www.doconut.com

Integration Guide

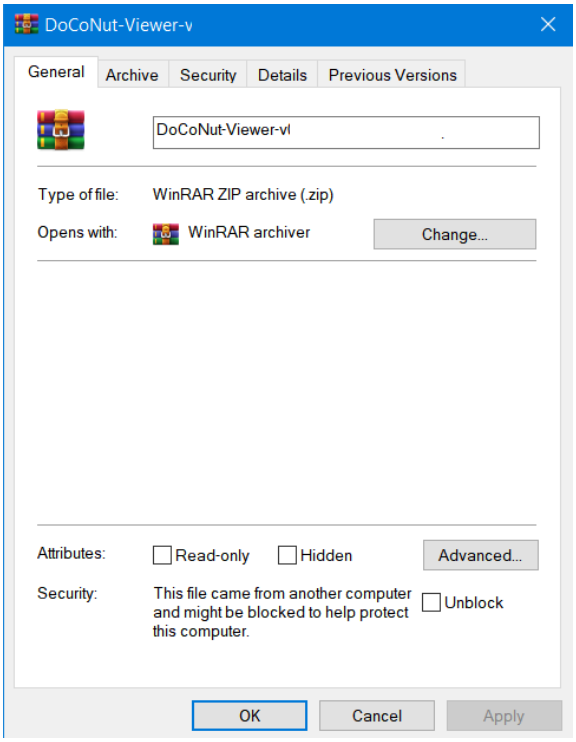
February 2023

Copyright 2022 www.doconut.com

Contents

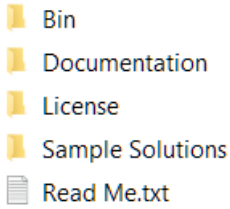
Trial Version

Once you download the zip files please right click on the zip file, open properties window and unblock the zip. Then extract it.



If you just see an unblock checkbox, then check it and click OK.

Once you extract the zip file you will see following folders inside it.



The sample solutions (projects) do not have the required DLLs referenced. Hence you need to either copy them to bin folder or once you open the solution from Visual Studio, browse add reference to them.

License File

Next you will need to copy the "Doconut.lic" file from License folder to the "bin" folder of your solution. You can specify a custom license path in web.config using <add key="DoconutLicensePath" value="~/Licenses" /> .For .Net Core keep it in "wwwroot" folder. The license works for "localhost". There is also a license for Microsoft Azure "azurewebsites.net" provided inside the License folder.

This is a fully functional trial for 60 days. Trial version shows watermarks on the document. Professional and Distribution license allows you to embed the license information in the code itself.

If you want to test your application on a stating or test server then please email us the URL and we will issue a trial license for that server.

Asp.Net

DLL Reference

Before running or compiling the visual studio solution, please ensure that you have added reference to DocumentViewer.dll, DocumentFormats.dll and DocumentConfig.dll. Secondly the Doconut.lic license file should be copied to "bin" folder.

Viewer Setup

Doconut viewer has a very flexible UI. All menus, function calls are external to the viewer. The viewer has just the thumbnail view on left and main view on right. You can also hide/show thumbnails as required. Please refer Doconut Manual PDF for more details.

This example is as per the "Viewer.aspx" sample page in Asp.Net sample solution.

The viewer has 100% width and 100% height by default. Thus it takes up the dimensions of the enclosing parent element.

```
<!-- Document Viewer Starts -->
<div id="divDocViewer">
    <asp:DocViewer ID="ctlDoc" runat="server" ShowThumbs="true" Au
        AutoFocusPage="false" Zoom="100" IncludeJQuery="false"  Ir
        ImageResolution="200" TimeOut="15" WatermarkInfo="^Sample
</div>
<!-- Document Viewer Ends -->
```

As shown in this image, divDocViewer DIV element encloses the DocViewer control. There is a css class defined on the page for divDocViewer as shown below:

```
#divDocViewer {
    width: 100%;
    border-radius: 6px;
    background-color: whitesmoke;
    border: 1px grey;
}
```

