DocFynd on Salesforce Classic

How to enable DocFynd on Salesforce Classic?

This article explains the steps to enable DocFynd on Salesforce Classic.

Step 1: Enable the Document list component

1. Open the Developer Console of your Salesforce org.
2. Click *File* on the top left and select *New* > *Visualforce Page*.
3. Enter "***docListVF***" and click OK.
4. Enter the code given below.

```
<apex:page showHeader="true" sidebar="true" lightningStyleSheets="true" standardController="Opportunity">
    <apex:includeLightning />
    <div id="LightningComponentid" />
<!-- the Id of div tag which will be used to render your LWC component  -->
    <script>
        if('{!$CurrentPage.parameters.isRelatedList}'){
            $Lightning.use("docfynd:docListApp", function() {
            $Lightning.createComponent("docfynd:documentList",
            {
                recordId : '{!$CurrentPage.parameters.id}',
                noOfRecToDisplay : '{!$CurrentPage.parameters.noOfRecToDisplay}',
                isVFPage : true,
                displayActions : '{!$CurrentPage.parameters.displayActions}',
                isRelatedList : '{!$CurrentPage.parameters.isRelatedList}',
                isDocListComp : '{!$CurrentPage.parameters.isDocListComp}'

            },
            "LightningComponentid", // the Id of div tag where your component will be rendered
            function(cmp) {});
        });
        } else {
            $Lightning.use("docfynd:docListApp", function() {
            $Lightning.createComponent("docfynd:documentList",
            {
                recordId : '{!$CurrentPage.parameters.id}',
                noOfRecToDisplay : 5,
                isVFPage : true,
                displayActions : true
            },
            "LightningComponentid", // the Id of div tag where your component will be rendered
            function(cmp) {});
        });
        }
    </script>
</apex:page>
```

5. Change the standard controller to the standard or custom objects for which you want the component to be displayed. By default, this is given for the "*Opportunity*" object in the code above.

6. Click "*Save*".

Step 2: Enable the Documents Upload component

1. Open the Developer Console of your Salesforce org.
2. Click *File* on the top left and select *New > Apex Class*.
3. Enter "*DocUploadVFController*" and click OK.
4. Enter the code given below.

```
public with sharing class DocUploadVFController {

    public Id recordId {get; set;}
    public String objApiName {get; set;}

    public DocUploadVFController(ApexPages.StandardController stdController){
        recordId = ApexPages.CurrentPage().getparameters().get('id');
        objApiName = recordId.getSObjectType().getDescribe().getName();
    }
}
```

5. Click "*Save*".
6. Next, click *File* on the top left and select *New > Visualforce Page*.
7. Enter "*docUploadVF*" and click OK.
8. Enter the code given below.

```
<apex:page showHeader="true" sidebar="true" lightningStyleSheets="true" standardController="Opportunity" extensi
    <apex:includeLightning />
    <div id="LightningComponentid" />
<!-- the Id of div tag which will be used to render your LWC component  -->
    <script>
        if('{!$CurrentPage.parameters.isRelatedList}'){
            $Lightning.use("docfynd:docUploadApp", function() {
            $Lightning.createComponent("docfynd:documentsUpload",
            {
                recordId : '{!recordId}',
                noOfRecToDisplay : '{!$CurrentPage.parameters.noOfRecToDisplay}',
                isVFPage : true,
                isRelatedList : '{!$CurrentPage.parameters.isRelatedList}',
                isDocListComp : '{!$CurrentPage.parameters.isDocListComp}',
                objectApiName : '{!objApiName}'


            },
            "LightningComponentid", // the Id of div tag where component will be rendered
            function(cmp) {});
        });
        } else {
            $Lightning.use("docfynd:docUploadApp", function() {
            $Lightning.createComponent("docfynd:documentsUpload",
            {
                recordId : '{!recordId}',
                isVFPage : true,
                objectApiName : '{!objApiName}'
            },
            "LightningComponentid", // the Id of div tag where your component will be rendered
            function(cmp) {});
        });
        }
```

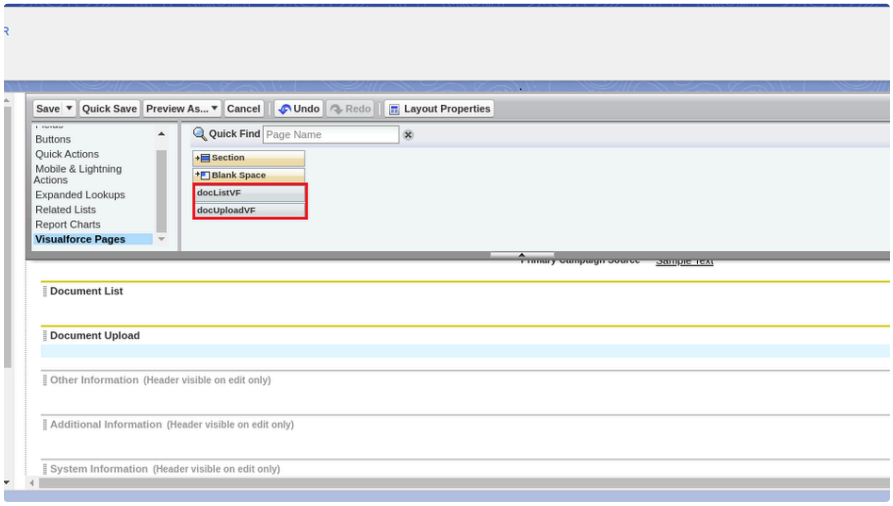
```
    </script>
</apex:page>
```

9. Change the standard controller to the standard or custom objects for which you want the component to be displayed. By default, this is given for the "*Opportunity*" object in the code above.

10. Click "*Save*".

Step 3: Add the created components to the page layout

1. From *Setup* screen, select "*Object Manager"* and search for the object for which you want the DocFynd components to be added.
2. Select the *Page Layout* tab on the left.
3. Click on the page layout to which you want to add the components.
4. Now, scroll down and select "*Visualforce Pages*" from the list.
5. The components that were created in the earlier steps namely "*docListVF*" and "*docUploadVF*" will display on the right side of the table.



DocFynd components

6. Now, drag and drop the section in the page layout where you want the component to be shown. Name it as "*Document List*". Select the "*1-Column*" radio button option. Click OK.
7. Add another section and name it as "*Document Upload"*. Select "*1-Column*" radio button option. Click OK.
8. Next, drag and drop the "*docListVF*" component inside the *Document List* section.
9. Click the properties icon from the top–right side of the component and change the *Height* to "550 px" and tick the "*Show scrollbars"* option. Click OK.

Page layout

10. Follow the same steps for the *"docUploadVF"* component as well.
11. After adding the components to the page layout click "*Save*".
12. Now you will find the DocFynd components visible in your page layout.

Ensure that your Visualforce page names and Apex classes are the same as mentioned in the code given.

😀 This completes the process and now DocFynd can be used in Salesforce Classic version as well.