



Enterprise Content Management apps in SharePoint 2013 and SharePoint Online solution pack (Module 1 of 8)

Microsoft Corporation

November 2014

**Applies to:** SharePoint 2013 and SharePoint Online

**Summary:** This solution pack includes code and documents that demonstrate and describe techniques that use enterprise content management features in SharePoint 2013 and SharePoint Online that can be delivered using apps.

©2014 Microsoft Corporation. All rights reserved.

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet website references, may change without notice. You bear the risk of using it.

Some examples are for illustration only and are fictitious. No real association is intended or inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. You may modify this document for your internal, reference purposes.

# Contents

The Enterprise Content Management apps in SharePoint 2013 and SharePoint Online solution pack contains eight modules, which are listed in Table 1.

**Table 1. Enterprise Content Management apps in SharePoint 2013 and SharePoint Online solution pack modules**

|  |  |  |
| --- | --- | --- |
| **Module** | **Name** | **Describes how to…** |
| **1** | **Document library templates** | **Implement a custom document library template when creating a document library. This sample describes how to use site columns, site content types, taxonomy fields, and version settings, and how to remove the default document content type from a document library.** |
| 2 | Document auto tagging | Automatically tag documents with metadata when documents are created or uploaded to SharePoint. This sample describes creation of taxonomy fields and content types, creation of document libraries with content types, registration of the ItemAdding and ItemAdded Remote Event Receiver, removal of Remote Event Receivers, retrieval of User Profile properties, and setting of taxonomy fields. |
| 3 | Information Management | Get or set site policies to manage the site lifecycle (closure and deletion of sites after a period of time). |
| 4 | Records management extensions | Enable and change in-place records management settings on your sites and lists. |
| 5 | Taxonomy operations | Create and read taxonomy data. |
| 6 | Bulk uploading documents | Bulk upload documents to document libraries (including OneDrive for Business). |
| 7 | Upload large files | Use different methods to upload large files to a document library. |
| 8 | Synchronize term groups | Synchronize term groups across multiple term stores. |

# [ECM.DocumentLibraries](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/ECM.DocumentLibraries)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this sample** | **How this app works** |
| This code sample shows how to create a list or document library, assign a content type to it, and remove the default content type using a provider hosted app. | Consider using this sample when you want to:   1. Create a list or document library with a default content type. 2. Maintain greater control over the addition, maintenance, or implementation of localized versions of your custom fields. 3. Remove the default content type on a list or library. 4. Apply library configuration settings when creating a list or library. | This code sample uses a provider-hosted app to create custom fields in the Managed Metadata Service, create lists and libraries with a custom content type, associate custom fields with content types, and remove the default content type from a document library. |

**Related samples**:

[ECM.Autotagging](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/ECM.AutoTagging)

# Understanding the code…

Users accessing the ECM.DocumentLibraries app must have permissions to manage lists. The **DoesUserHavePermission** method in **Default.aspx.cs** checks the user’s permissions to ensure they can manage lists. If the user does not have permissions to manage lists, this sample app presents an error message to the user, as shown in Figure 1.

private bool DoesUserHavePermission()

{

var spContext = SharePointContextProvider.Current.GetSharePointContext(Context);

using (var ctx = spContext.CreateUserClientContextForSPHost())

{

BasePermissions perms = new BasePermissions();

perms.Set(PermissionKind.ManageLists);

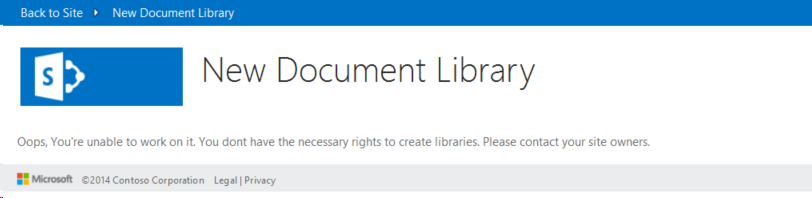
ClientResult<bool> \_permResult = ctx.Web.DoesUserHavePermissions(perms);

ctx.ExecuteQuery();

return \_permResult.Value;