This takes care of the viewer's width. You can change the width property as required. For height, it is set via JavaScript. This also helps in making the viewer responsive as the browser resizes or rotates.

```
var height = "innerHeight" in window ? window.innerHeight : document.documentElement.offsetHeight;
$("#divDocViewer").height(height - 80);
```

You can adjust the height value as required by your page. Please refer Viewer.aspx sample for further details.

```
// If you want to resize viewer with window resizing
$(window).on("resize",

    function() {

        var h = "innerHeight" in window ? window.innerHeight : document.documentElement.offsetHeight;
        $("#divDocViewer").height(h - 80);

        }
 );
```

Document Handler

For Asp.Net and Asp.Net MVC one important step in setting up the viewer is defining the handler in web.config. Please refer sample solution's web.config and copy the DocImage.axd handler to your web.config. Depending upon your IIS version you will either need System.Web or System.WebServer section. In most modern scenarios it is System.WebServer.

```
<!-- You need this handler for document viewer to work with your project, IIS below ver 7-->
<httpHandlers>
  <add verb="GET,POST" path="DocImage.axd" type="DotnetDaddy.DocumentViewer.DocImageHandler, DocumentViewer"/>
</httpHandlers>
```

```
  <!-- For IIS 7 or above.-->

<system.webServer>
  <handlers>
    <add name="DocImage" verb="GET,POST" path="DocImage.axd" type="DotnetDaddy.DocumentViewer.DocImageHandler, DocumentViewer"/>
  </handlers>
</system.webServer>
```

This setting differs when using WebFarm scenario. It is explained in the WebFarm section below. If you face any issue while deploying your website in production, then please set the BasePath property of the control. More details in the deployment section.

When using Cloud option the handler declaration will change. Following example shows how to use the Amazon S3 handler.

```
<system.webServer>
  <handlers>
    <add name="DocImage" verb="GET,POST" path="DocImage.axd" type="DotnetDaddy.DocumentCloud.AmazonS3ImageHandler, DocumentCloud"/>
  </handlers>
</system.webServer>
```

Following are the available cloud provider handlers:

- AmazonS3ImageHandler
- AzureStorageImageHandler
- GoogleCloudImageHandler
- RedisImageHandler
- DropBoxImageHandler
- FTPImageHandler
- CDNImageHandler

Listed below are the web.config settings for various providers. These are used as a fall-back if incase the settings are not available from the code. Consider this declaration as optional.

```
<add key="DoconutPngExportPath" value="~/Export" />

<add key="DoconutCache" value="false" />
<add key="DoconutMemoryCache" value="false" />
<add key="DoconutCacheTimeMinutes" value="60" />

<add key="DoconutSearch" value="false" />
<add key="DoconutAnnotation" value="false" />
<add key="DoconutLinks" value="false" />
<add key="DoconutWatermark" value="false" />
<add key="DoconutRotateFlip" value="false" />
<add key="DoconutHide" value="false" />

<add key="DoconutGoogleServiceAuthJsonPath" value="C:\path\google-service-auth.json" />
<add key="DoconutGoogleBucketName" value="my-bucket" />

<add key="DoconutDropBoxToken" value="9iOoxzPp4hwqXXXXXXXXXXXXXXXXD9zlsJI" />

<add key="DoconutAzureStorageConnectionString" value="XXXXXX-EndpointSuffix=core.windows.net" />
<add key="DoconutAzureContainerName" value="my-container" />

<add key="DoconutAwsS3Key" value="KEY-RKX3NZLT4NQOW" />
<add key="DoconutAwsS3Secret" value="SECRET-b9ULkNBvTU8tKg7KZiC5ijY" />
<add key="DoconutAwsS3BucketName" value="my-container" />
<add key="DoconutAwsS3RegionEndpoint" value="xxxxx" />

<add key="DoconutFtpHost" value="ftp.xxx.xxx" />
<add key="DoconutFtpSecure" value="false" />
<add key="DoconutFtpUser" value="xxxxx" />
<add key="DoconutFtpPassword" value="xxxx" />

<add key="DoconutCdnUrl" value="https://xxx.xxxx.net" />
```

