ASP.NET Core PDF Viewer User Guide

TerminalWorks
ASP.NET Core PDF Viewer
Usage Guide

Contents

About

TerminalWorks ASP.NET Core PDF Viewer is a web component for PDF display in ASP.NET Core applications.

Our component is based on reliable Mozilla PDF viewer pdf.js https://mozilla.github.io/pdf.js

Official web page: https://www.terminalworks.com/web-pdf-viewer

Requirements

ASP.NET Core 3.1 or newer.

It is a Razor class library and supported ASP.NET Core tecnologies are: Blazor, RazorPages, MVC and WebAssembly.
It is build as a .NET Standard 2.1.

Supported browsers

All modern browsers, including IE11, are supported.
IE10 and older IE versions aren't supported.
Safari is supported from version 11 and above.

Supported functionalities

- mobile responsive
- accessiblity friendly
- standard viewer features
- easy to translate
- customizable themes

Standard viewer features

- open and download PDF
- display annotations
- zoom
- single/multi-page view
- select texts
- rotate
- print
- bookmarks
- page navigation
- search PDF

Licensing

If not licensed, TWPdfViewer will be fully functional but it will show Demo watermark.
License is valid for 1 year free upgrades and official support.
After that, TWPdfViewer will work for all the versions of the component which were available at the time of licensing.
It is possible to buy additional license upgrade.

How to apply license?

For your project, it is enough that you call this code just once, but before TWPdfViewer component is used.
If you don't apply a correct license, demo watermark will be shown.

```
using Terminalworks.PdfViewer.AspNetCore;

var licensing = new Licensing(companyName, licenseKey);
if (!Licensing.IsLicensed) {
  var error = Licensing.LicenseStatus; // information why it isn't
valid
}
```

Installation

This installation guide will guide you trough all the needed steps to get started with ASP.NET Core PDF Viewer.
Please make sure to follow the instructions depending on the technology you're using in your project.

Make sure to add a reference to Terminalworks.PdfViewer.AspNetCore component. You can reference it through NuGet.

Run the following command in the Package Manager Console:

```
install-package Terminalworks.PdfViewer.AspNetCore
```

Each installation, no matter the technology you're using has 3 simple steps:

1. Add link to needed CSS files
2. Add link to needed Javascript files
3. Add the component to the page

RazorPages

Steps for using TWPdfViewer component in Razor Pages

1. **In your HTML section of the page, you will need to add 2 CSS links.**

File location: *Pages\Shared_Layout.cshtml*

```
<head>
  <!-- your other head content -->
  @RenderSection("Head", required: false)
</head>
```

File location: *Pages\{YOUR_PAGE}.cshtml* (e.g. Index.cshtml)

```
@section Head {
  <link rel="stylesheet"
  href="/_content/Terminalworks.PdfViewer.AspNetCore/css/tw-pdf-
  viewer.css" />
  <link rel="stylesheet" id='tw-web-theme'
     href="/_content/Terminalworks.PdfViewer.AspNetCore/css/themes/
     white-shapes/tw-pdf-viewer-theme.css" />
}
```

tw-pdf-viewer.css is a general CSS of TWPdfViewer and we recommend that you don't change it.
tw-pdf-viewer-theme.css is theme specific.
You can read more about changing theme at Changing the theme

2. **Add needed Javascript as shown in the example below**

File location: Pages\Shared_Layout.cshtml

```
<head>
  <!-- your other head content -->
  @RenderSection("Scripts", required: false)
</head>
```

File location: *Pages\{YOUR_PAGE}.cshtml* (e.g. Index.cshtml)

```
    <body>
      <!-- your other body content -->
      @section Scripts {
        <script  src='/_content/Terminalworks.PdfViewer.AspNetCore/js/
        pdf.js'></script>
        <script src='/_content/Terminalworks.PdfViewer.AspNetCore/js/
        tw_viewer.js'></script>
      }
    </body>
```

