6.5.7. Configure provider

To setup the application to use the newly created provider we'll need to configure it. Open web.config and add the bold lines from the code sample below.

```
<configuration>
   <configSections>
      <!-- other sections and groups for ASP.NET not shown here -->
      <section name="pdfStorage"
               type="TallComponents.Web.Storage.PdfStorageProviderSection,↵
                     TallComponents.Web.PdfViewer" />
   </configSections>
   <pdfStorage defaultProvider="database">
      <providers>
         <clear />
         <add name="database"
              type="DatabaseStorageDemo.Storage.DatabaseStorageProvider,↵
                    DatabaseStorageDemo"
              connectionString="default"/>
      </providers>
   </pdfStorage>
     ...
</configuration>
```

Code sample 6-8 Storage configuration in web.config

Note that the custom attribute connectionString matches the name of the setting used in the Initialize method. The value for this property should match the name of the connection string added to web.config in section 6.5.7.

6.5.8. User interface

Now that the provider is complete and configured, you can start building a user interface with the controls provided by PDFWebViewer.NET.

Please see default.aspx in the DatabaseStorageDemo, included with the product, for a demonstration of what you can do with this provider.

6.6. Controlling access to PDF documents

Sometimes it is not desirable to give unlimited access to a PDF document. In addition to document based access control that you can implement using normal ASP.NET access control methods, PDFWebViewer.NET also allows you to control access to individual pages in a document.

6.6.1. The GetPage method

Page level access control is accomplished through the GetPage method on PdfStorageProvider. As discussed elsewhere in this chapter, it's possible to create your own customized version of a storage provider.

When overriding the storage provider, it's possible to implement custom logic for the GetPage method. The LimitedAccess demo, inlcuded with the product, demonstrates how to do this.

6.6.2. Limited Access Demo

The limited access demo shows how you can replace any page in a document with another.

```
/// <summary>
/// A custom PdfStorageProvider that demonstrates how to control access
/// to pages.
/// </summary>
public class LimitedAccessStorageProvider : FileSystemStorageProvider
{
  /// <summary>
  /// Gets the page.
  /// </summary>
  /// <param name="key">The unique key for the document.</param>
  /// <param name="pageIndex">Index of the page.</param>
  /// <returns>The requested page or <c>null</c>.</returns>
  public override Page GetPage( string key, int pageIndex )
  {
    // This implementation only provide access to odd pages
    if ( ( pageIndex % 2 ) == 1 )
    {
      // even page, show the substitute page
      return GetCustomPage();

      // Note that if you return null here the page will be
      // rendered as a transparent image
      // and the thumbnail will remain blank.
    }
    else
    {
      // Odd page, show the page
      return base.GetPage( key, pageIndex );
    }
  }

  private Page GetCustomPage()
  {
    try
    {
      // Open not-accessible.pdf and return the first page
      Document doc = new Document( new FileStream(
                  HostingEnvironment.MapPath( "~/not-accessible.pdf" ),
                  FileMode.Open,
                  FileAccess.Read ), "not-accessible.pdf", "" );

      return doc.Pages[ 0 ];
    }
    catch ( Exception )
    {
      // Returning null will cause the page to be rendered as
      // a transparent image.
      return null;
    }
  }
}
```

Code sample 6-9 Controlling access to pages (LimitedAccessDemo)

7. Services

PDFWebViewer.NET provides services to provide asynchronous access to PDF content. These services must be configured correctly in order for them to work. This chapter introduces the services and shows you how to configure them.

7.1. PdfToImageHandler

PdfToImageHandler is the service that renders thumbnails and page previews. It is implemented by the `TallComponents.Web.Pdf.PdfToImageHandler` class. It is a custom Http handler.

7.2. SearchHandler

SearchHandler is the service that enables full-text searching in PDF documents. It is implemented by the `TallComponents.Web.Pdf.SearchHandler`class. This service is optional and only needs to be configured if you want to use full-text searching.

7.3. Configuring a service

In order for PDFWebViewer.NET to use the services, they must to be configured to respond to a URL. There are three ways to do this:

1. Setup a custom handler file (.ashx)
2. Configuring the handler through web.config
3. Setup routing for the handler

The third option is only available if your application is running ASP.NET 3.5 SP1 and newer.

7.3.1. Configuring a URL using a custom handler file (.ashx)

An .ashx file is represents a custom handler in ASP.NET. This means that any request for the .ashx file is handled directly by the logic in that file.

To enable this create a blank .ashx file in your project and copy in the following code snippet.

```
<%@ WebHandler Class="TallComponents.Web.Pdf.PdfToImageHandler" %>
```

Code sample 7-1 Custom handler declaration in an .ashx file for PdfToImageHandler.

This is the only code that is required in the file, don’t add anything else. The url to the file can now be used as the ServiceUrl property on the PdfViewer and PdfThumbnailsList controls. The handler requires no further configuration.