You can add the cloud config to webserver cache or save it to disk. Please use the `Application_Start` event in Global.asax.

```
void Application_Start(object sender, EventArgs e)
{
    // We are saving the config instances to the webserver's cache
    // This is useful when using webfarm and you don't want to save
    // the credentials in web.config or to disk

    using (var objViewer = new DocViewer())
    {
        var cdnConfig = Utility.GetUploadConfig(CloudLocation.CDN);

         // use this only if SaveConfigToDisk = true
        cdnConfig.WebfarmPath = Server.MapPath("~/export");

        objViewer.SaveCloudConfig(cdnConfig);
    }
}
```

Client JavaScript API

The API calls like Zoom, Navigation etc. are called on the client side JavaScript instance of the Document Viewer. The object here is objDoc.

```
function RotateDocument(iAngle)
{
      objDoc.Rotate(objDoc.CurrentPage(), iAngle);
}
```

Bootstrap CSS

When using the viewer with bootstrap, you will need following class defined on your page. It will prevent the viewer's main view from shrinking.

```
.row {
    margin-right: 0;
    margin-left: 0
}
```

Web.Config

Following two settings are used for restrictinng document viewing by unwanted clients. Only the current user is able to view it.

Specify true for DoconutUnSafeMode and then anyone can view. In webfarm mode as session is not available, IP address is used to retrict it.

```
<add key="DoconutUnSafeMode" value="false" />
<add key="DoconutShowInfo" value="true" />
```

DoconutShowInfo flag is used to show doconut viewer information, when the DocImage.axd url is opened up in browser without any query string parameters.

WebFarm

The default mode for Document Viewer is using Asp.Net Cache, hence the single asp.net process memory. However you can use the viewer form a shared file system and have multiple worker processes. For more details please review samples related to WebFarm.

You will need an additional update in web.config. Specify the handler as DiskImage handler as shown below:

```
<httpHandlers>
  <add verb="GET,POST" path="DocImage.axd" type="DotnetDaddy.DocumentViewer.DiskImageHandler, DocumentViewer"/>
</httpHandlers>

<!-- For webfarms use DiskImageHandler instead of DocImageHandler -->
```

IIS 7 or above use below in System.webServer

```
<handlers>
  <add name="DocImage" verb="GET,POST" path="DocImage.axd" type="DotnetDaddy.DocumentViewer.DiskImageHandler, DocumentViewer" />
</handlers>
```

Next, you will need to define an app setting value for *DoconutPngExportPath* as shown below. This is used by the sample code in webfarm.aspx

```
<appSettings>
  <add key="DoconutPngExportPath" value="~/Export"/>
</appSettings>
```

You may also use a shared path instead of using a path inside the web directory. Apart from this there are two additional settings that govern the wait time for loading a page from disk.

```
<add key="DoconutPageWaitTimeSeconds" value="5"/>
<add key="DoconutStartWaitFromPage" value="5"/>
```

Page wait time seconds: The handler will wait for so many seconds while the page file is still not available.
Start wait from page: The above code will be valid from page number higher than the value mentioned here.

For .NET Core WebFarm set-up please refer .NET core section below.

Asp.Net MVC

MVC sample uses the same three DLLs and Doconut.lic license file as used by the Asp.Net project. All of the above sections for Asp.Net are applicable. However the way the viewer and its required scripts render differs for MVC.

Please refer the Simple MVC Viewer sample solution.

Controller

The index action of the home controller creates an instance of document viewer and sets all the necessary properties.

```
var viewer = new DocViewer
{
    ID = "ctlDoc",
```

It then stores the reference Scripts, CSS, and JavaScript initializer in ViewBag. This is later rendered and used in the razor view.

```
ViewBag.ViewerScripts = viewer.ReferenceScripts();
ViewBag.ViewerCSS = viewer.ReferenceCss();
ViewBag.ViewerID = viewer.ClientID;
ViewBag.ViewerObject = viewer.JsObject;
ViewBag.ViewerInit = viewer.GetAjaxInitArguments(token);
```

It then stores the reference Scripts, Css, and JavaScript initializer in ViewBag. This is later rendered and used in the razor view.