3. **Add the component to your page**

File location: *Pages\{YOUR_PAGE}.cshtml* (e.g. Index.cshtml)

```
@using PdfViewer = Terminalworks.PdfViewer.AspNetCore

<component
  type="typeof(PdfViewer)"
  param-Id="@("twPdfViewer1")"
  render-mode="Static"
  param-IsStatic='@("Yes")'
/>
```

For examples how to set initial parameters look at Viewer options

MVC

Steps for using TWPdfViewer component in MVC

1. **In your HTML section of the page, you will need to add 2 CSS links.**

File location: *Views\Shared_Layout.cshtml*

```
<head>
  <!-- your other head content -->
  @RenderSection("Head", required: false)
</head>
```

File location: *Views\Home\{YOUR_VIEW}.cshtml* (e.g. Index.cshtml)

```
@section Head {
  <link rel="stylesheet"
  href="/_content/Terminalworks.PdfViewer.AspNetCore/css/tw-pdf-
  viewer.css" />
  <link rel="stylesheet" id='tw-web-theme'
  href="/_content/Terminalworks.PdfViewer.AspNetCore/css/themes/
  white-shapes/tw-pdf-viewer-theme.css" />
}
```

tw-pdf-viewer.css is a general CSS of TWPdfViewer and we recommend that you don't change it.
tw-pdf-viewer-theme.css is theme specific.
You can read more about changing theme at Changing the theme

2. Add needed Javascript as shown in the example below

File location: Views\Shared_Layout.cshtml

```
<head>
  <!-- your other head content -->
  @RenderSection("Scripts", required: false)
</head>
```

File location: Views\Home\{YOUR_VIEW}.cshtml (e.g. Index.cshtml)

```
<body>
  <!-- your other body content -->
  @section Scripts {
    <script src='/_content/Terminalworks.PdfViewer.AspNetCore/js/
    pdf.js'></script>
    <script src='/_content/Terminalworks.PdfViewer.AspNetCore/js/
    tw_viewer.js'></script>
  }
</body>
```

3. Add component to your page

File location: Views\Home\{YOUR_VIEW}.cshtml (e.g. Index.cshtml)

```
@using PdfViewer = Terminalworks.PdfViewer.AspNetCore

<component
  type="typeof(PdfViewer)"
  param-Id="@("twPdfViewer1")"
  render-mode="Static"
  param-IsStatic='@("Yes")'
/>
```

Blazor

Steps for using TWPdfViewer component in Blazor

1. **In your HTML section of the page, you will need to add 2 CSS links.**

File location: *Pages\{YOUR_PAGE}.razor* (e.g. Index.razor)

```
@section Head {
  <link rel="stylesheet"
  href="/_content/Terminalworks.PdfViewer.AspNetCore/css/tw-pdf-
  viewer.css" />
  <link rel="stylesheet" id='tw-web-theme'
     href="/_content/Terminalworks.PdfViewer.AspNetCore/css/themes/
     white-shapes/tw-pdf-viewer-theme.css" />
}
```

tw-pdf-viewer.css is a general CSS of TWPdfViewer and we recommend that you don't change it.
tw-pdf-viewer-theme.css is theme specific.
You can read more about changing theme at Changing the theme

2. **Add needed Javascript as shown in the example below**

File location: Pages_Host.cshtml

```
<body>
  <!-- your other body content -->
  <script src='/_content/Terminalworks.PdfViewer.AspNetCore/js/
  pdf.js'></script>
  <script src='/_content/Terminalworks.PdfViewer.AspNetCore/js/
  tw_viewer.js'></script>
</body>
```

3. **Add the component to your page**

File location: Pages\{YOUR_PAGE}.razor (e.g. Index.razor)

```
@using TWWebCorePdfViewer.Pages.Shared;
<TWPdfViewer />
```