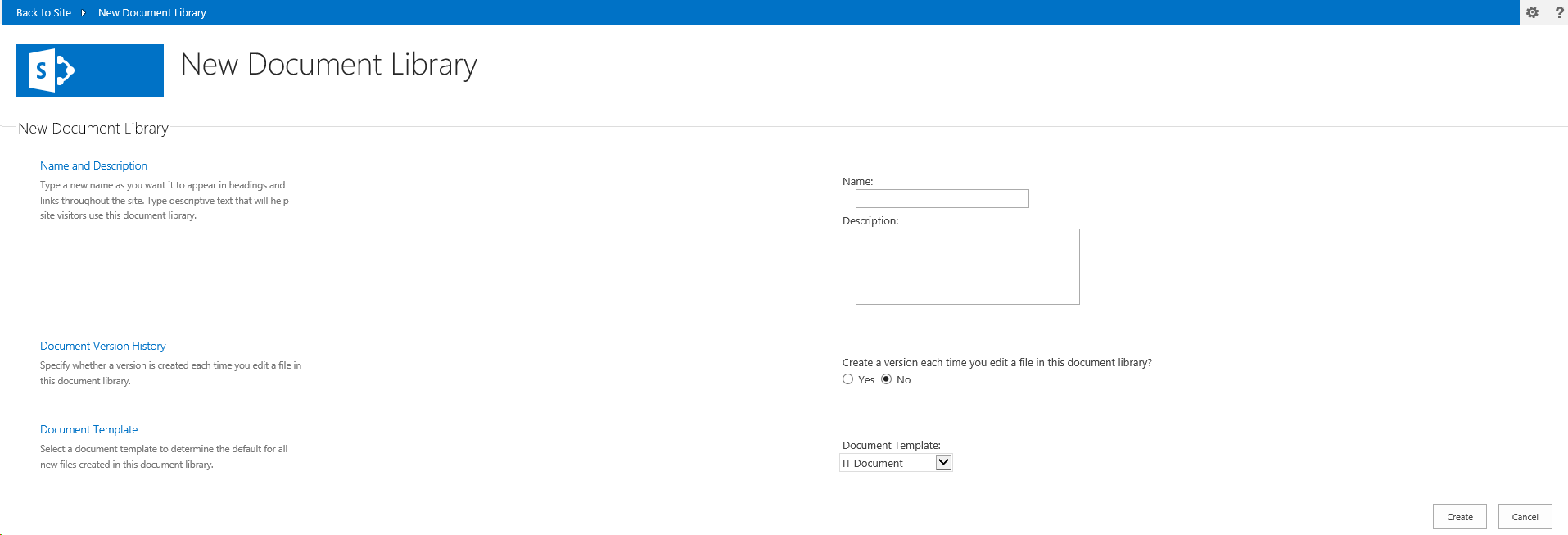
}

}

**Figure 1. Error message that displays when a user does not have manage list permissions**

When you start this app, the launch page displays as shown in Figure 2. The ECM.DocumentLibraries launch page looks exactly like the page to add a new document library when you select **Site Contents > add an app > Document Library > Advanced Options.** The main difference between these two apps are the values shown in the **Document Template** dropdown list. The Document Template dropdown list displays custom document library templates (IT Document and Contoso Document). When the user chooses **Create**, the selected custom content type is assigned to the new document library.

