# Office AMS: Document library TEmplates PROVIDER HOSTED APP

|  |  |
| --- | --- |
| Summary: | Applies to: |
| This sample demonstrates how to implement Document Library Templates in a provider hosted application. | * Office 365 Multi-Tenant (MT) * Office 365 Dedicated (D) * SharePoint 2013 on-premises |
| Solution: | ECM.DocumentLibraries, version 1.0 |
| Author: | Brian Michely, Vesa Juvonen, Bert Jansen, Frank Marasco (Microsoft) |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // THIS CODE IS PROVIDED \*AS IS\* WITHOUT WARRANTY OF  // ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY  // IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR  // PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

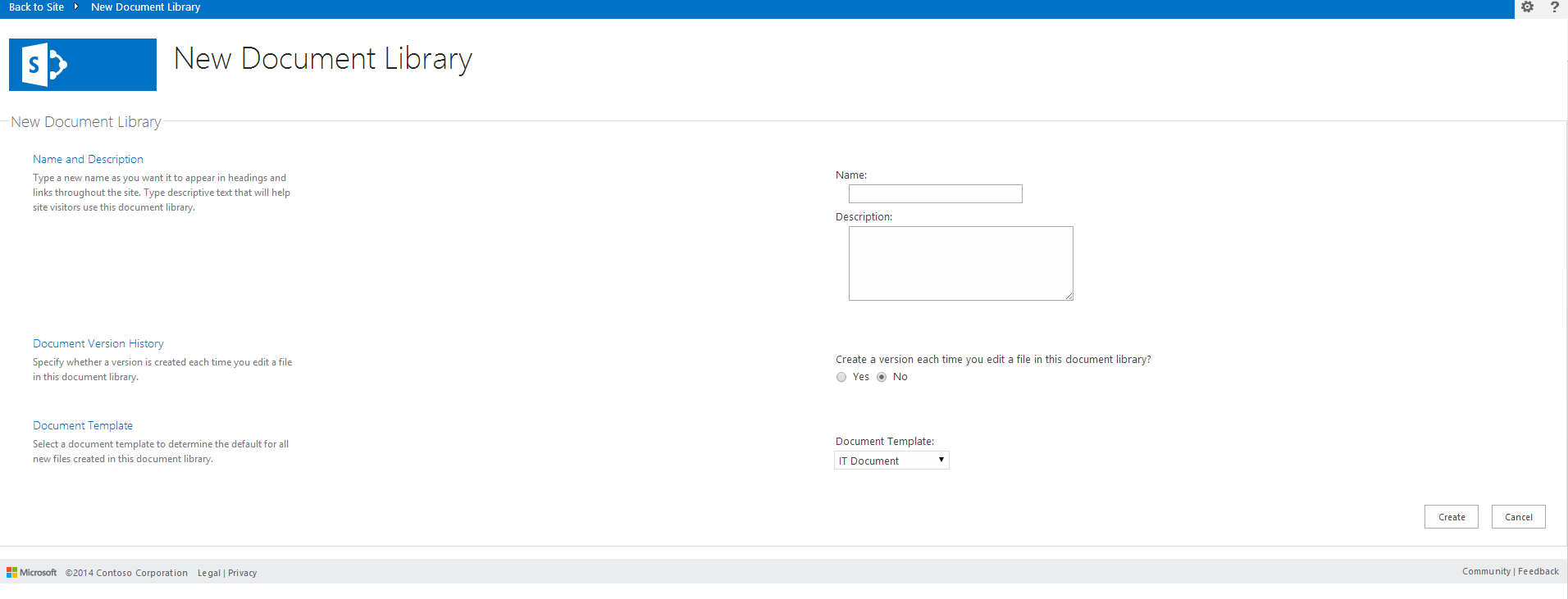
## Version log

Change log between different releases.

|  |  |
| --- | --- |
| Version | Notes |
| 1.0 | Initial |

# General comments

This sample scenario shows how one can implement Document Library templates using a provider hosted application instead of the feature framework or sandbox solution. This sample demonstrates creating Site columns, Site Content Types, Creation of Taxonomy fields, the removal of the Default Document Content Type from the created library, and versioning settings of the library. To make this application available to all site collections, you can deploy the app a tenant scope application, which would make this application available in all site collections.



# SCENARIO: Creating of SITE COLUMNS

private const string DEFAULT\_DOCUMENT\_CT\_NAME = "Document";

/// COMMON

private const string CT\_GROUP = "Contoso Content Types";

private const string CT\_DESC = "Create a new Document";

private const string FIELDS\_GROUP\_NAME = "Contoso Columns";

/// CONTENT TYPE IT DOCUMENT

private const string ITDOCUMENT\_CT\_ID = "0x01005D4F34E4BE7F4B6892AEBE088EDD215E";

private const string ITDOCUMENT\_CT\_NAME = "IT Document";

/// CONTENT TYPE CONTOSO DOCUMENT

private const string CONTOSODOCUMENT\_CT\_ID = "0x0100A112247905884D0DA49735433433A93C";

private const string CONTOSODOCUMENT\_CT\_NAME = "Contoso Document";