Reference

```
@section css {
    @* CSS for the Viewer *@

    @Html.Raw(ViewBag.ViewerCSS)


@section scripts {


    <!-- Viewer header scripts include-->
    @Html.Raw(ViewBag.ViewerScripts)
```

Init

```
$(window).on("load", function () {

    /* Init code for viewer */
    @Html.Raw(ViewBag.ViewerInit)

    token = '@ViewBag.token';

    Resize();
});
```

OpenFile

OpenFile action is called from the client side script. This creates an instance of "DocViewer" but here it is just to get back a valid token for the document being viewed. This token is then used from the view as follows.

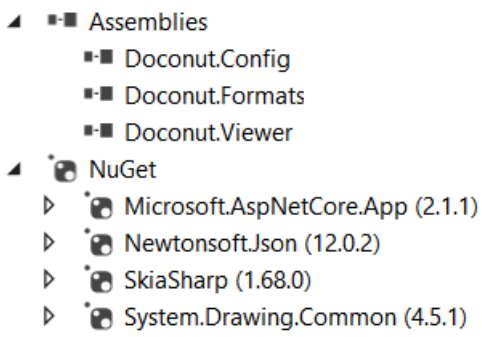
```
function OpenDocument(fileName) {

    loader.show();

    $.ajax({
        type: "POST",
        cache: false,
        url: "/Home/OpenFile?name=" + fileName,
        success: function (data) {
            token = data;          // store token for print
            objctlDoc.View(token); // use global object [objctlDoc] to view any document
        },
        error: function (textStatus, errorThrown, data) {
            alert("Unable to open document. Error: " + data.responseText);
            loader.hide();
        }
    });
}
```

Asp.Net Core

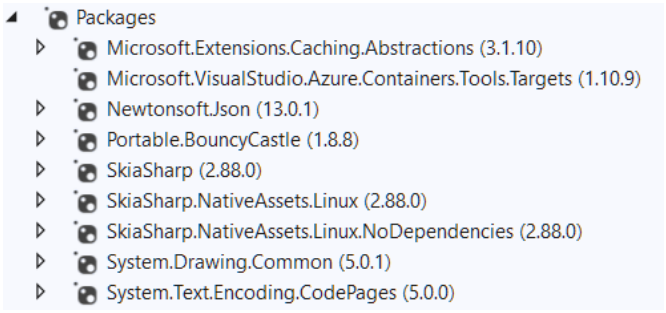
Asp.Net core project uses .Net Standard 2.0 compiled DLLs. Works with .Net Core 3.1, 5.0, Blazor, Razor pages. The .Net Core MVC sample is very similar to the Asp.Net MVC sample; hence few previous sections for MVC do apply here.

References And Nuget

Please reference the three DLLs provided in the .Net Core trial zip. You will need to add following two Nugets; Newtonsoft.Json and specific SkiaSharp version (2.80.3).



For Linux / Docker you need additional nugets as shown below:



Newtonsoft.Json, SkiaSharp (2.80.3),, SkiaSharp.NativeAssets.Linux.NoDependencies (2.80.3), System.Drawing.Common, System.Text.Encoding.CodePages.

Startup.cs

In the "ConfigureServices" section, define following:

```
services.AddHttpContextAccessor();

services.AddMemoryCache();

services.AddSingleton<IFileProvider>(
    new PhysicalFileProvider(
        Path.Combine(Directory.GetCurrentDirectory(), "wwwroot")));
```

AddHttpContextAccessor is required for licensing and AddMemoryCache for storing document objects.

In the "Configure" section, define the mapping for Doconut handler.

```
app.MapWhen(
    context => context.Request.Path.ToString().EndsWith("DocImage.axd"),
    appBranch =>
    {
          appBranch.UseDoconut(new DoconutOptions { ShowDoconutInfo = true, UnSafeMode = false});
    });
```

WebFarm

Enabling webfarm (multiple worker process) scenario in .NET Core is pretty easy. Just change the middleware used in startup.cs

```
app.MapWhen(
    context => context.Request.Path.ToString().EndsWith("DocImage.axd"),
    appBranch =>
    {
        appBranch.UseDoconutWebFarm(new WebFarmOptions
        {
            Path = Path.Combine(env.WebRootPath, webFarmFolder),
            PageWaitTimeSeconds = 5,
            StartWaitFromPage = 5,
            UnSafeMode = false
        }
        );

    });
```

Here the webfarm is a folder in *wwwroot*. However ideally it should be a central shared path that each load server should have read and write access.