If you use render-mode="Static" to make it work you need to add IsStatic="@("Yes")"

```
@using TWWebCorePdfViewer.Pages.Shared;
<TWPdfViewer IsStatic="@("Yes=")" />
```

For examples how to set initial parameters look at Viewer options

WebAssembly

Steps for using TWPdfViewer component in WebAssembly

1. **In your HTML section of the page, you will need to add 2 CSS links.**

File location: *wwwroot\{YOUR_PAGE}.html* (e.g. index.html)

```
<link rel="stylesheet"
href="/_content/Terminalworks.PdfViewer.AspNetCore/css/tw-pdf-
viewer.css" />
<link rel="stylesheet" id="tw-web-theme"
href="/_content/Terminalworks.PdfViewer.AspNetCore/css/themes/light/t
w-pdf-viewer-theme.css" />
```

tw-pdf-viewer.css is a general CSS of TWPdfViewer and we recommend that you don't change it.
tw-pdf-viewer-theme.css is theme specific.
You can read more about changing theme at Changing the theme

1. Add needed Javascript as shown in the example below

File location: wwwroot\{YOUR_PAGE}.html (e.g. index.html)

```
<body>
  <!-- your other body content -->
  <script src='/_content/Terminalworks.PdfViewer.AspNetCore/js/
  pdf.js'></script>
  <script src='/_content/Terminalworks.PdfViewer.AspNetCore/js/
  tw_viewer.js'></script>
</body>
```

2. Add the component to your page

File location: Pages\{YOUR_PAGE}.razor (e.g. Index.razor)

```
@using TWWebCorePdfViewer.Pages.Shared;
<TWPdfViewer IsStatic="@("Yes=")" />
```

For examples how to set initial parameters look at Viewer options

Blazor component and Blazor rendering modes

Our TWPdfViewer is a Razor class library component.
It has 3 possible render modes in Blazor server app or in your ASP.NET Core RazorPages/MVC web applications

1. ServerPrerendered

Renders the component into static HTML and includes a marker for a Blazor Server app. When the user-agent starts, this marker is used to bootstrap a Blazor app.

2. Server

Renders a marker for a Blazor Server app. Output from the component isn't included. When the user-agent starts, this marker is used to bootstrap a Blazor app.

3. Static

Renders the component into static HTML.

For WebAssembly app, from .NET 5, for hosted WebAssembly, available render modes are:

1. WebAssembly

Renders a marker for a Blazor WebAssembly app for use to include an interactive component when loaded in the browser. The component isn't prerendered. This option makes it easier to render different Blazor WebAssembly components on different pages.

2. WebAssemblyPrerendered

Prerenders the component into static HTML and includes a marker for a Blazor WebAssembly app for later use to make the component interactive when loaded in the browser.

Static rendering

TWPdfViewer doesn't have any need for interactivity with the server so recommended way is to use Static rendering mode (for Blazor Server app) or WebAssembly render mode (for WebAssembly web apps), but it will work also for other render modes.

In case static render is used, it is a mandatory to set additional parameter IsStatic="Yes".

More about integrating components from official documentation: Component tag helper

IE11 specifics

IE11 and Blazor/WebAssembly

- IE11 doesn't support WebAssembly at all.
- IE11 doesn't support by default Blazor, but it is possible with a polyfill. In our testing projects we used the polyfill found at: https://github.com/Daddoon/Blazor.Polyfill

IE11 and our WEB PDF viewer known issues

- Bookmark view can't be resized and has fixed width size

Viewer Options

TWPdfViewer component has two parameters:

1. *IsStatic*
2. *Options*

IsStatic property is only important if *render-mode* is set to *Static*.
In that case, that property **must be** set to *"Yes"*.
For remaining two rendering modes: *Server* and *ServerPrerender* that property can be omitted.

Setting properties is done through *Options* parameter.