//FIELD BUSINESS UNIT

private readonly Guid FLD\_BUSINESS\_UNIT\_ID = new Guid("91AE1803-2F95-427F-97DB-5CE1652C07B0");

private const string FLD\_BUSINESS\_UNIT\_INTERNAL\_NAME = "\_BusinessUnit";

private const string FLD\_BUSINESS\_UNIT\_DISPLAY\_NAME = "Business Unit";

//FIELD CLASSIFICATION

private readonly Guid FLD\_CLASSIFICATION\_ID = new Guid("D7A785FC-7974-4CBD-864C-AE0012E97A22");

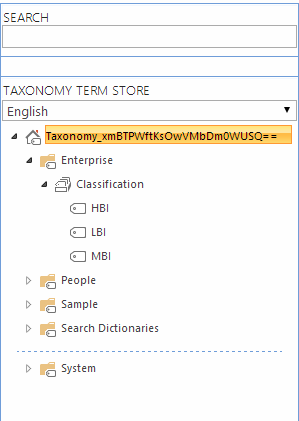
private const string FLD\_CLASSIFICATION\_INTERNAL\_NAME = "\_classification";

private const string FLD\_CLASSIFICATION\_DISPLAY\_NAME = "Classification";

private const string TAXONOMY\_GROUP = "Enterprise";

private const string TAXONOMY\_TERMSET\_CLASSIFICATION\_NAME = "Classification";

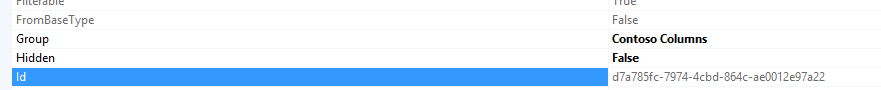
Since this assumes that the Term Store is available and a Term Group named “Enterprise” is created as well as a Term set named “Classification” is already created as depicted below.



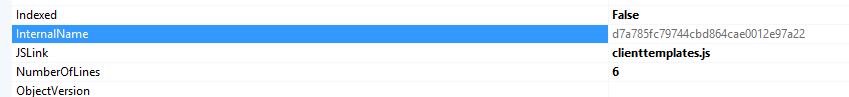
Note: See AMS sample core.mms and or core.mmsSync which demonstrates how to perform taxonomy related operations.

Now when working with Taxonomy fields from a user point of view, it looks like there is only one field type that creates only a single taxonomy field in SharePoint; but there is actual two fields. With that said, if you go and delete the Taxonomy Column, the hidden field still exists and you will review an exception when trying to recreate the taxonomy field. This hidden field will have an internal name of the guid assigned to the Taxonomy Field that you are creating.

### Taxonomy Field Column



### Taxonomy Hidden Field



In order to handle this situation and prevent an error doing your testing we should delete the hidden Taxonomy field if an exception occurs.

public static Field CreateTaxonomyField(this Web web, Guid id, string internalName, string displayName, string group, string mmsGroupName, string mmsTermSetName)

{

try

{

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

web.WireUpTaxonomyField(id, mmsGroupName, mmsTermSetName);

\_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;

}

}

We use the OfficeAMS extensions to create the content columns in the host site collection.

//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);

}

# CREATION OF THE CONTENT TYPE

We use the OfficeAMS extensions to create the content type in the host site collection.

//check the content type

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

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

CT\_DESC,

CONTOSODOCUMENT\_CT\_ID,

CT\_GROUP);

}

# CREATION OF THE LIBRARIES

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);

//we are going to remove the default Document Content Type

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

ctx.Web.Context.ExecuteQuery();

}

Notice the RemoveContentType member. This is an extension that will remove the default Document content type in the list.

/// <summary>

/// Removes a content type from a list/library

/// </summary>

/// <param name="list">The list</param>

/// <param name="contentTypeName">The content type name to remove from the list</param>

public static void RemoveContentType(this List list , string 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();

}

}

# USER VALIDATION

Now, if a user tries to access the provider hosted application, the sample solution will validate if the user has manage list permission.

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

using (var ctx = spContext.CreateUserClientContextForSPHost())

{

BasePermissions perms = new BasePermissions();

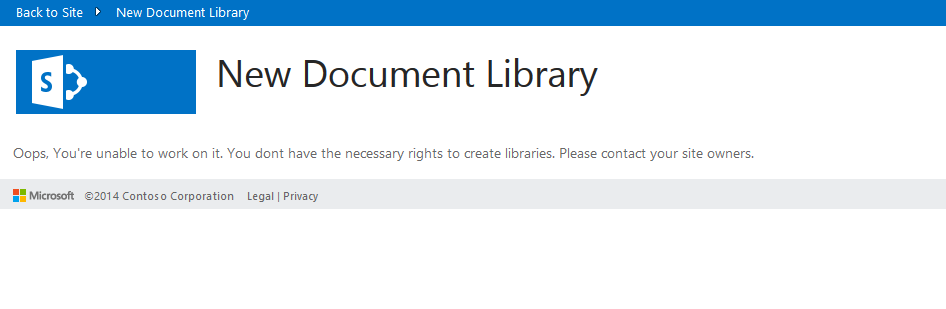
perms.Set(PermissionKind.ManageLists);

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

ctx.ExecuteQuery();

return \_permResult.Value;

}



# Dependencies

* Microsoft.SharePoint.Client.dll
* Microsoft.SharePoint.Client.Runtime.dll
* Microsoft.SharePoint.Client.Taxonomy
* OfficeAMS.Core