**Figure 2. Launch page of the ECM.DocumentLibraries app**



When users choose **Create**, the **CreateLibrary\_Click** method in **Default.aspx.cs** checks the selected Default Template and makes calls to either the **CreateITDocumentLibrary** or **CreateContosoDocumentLibrary** method in **ContentTypeManager.cs**.

protected void CreateLibrary\_Click(object sender, EventArgs e)

{

try

{

var \_spContext = SharePointContextProvider.Current.GetSharePointContext(Context);

var \_templateSelectedItem = this.DocumentTemplateType.Value;

var \_libraryToCreate = this.GetLibraryToCreate();

using (var \_ctx = \_spContext.CreateUserClientContextForSPHost())

{

\_ctx.ApplicationName = "AMS ECM.DocumentLibraries";

ContentTypeManager \_manager = new ContentTypeManager();

switch(\_templateSelectedItem)

{

case "IT Document":

\_manager.CreateITDocumentLibrary(\_ctx, \_libraryToCreate);

break;

case "Contoso Document":

\_manager.CreateContosoDocumentLibrary(\_ctx, \_libraryToCreate);

break;

}

}

Response.Redirect(this.Url.Value);

}

catch (Exception \_ex)

{

throw;

}

}

The code from the **CreateContosoDocumentLibrary** method that follows performs the following tasks:

1. Creates custom fields in the Managed Metadata Service.
2. Creates a content type.
3. Associates the custom fields with the content types.
4. Creates the document library with the content type.

public void CreateContosoDocumentLibrary(ClientContext ctx, Library library)

{

// Check the fields.

if (!ctx.Web.FieldExistsById(FLD\_CLASSIFICATION\_ID)){

ctx.Web.CreateTaxonomyField(FLD\_CLASSIFICATION\_ID,

FLD\_CLASSIFICATION\_INTERNAL\_NAME,

FLD\_CLASSIFICATION\_DISPLAY\_NAME,

FIELDS\_GROUP\_NAME,

TAXONOMY\_GROUP,

TAXONOMY\_TERMSET\_CLASSIFICATION\_NAME);

}

// Check the content type.

if (!ctx.Web.ContentTypeExistsById(CONTOSODOCUMENT\_CT\_ID)){

ctx.Web.CreateContentType(CONTOSODOCUMENT\_CT\_NAME,

CT\_DESC, CONTOSODOCUMENT\_CT\_ID,

CT\_GROUP);

}

// Associate fields to content types.

if (!ctx.Web.FieldExistsByNameInContentType(CONTOSODOCUMENT\_CT\_NAME, FLD\_CLASSIFICATION\_INTERNAL\_NAME)){

ctx.Web.AddFieldToContentTypeById(CONTOSODOCUMENT\_CT\_ID,

FLD\_CLASSIFICATION\_ID.ToString(),

false);

}

CreateLibrary(ctx, library, CONTOSODOCUMENT\_CT\_ID);

}

CreateContosoDocumentLibrary calls the **CreateTaxonomyField** method which is part of the **OfficeDevPnP.Core**. CreateTaxonomyField creates a field in the Managed Metadata Service from the provider hosted app.

public static Field CreateTaxonomyField(this Web web, Guid id, string internalName, string displayName, string group, TermSet termSet, bool multiValue = false)

{

internalName.ValidateNotNullOrEmpty("internalName");

displayName.ValidateNotNullOrEmpty("displayName");

termSet.ValidateNotNullOrEmpty("termSet");

try

{

var \_field = web.CreateField(id, internalName, multiValue ? "TaxonomyFieldTypeMulti" : "TaxonomyFieldType", true, displayName, group, "ShowField=\"Term1033\"");

WireUpTaxonomyField(web, \_field, termSet, multiValue);

\_field.Update();

web.Context.ExecuteQuery();

return \_field;

}

catch (Exception)

{

/// If there is an exception, the hidden field might be present.

FieldCollection \_fields = web.Fields;

web.Context.Load(\_fields, fc => fc.Include(f => f.Id, f => f.InternalName));

web.Context.ExecuteQuery();

var \_hiddenField = id.ToString().Replace("-", "");

var \_field = \_fields.FirstOrDefault(f => f.InternalName == \_hiddenField);

if (\_field != null)

{

\_field.DeleteObject();

web.Context.ExecuteQuery();

}

throw;

}

}

CreateContosoDocumentLibrary calls the **CreateContentType** method which is part of the **OfficeDevPnP.Core**. CreateContentType creates a new content type.

public static ContentType CreateContentType(this Web web, string name, string description, string id, string group, ContentType parentContentType = null)

{

LoggingUtility.Internal.TraceInformation((int)EventId.CreateContentType, CoreResources.FieldAndContentTypeExtensions\_CreateContentType01, name, id);

// Load the current collection of content types.