Example for RazorPages/MVC:

```
@using PdfViewer = Terminalworks.PdfViewer.AspNetCore
<component
  type="typeof(PdfViewer.Pages.Shared.TWPdfViewer)"
  render-mode="Static"
  IsStatic="Yes"
  param-Options='@new PdfViewer.Options {
        Bookmark = new PdfViewer.BookmarkOptions {
                ShowBookmarksOnOpen = true,
                BookmarksFullyExpanded = false,
                PreserveBookmarksState = true
            }
    }
/>
```

Another way of defining options is in C# part:

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;
var options = new PdfViewer.Options {
  Bookmark = new PdfViewer.BookmarkOptions {
    ShowBookmarksOnOpen = true,
    BookmarksFullyExpanded = false,
    PreserveBookmarksState = true
  }
};
```

And how to use them in Blazor app:

```
<TWPdfViewer Options='@options' />
```

Rest of examples, for brevity, will show only how to change Options in C# code.
How to apply it depends if you are using RazorPages/MVC component (*param-Options='@options'*) or Blazor/WebAssembly (*Options='@options'*)

How to open document automatically

Please note that application of the options described in this section depend on the initial set up explained here.

In this example it will load document at web site's location *test-pdf/pdfprint-manual.pdf* and show its second page.
Notes:

- if document is password protected it will show password prompt before loading the document
- if external site allows CORS, it is also possible to set external URL

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  DocumentUrl = "test-pdf/pdfprint-manual.pdf",
  InitialPageNumber = 2
};
```

This example will load password protected document if the password is correct from the secured document url.

SecuredDocumentUrl, also as password, is sent only for non-Static rendering.

Differences between DocumentUrl and SecuredDocumentUrl:

- DocumentUrl is visible in browser inspect element and SecuredDocumentUrl isn't
- DocumentUrl can be used for every blazor rendering type and SecuredDocumentUrl can't be used for a Static rendering

Example for RazorPages/MVC:

```
@using PdfViewer = Terminalworks.PdfViewer.AspNetCore
<component
  type="typeof(PdfViewer.Pages.Shared.TWPdfViewer)"
  render-mode="Server"
  param-SecuredDocumentUrl='@("test-pdf/password-protected.pdf")',
  param-Password = '@("testPassword")'
/>
```

Switching between single and multi page view

Please note that application of the options described in this section depend on the initial set up explained here.

This example will open viewer in multi-page view mode.
ViewerType has a default value of *PdfViewer.ViewerTypes.MultiPageViewerMulti*.

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  ViewerType = PdfViewer.ViewerTypes.MultiPageViewer
};
```

This example will open viewer in single-page view mode, which means it will show one page at a time.

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  ViewerType = PdfViewer.ViewerTypes.SinglePageViewer
};
```

Setting available zoom options

Please note that application of the options described in this section depend on the initial set up explained here.

In this example:

- it will show page in 70% of an original size (*ZoomValue* default value is *100*)

If *IsZoomFitToPage == true*, it would ignore *ZoomValue* property and it would calculate page size according to the viewer component size. *IsZoomFitToPage* default value is *true*.

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  Zoom = new PdfViewer.ZoomOptions {
    IsZoomFitToPage = false,
    ZoomValue = 70
  }
};
```

Setting available toolbar options

Please note that application of the options described in this section depend on the initial set up explained here.

This is an example which shows all available toolbar options and its default values.
If you are fine with the default value, you can omit it, otherwise set it to *false*.

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  Toolbar = new PdfViewer.ToolbarOptions {
    IsBookmarksVisible = true,
    IsDocumentInfoVisible = true,
    IsMultiViewVisible = true,
    IsNextPageVisible = true,
    IsNextVisitedVisible = true,
    IsOpenVisible = true,
    IsPageNumberVisible = true,
    IsPreviousPageVisible = true,
    IsPreviousVisitedVisible = true,
    IsPrintVisible = true,
    IsRotateClockwiseVisible = true,
    IsRotateCounterClockwiseVisible = true,
    IsDownloadVisible = true,
    IsCloseVisible = true,
    IsSearchVisible = true,
    IsSingleViewVisible = true,
    IsToolbarVisible = true,
    IsTooltipVisible = true,
    IsZoomDropDownVisible = true,
    IsZoomInVisible = true,
    IsZoomOutVisible = true
  }
};
```