Then add following code in OpenDocument action of HomeController.cs

```
// Webfarm
if (Startup.useWebfarm)
{
    docViewer.ExportToPng(cachePath);
}
```

Cloud

Just change the middleware used in startup.cs

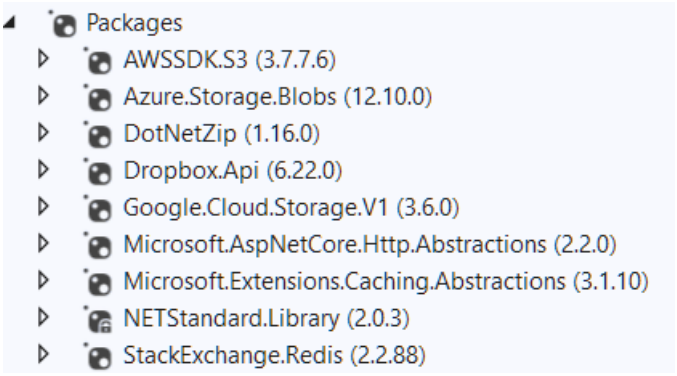
```
var cloudOptions = new Cloud.CloudOptions
{
    PngPath = Path.Combine(env.WebRootPath, webFarmFolder),
    Location = CloudLocation.AmazonS3,
    LogErrors = true,
    LogPath = Path.Combine(env.WebRootPath, "logs"),
};

cloudOptions.FunctionsConfig.CachePages = true;
cloudOptions.FunctionsConfig.CacheInMemory = true;

appBranch.UseDoconutCloud(cloudOptions);
```

For more details of the options used, please refer the Doconut Manual or refer the Cloud samples.

Then add following code in OpenDocument action of HomeController.cs. This will upload your current document to the selected cloud provider.

```
var azureConfig = new AzureConfig { DoconutAzureContainerName = "containe
docViewer.ExportToCloud(azureConfig);
```

Following are the Nuget packages required.



Blazor

Blazor Server apps use ASP.NET Core SignalR to communicate with the browser. Hence in order to use IhttpContextAccessor it is required that WebSockets is enable on the server. Please refer this article:

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/websockets?view=aspnetcore-3.1#enabling-websockets-on-iis

If you get any license error while deploying the blazor sample then please email support.

Deployment and IIS

The deployment is very simple. It is just a matter of publishing and copying the published code to your IIS website. However please ensure that Doconut.lic is deployed to bin directory or you may choose to embed the license information in code itself.

Also ensure that the Doconut handler is present in the web.config. For .Net Core version ensure that the middleware is added to application start.

While testing your live or test website never use Internet Explorer running from the server itself (windows server 2012, 2016, 2019 or higher). As IE runs in high security mode and will block many portions of the viewer. Instead always check from a client PC. Use latest version of IE/Edge or Chrome, Firefox.

If you receive an Init error while viewing then there are two possibilities 1) Handler is not defined in the web.config 2) Your DocImage.axd path is different. For this please set the BasePath property of the control to match your website's correct DocImage.axd url. Please refer a dedicated documentation for this inside Docs folder.

IIS and Application Pool

Your IIS application pool should be configured with single worker process. If you require a multiple worker / webfarm scenario then please refer the WebFarm samples and modify the DocImage handler in web.config accordingly.

Apart from these there are few other application pool settings that can be useful in production. Open IIS > Your Application Pool > Right Click > Open Advance Settings.

General > Queue Length	7,500 to 10,000
> Start Mode	AlwaysRunning
CPU > Limit Percent	70 to 90
> Limit Action	KillW3wp
Process Model > Idle Time-out	0
> Max worker process	0 (if higher, refer WebFarm)
Recycling > Private Memory Limit	Set as per your server capacity

Documentation

Please refer Doconut Manual/PDF inside the Documentation folder inside the trial zip. For any further assistance please email admin@doconut.com or visit https://doconut.com/Document-Viewer/Faq.aspx