ContentTypeCollection contentTypes = web.ContentTypes;

web.Context.Load(contentTypes);

web.Context.ExecuteQuery();

ContentTypeCreationInformation newCt = new ContentTypeCreationInformation();

// Set the properties for the content type.

newCt.Name = name;

newCt.Id = id;

newCt.Description = description;

newCt.Group = group;

newCt.ParentContentType = parentContentType;

ContentType myContentType = contentTypes.Add(newCt);

web.Context.ExecuteQuery();

// Return the content type object.

return myContentType;

}

CreateContosoDocumentLibrary calls the **AddFieldToContentTypeById** method, which is part of the **OfficeDevPnP.Core**. AddFieldToContentTypeById associates a field with a content type.

public static void AddFieldToContentTypeById(this Web web, string contentTypeID, string fieldID, bool required = false, bool hidden = false)

{

// Get content type.

ContentType ct = web.GetContentTypeById(contentTypeID);

web.Context.Load(ct);

web.Context.Load(ct.FieldLinks);

web.Context.ExecuteQuery();

// Get field.

Field fld = web.Fields.GetById(new Guid(fieldID));

// Add field association to content type.

AddFieldToContentType(web, ct, fld, required, hidden);

}

CreateContosoDocumentLibrary calls the **CreateLibrary** method in **ContentTypeManager.cs** to create the document library. The **CreateLibrary** method assigns library settings such as the document library’s description, document versioning, and associated content types.

private void CreateLibrary(ClientContext ctx, Library library, string associateContentTypeID)

{

if (!ctx.Web.ListExists(library.Title))

{

ctx.Web.AddList(ListTemplateType.DocumentLibrary, library.Title, false);

List \_list = ctx.Web.GetListByTitle(library.Title);

if(!string.IsNullOrEmpty(library.Description)) {

\_list.Description = library.Description;

}

if(library.VerisioningEnabled) {

\_list.EnableVersioning = true;

}

\_list.ContentTypesEnabled = true;

\_list.Update();

ctx.Web.AddContentTypeToListById(library.Title, associateContentTypeID, true);

// Remove the default Document Content Type.

\_list.RemoveContentTypeByName(ContentTypeManager.DEFAULT\_DOCUMENT\_CT\_NAME);

ctx.Web.Context.ExecuteQuery();

}

else

{

throw new Exception("A list, survey, discussion board, or document library with the specified title already exists in this Web site. Please choose another title.");

}

}

CreateLibrary calls **RemoveContentTypeByName** in **ListExtensions.cs**, which is part of the **OfficeDevPnP.Core**. RemoveContentTypeByName removes the default content type on the document library.

public static void RemoveContentTypeByName(this List list, string contentTypeName)

{

if (string.IsNullOrEmpty(contentTypeName))

{

throw (contentTypeName == null)

? new ArgumentNullException("contentTypeName")

: new ArgumentException(CoreResources.Exception\_Message\_EmptyString\_Arg, "contentTypeName");

}

ContentTypeCollection \_cts = list.ContentTypes;

list.Context.Load(\_cts);

IEnumerable<ContentType> \_results = list.Context.LoadQuery<ContentType>(\_cts.Where(item => item.Name == contentTypeName));

list.Context.ExecuteQuery();

ContentType \_ct = \_results.FirstOrDefault();

if (\_ct != null)

{

\_ct.DeleteObject();

list.Update();

list.Context.ExecuteQuery();

}

}

After creating the document library, navigate to the **Library settings** on your document library to review the name, description, document versioning setting, content type, and custom fields you assigned to your document library.

**Figure 3. Library settings applied by this sample app**



**Figure 4. Content Type and custom field applied to this document library**