For SearchHandler the .ashx file should look like this:

```
<%@ WebHandler Class="TallComponents.Web.Pdf.SearchHandler" %>
```

Code sample 7-2 Custom handler declaration in an .ashx file for SearchHandler.

The name of the .ashx file can now be used as the SearchServiceUrl property on the PdfViewer control. For example:

```
<tc:PdfViewerToolbar
     ID="Toolbar"
     runat="server"
     PdfViewerControlId="pdfViewer"
     ShowSearchBox="true" />

<tc:PdfThumbnailsList
     ID="thumbnails"
     runat="server"
     PdfViewerControlId="pdfViewer"
     Width="600"
     Height="160"
     ServiceUrl="pageasimage.ashx"
     ThumbnailCssClass="thumb"
     SelectedThumbnailCssClass="active"
     Orientation="Horizontal"/>

<tc:PdfViewer
     ID="pdfViewer"
     runat="server"
     Width="600"
     Height="570"
     Zoom="0.75"
     ServiceUrl="pageasimage.ashx"
     SearchServiceUrl="search.ashx" />
```

Code sample 7-3 Controls using the url's declared for PDFWebViewer.NET services (Library demo)

The benefit of this method is that there is a file in the site marking the URL. Because of this, it's possible to select the file as the service url in Visual Studio.

In addition, you do not need to add additional handlers to web.config file which may not be allowed if your application is deployed to a shared hosting provider.

The downside is that there is an extra file to be managed.

7.3.2. Configuring a URL using web.config

An alternate method of configuring a URL for the PdfToImageHandler is adding a handler in web.config. The following code sample demonstrates this.

```
<configuration>
  ...
  <system.web>
    <httpHandlers>
       <!-- required -->
      <add verb="GET,HEAD"
           path="pdfasimage.ashx"
           validate="false"
           type="TallComponents.Web.Pdf.PdfToImageHandler,↵
           TallComponents.Web.PdfViewer"/>
       <!-- optional, required for searching -->
       <add verb="POST"
           path="search.ashx"
           validate="false"
           type="TallComponents.Web.Pdf.SearchHandler,↵
           TallComponents.Web.PdfViewer"/>
    </httpHandlers>
  </system.web>
  ...
</configuration>
```

Code sample 7-4 Configuring URLs for services using web.config

Though this does not give you the benefit of being able to select the URL in Visual Studio, it does not require an additional file to be added to your website.

8. Advanced configuration options

8.1. Introduction

Displaying PDF documents can be complex at times. In order to ensure correct display of your documents PDFWebViewer.NET supports external CMaps and external fonts.

8.2. External CMaps

The PDF format supports many different ways to encode text. In order to display the text PDFWebViewer.NET needs to map each character to a glyph in a font. This is done using a Character Map (CMap). The most common mappings are built into PDFWebViewer.NET. Sometimes however a specific piece of text in a document may not display in correctly because it uses a mare exotic mapping. This most commonly occurs when viewing text with non-western characters like Arab, Chinese and Japanese.

Included with the product download is a set of CMap files (Character Map). These files enable PDFWebViewer.NET to process even the more exotic mappings.

8.2.1. Using CMaps

PDFWebViewer.NET searches for CMap files in the folder `cmaps` in the root of your web application. You'll find the CMap files in the product download under `support/cmaps`. Copy the files into your site and PDFWebViewer.NET will automatically load and use the correct mapping.

Note: The entire set of CMap files will use over 9Mb of diskspace. If you know beforehand what CMaps you need you can copy only those specific files into your site.

8.3. External fonts

PDF documents can embed the fonts used within the document. This is however not required. If a font is used that is not embedded in the document, PDFWebViewer.NET will automatically look for the correct font file in specific locations.

8.3.1. Font resolving

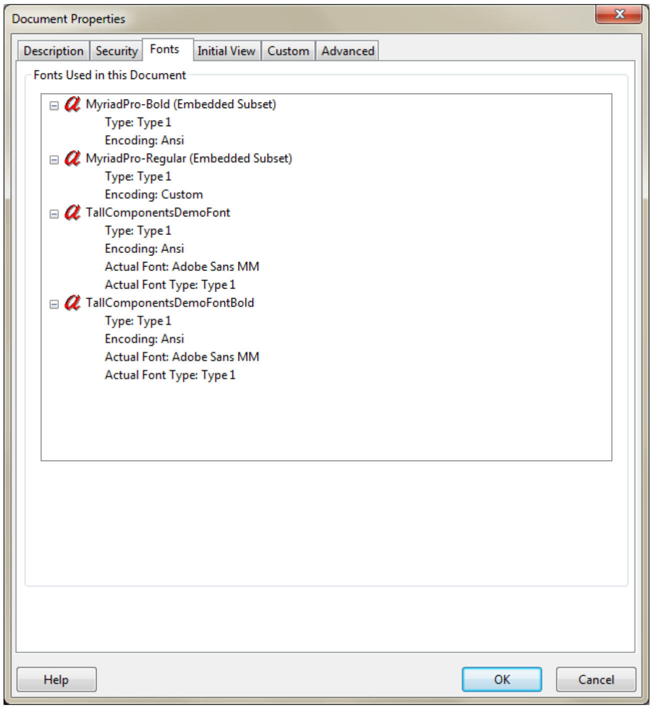
When PDFWebViewer.NET needs to render a font it will take a couple of steps to determine what font it should use. The steps are:

1. Check `fontsubstitutions.xml` in the root of you web application for a substitute font.
2. If no substitution is defined, try to use embedded font information.
3. If the font is not embedded, look for a matching font file in
   - the `fonts` folder in the root of your web application
   - the Windows fonts folder

When looking for fonts PDFWebViewer will use the name of the font as defined in the PDF document with the extension .ttf.

You can see the name of the font when you open the PDF document in Adobe Acrobat and hit Ctrl-D on the keyboard or select File › Properties from the menu. The font information is on the Fonts tab of the document properties dialog.

Note: Type 1 fonts, also known as PostScript fonts, are not supported as external fonts. Use TrueType or OpenType fonts.



Fonts Used in this Document

- MyriadPro-Bold (Embedded Subset)
  - Type: Type 1
  - Encoding: Ansi
- MyriadPro-Regular (Embedded Subset)
  - Type: Type 1
  - Encoding: Custom
- TallComponentsDemoFont
  - Type: Type 1
  - Encoding: Ansi
  - Actual Font: Adobe Sans MM
  - Actual Font Type: Type 1
- TallComponentsDemoFontBold
  - Type: Type 1
  - Encoding: Ansi
  - Actual Font: Adobe Sans MM
  - Actual Font Type: Type 1

Figuur 8-1 Font information in Adobe Acrobat

For example, in the dialog above the `TallComponentsDemoFont` is not embedded. When PDFWebViewer.NET looks for it on disk it will try to load a file named `TallComponentsDemoFont.ttf`.

8.3.2. Using fontsubstitution.xml for specific fonts

You can control the fonts used by PDFWebViewer.NET through `fontsubstitutions.xml`. The sample below demonstrates how you can specify what font files to load for the document shown in the previous section.

```
<fontsubstitutionmap version="1">
 <substitute fontname="TallComponentsDemoFont" path="PTSans.ttf" />
 <substitute fontname="TallComponentsDemoFontBold" path="PTSans-Bold.ttf" />
</fontsubstitutionmap>
```

Code Sample 8-1 Example fontsubstitution.xml

Note: If you specify a font substitution the font will always be rendered using the specified font, even if font data is embedded in the PDF document.

8.3.3. Emulating bold and italic styles

If you do not have specific fonts for each style you can tell PDFWebViewer.NET to emulate bold and/or italic. For example:

```
<fontsubstitutionmap version="1">
 <substitute fontname="DemoFont" path="Arial.ttf" />
 <substitute fontname="DemoFontBold" path="Arial.ttf" bold="1" />
 <substitute fontname="DemoFontItalic" path="Arial.ttf" italic="1" />
</fontsubstitutionmap>
```

Code Sample 8-2 Example fontsubstitution.xml

8.3.4. Specifying the default substitution font

By default PDFWebViewer.NET will display any font that it cannot find using the standard Times font found on any Windows system. You can override the default font through `fontsubstitutions.xml`. The example below sets the Arial unicode font as the default substitution font.

```
<fontsubstitutionmap version="1">
 <defaultsubsitutionfont>arialuni.ttf</defaultsubsitutionfont>
</fontsubstitutionmap>
```

Code Sample 8-3 Setting the default font in fontsubstitution.xml

Note: Make sure you pick a default font that supports all characters in the documents that you are going to display. Characters that are not in the font will not be displayed correctly.

9. Appendix A : Tips

9.1. Registering a tag prefix for ASP.NET

To register a tag prefix for the PDFWebViewer.NET controls for a single page include the following line at the top of the page.

```
<%@ Register TagPrefix="tc" Namespace="TallComponents.Web.Pdf"
Assembly="TallComponents.Web.PdfViewer" %>
```

Code sample 9-1 Registering the tc tag prefix for a single page

To register the prefix for the entire application, locate the controls section section in web.config and add the line highlighted in the code sample below.

```
<configuration>
  ...
  <system.web>
    <pages>
      <controls>
        ...
        <add tagPrefix="tc" namespace="TallComponents.Web.Pdf"
             assembly="TallComponents.Web.PdfViewer"/>
      </controls>
    </pages>
  </system.web>
</configuration>
```

Code sample 9-2 Registring the tc tag prefix in web.config