Example how to hide *Bookmark* and *Document info buttons* from the toolbar:

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  Toolbar = new PdfViewer.ToolbarOptions {
    IsBookmarksVisible = false,
    IsDocumentInfoVisible = false
  }
};
```

How to enable or disable text selection

Please note that application of the options described in this section depend on the initial set up explained here.

By default, *TextSelectionDisabled* has value *false* so text selection is enabled.
This example will disable mouse text selection.

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  TextSelectionDisabled = true
};
```

How to change or translate component built-in texts

Please note that application of the options described in this section depend on the initial set up explained here.

More in detail at -> Translating UI and messages

Setting available find options

Please note that application of the options described in this section depend on the initial set up explained here.

This is an example which show all available *FindOptions* properties and its default values.
If you are fine with the propertie's default value, you can omit it, otherwise set it to *true*.

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  Find = new PdfViewer.FindOptions {
    CaseSensitive = false,
    HighlightAll = false,
    WholeWord = false
  }
};
```

This example shows how to have case sensitive search.

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  Find = new PdfViewer.FindOptions {
    CaseSensitive = true
  }
};
```

Setting available bookmarks options

Please note that application of the options described in this section depend on the initial set up explained here.

In this example:

- bookmarks will be shown automatically on document load if the document have it (*ShowBookmarksOnOpen* default value is *false*)
- shown bookmarks will not be in fully expanded state (*BookmarksFullyExpanded* default value is *false*)
- once closed bookmarks on reopen will preserve expanded bookmark items (*PreserveBookmarksState* default value is *false*)

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

var options = new PdfViewer.Options {
  Bookmark = new PdfViewer.BookmarkOptions {
    ShowBookmarksOnOpen = true,
    BookmarksFullyExpanded = false,
    PreserveBookmarksState = true
  }
};
```

Customization

ASP.NET Core PDF Viewer comes with various options of customization. Starting from changing the fonts to suit your needs all the way to adjusting the whole theme. You can explore all the options in the guides attached to this section.

Changing the theme

You can use one of already built-in themes or you can create yours.

In current version, available built-in themes are:

- *blue-circle*
- *blue-element*
- *blue-grey*
- *colorful-morphism*
- *dark-cyan*
- *dynamic-purple*
- *light*
- *purple-blue*
- *red-purple*
- *forest-brown*
- *white-shapes*

List of available built-in themes can be obtained by calling from Javascript:

```
window.TWPdfViewerUtil.getBuiltInThemes();
```

Must have *id="tw-web-theme"*, otherwise it will not work changing theme through Javascript.

There are two ways to change the theme:

1. **Directly in HTML**

```
<link rel="stylesheet"
href="/_content/Terminalworks.PdfViewer.AspNetCore/css/themes/{THEME_
NAME}/tw-pdf-viewer-theme.css" id="tw-web-theme" />
```

Example:

```
<link rel="stylesheet"
href="/_content/Terminalworks.PdfViewer.AspNetCore/css/themes/white-
shapes/tw-pdf-viewer-theme.css" id="tw-web-theme" />
```

Or if you're using your custom theme:

```
<link rel="stylesheet" href="{URL_TO_YOUR_CSS_FILE}" id="tw-web-
theme" />
```

2. From Javascript

Using one of the built-in themes:

```
window.TWPdfViewerUtil.changeThemeByName("{THEME_NAME}");
```

Example:

```
window.TWPdfViewerUtil.changeThemeByName("blue-circle");
```

Or if you are using your custom theme:

```
window.TWPdfViewerUtil.changeThemeBySrcUrl("{URL_TO_YOUR_CSS_FILE}");
```

Example:

```
window.TWPdfViewerUtil.changeThemeBySrcUrl("/css/custom-theme.css");
```

In both ways, every instance of TWPdfViewer on-page will be affected.

Creating custom CSS theme

If you would like to use yours icons for the buttons or/and change colors of the TWPdfViewer, you can. Take some of the existing CSS themes (like *css/light/tw-pdf-viewer-light.css*) as a template, change its colors and src to your image buttons files.

On your HTML page, instead of link to:

```
<link rel="stylesheet" id="tw-web-theme"
href="/_content/Terminalworks.PdfViewer.AspNetCore/css/tw-pdf-viewer-
light-theme.css">
```

Use:

```
<link rel="stylesheet" href="{path_to_your_css}" />
```

Note: You can use multiple instances of the viewer on the same page, but it isn't possible to use different CSS theme. CSS viewer theme is loaded per page and affects all the instances.

Managing fonts

The main font used by the component is: ***Arial***

The fallback fonts are: *Verdana, Helvetica, Tahoma, Helvetica Neue, -apple-system, Liberation Sans*

Arial, Verdana, Helvetica, Tahoma, Helvetica Neue are available on Windows and the latest versions of OS X.
Arial is also available on certain Linux distributions.
In case *Arial* and all the consecutive fonts are not available on Linux, the fallback font will be *Liberation Sans* which is the metric equivalent of *Arial* font.
-apple-system is here just to make sure older versions of OS X are covered in case the preceding fonts are not available.
This way we made sure our component will look and feel the same on a wide array of operating systems and their versions.

The class where fonts are defined is the following and you can find it in tw-pdf-viewer.css file:

```
.tw2018elem_mainContainer[data-tw2018-pdf-viewer] {
  text-align: left;
  position: relative;
  font-family: Arial, "Verdana, Helvetica, Tahoma, "Helvetica
Neue",
  -apple-system, "Liberation Sans";
  font-size: 14px;
  font-weight: 400;
}
```

In case you wish to load your custom font, first, you need to specify your own font-family name using @font-face like in the example below:

```
@font-face{
  font-family: myAwesomeCustomFont;
  src: url(perfect_grey.woff)
}
```

Afterwards, define your custom font in *.tw2018elem_mainContainer[data-tw2018-pdf-viewer]* class, make sure to position your custom font first when defining it in the font-family property:

```
.tw2018elem_mainContainer[data-tw2018-pdf-viewer] {
    text-align: left;
    position: relative;
    font-family: myAwesomeCustomFont, Arial, Verdana, Helvetica,
Tahoma, "Helvetica Neue",
    -apple-system, "Liberation Sans";
    font-size: 14px;
    font-weight: 400;
}
```

If you'd like to use your own custom font on a few components only, make sure to override the default font-family directly on those components by creating your own custom class or adding the custom font-family in already available CSS on that component.

Translating UI and messages

By default, text is in English. It can be ovveriden.

There are five types of text:

- Aria-labels for accessibility
- Toolbar button's tooltips
- Messages
- Document info
- Miscellaneous text

TWPdfViewer has a property *Options* which has a property *Texts*.
Texts has properties:

- *Messages*
- *DocumentInfoTexts*
- *MiscTexts*
- *AriaLabels*
- *Tooltips*.

DocumentInfoText properties and its default values:

Author = "Author:"
CreationDate = "CreationDate:"
Creator = "Creator:"
FileName = "FileName:"
FileSize = "FileSize:"
Keywords = "Keywords:"
ModificationDate = "ModificationDate:"
PageCount = "Page Count:"
PageSize = "Page Size:"
PDFProducer = "PDF Producer:"
PDFVersion = "PDF Version:"
Subject = "Subject:"
Title= "Title:"

AriaLabels properties and its default values:

CloseBookmarks = "Close bookmarks"
ClosePdfDocumentBtn = "Close PDF document"
CurrentPage = "Current page"
CurrentZoomValue = "Current zoom value"
DocumentInfoBtn = "Document information"
DownloadBtn = "Download PDF document"
NextFindBtn = "Next find"
NextPageBtn = "Next page"
NextVisitedPageBtn = "Next visited page"
MessageCloseBtn = "Message close"
MultiPageViewBtn = "Multi page view"
OpenBtn = "Open PDF document"
PasswordField = "Password"
PreviousFindBtn = "Previous find"
PreviousPageBtn = "Previous page"
PreviousVisitedBtn = "Previous visited page"
PrintBtn = "Print PDF document"
RotateClockwiseBtn = "Rotate clockwise"
RotateCounterClockwiseBtn = "Rotate counter-clockwise"
SearchBtn = "Search"
SearchOptionsBtn = "Search options"
SearchTerm = "Search term"
ShowBookmarksBtn = "Show bookmarks"
SinglePageViewBtn = "Single page view"
ZoomDropDownArrow = "Open zoom drop-down"
ZoomInBtn = "Zoom in"
ZoomOutBtn = "Zoom out"

MiscText properties and its default values:

CancelBtn = "Cancel"
CloseBtn = "Close"
EnterPassword = "Enter the password to open this PDF File"
FindInDocument = "Find in document..."
HighlightAll = "Highlight all"
MatchCase = "Match case"
OKBtn = "OK"
ZoomActualSize = "Actual page size"
ZoomPageLevel = "Zoom to page level"
WholeWord = "Whole word"

Tooltips properties and its default values:

ClosePdfDocumentBtn = "Close PDF document"
DocumentInfoBtn = "Document information"
DownloadBtn = "Download PDF document"
MultiPageViewBtn = "Multi page view"
NextPageBtn = "Next page"
NextVisitedPageBtn = "Next visited page"
OpenBtn = "Open PDF document"
PreviousPageBtn = "Previous page"
PreviousVisitedBtn = "Previous visited page"
PrintBtn = "Print PDF document"
ShowBookmarksBtn = "Show bookmarks"
SinglePageViewBtn = "Single page view"
RotateClockwiseBtn = "Rotate clockwise"
RotateCounterClockwiseBtn = "Rotate counter-clockwise"
ZoomInBtn = "Zoom in"
ZoomOutBtn = "Zoom out"

Messages properties and its default values:

FindReachedBottom = "Reached end of document, continued from top."
FindReachedTop = "Reached top of document, continued from bottom."
InvalidPassword = "Wrong password, please try again."
LoadDocumentError = "Load document error:"
PrintingPopupWarning = "For direct printing, popup must be allowed."
PreparingForPrint = "Preparing pages for print:"
TextNotFound = "Text not found."
UnknownError = "Unknown error"

Example

Example how to change MiscText.FindInDocument in Razor pages

```
using PdfViewer = Terminalworks.PdfViewer.AspNetCore;

<component type="typeof(PdfViewer.Pages.Shared.TWPdfViewer)"
render-mode="Static"
param-IsStatic='@("Yes")'
param-Options='@new PdfViewer.Options {
  Texts = new PdfViewer.ViewerTexts {
    MiscText = new PdfViewer.Texts.MiscText {
    FindInDocument = "Trova in documento"
      }
    }
}'/>
```

Another way of changing texts – UserDefined property dictionary

This could be useful, if you have different files with translations for different languages

```
var options = new Terminalworks.PdfViewer.AspNetCore.Options();
var userDefined = new Dictionary<string, string>
{
  { TextCodeNames.AriaLabels.CurrentPage, "La pagina attuale" }
  // you can override all or just some texts...
};
options.Texts = new ViewerTexts
{
  UserDefined = userDefined
};
```