



Enterprise Content Management apps in SharePoint 2013 and SharePoint Online solution pack (Module 8 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.** |

# [Core.MMSSync app](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.MMSSync)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this** | **How the app works** |
| This sample app for SharePoint demonstrates how to synchronize a source and target taxonomy. | Consider using this sample when you want to:   * Synchronize two taxonomies. For example, you might use both SharePoint Online and SharePoint Server 2013 on-premises for different sets of data but they must use the same taxonomy. * Synchronize changes made to a specific term group only. | This app synchronizes two term stores in the Managed Metadata service. One term store is the source term store, and the other is the target term store. Term groups are synchronized from the source to the target term store using either:   * The TermStore object * The ChangeInformation object   Note: These objects belong to the Microsoft.SharePoint.Client.Taxonomy assembly. |

**Related samples**:

[OfficeDevPnP.Core](https://github.com/OfficeDev/PnP/tree/dev/OfficeDevPnP.Core)

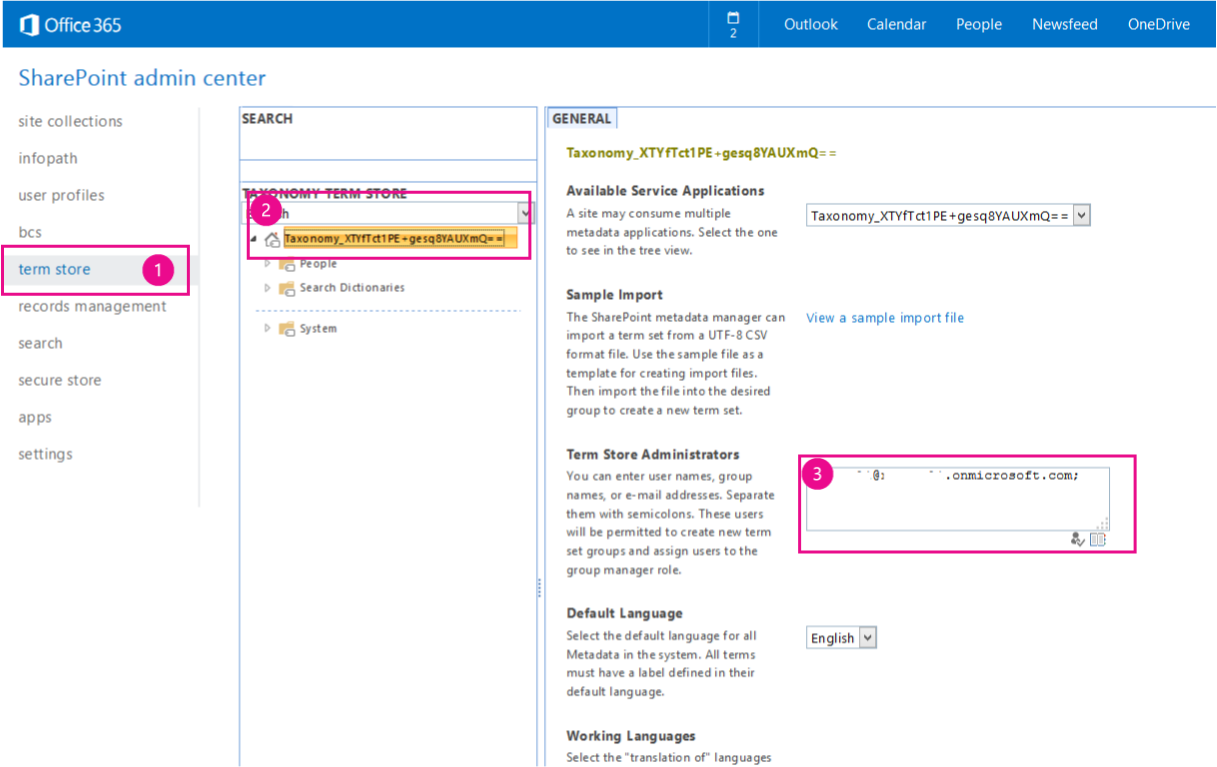
[Core.MMS](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.MMS)

# Before you begin…

Ensure that you have performed the following steps before running this app:

1. Assign Term Store permissions to your user who is running this sample. Figure 1 shows the Office 365 Admin Center where your Office 365 administrator can assign these permissions.

**Figure 1. Assign user permissions to the Term Store in the Office 365 Admin Center.**

****

# Understanding the code…

When you start this sample, a console application displays, as shown in Figure 2. You are prompted to enter the following information:

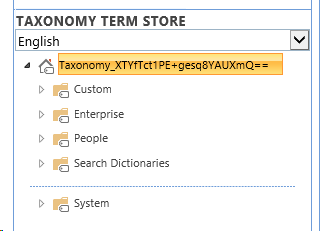
* Source Managed Metadata Service information
  + The URL of your Office 365 Admin Center that contains the source term store (this is the URL of the source MMS). For example, you might enter <https://contososource-admin.sharepoint.com>.
  + The credentials of a user that is a Term Store Administrator on your source Managed Metadata Service.
* Target Managed Metadata Service information
  + The URL of your Office 365 Admin Center that contains the target term store (this is the URL of the target MMS). For example, you might enter <https://contosotarget-admin.sharepoint.com>).
  + The credentials of a user that is a Term Store Administrator on your target Managed Metadata Service.
* The type of operation you want to perform. You can perform the following operations:
  + Move a term group (scenario 1) using the TermStore object.
  + Process changes (scenario 2) using the ChangeInformation object.
* After selecting scenario 1 or scenario 2, you must enter the name of the Term Group you want to synchronize from your source to your target Managed Metadata Service. For example, you might enter **Enterprise**, as shown in Figure 3.

**Important:** This code sample can work with SharePoint Online and SharePoint Server 2013 on-premises.

**Figure 2. Core.MMSSync console application**



**Figure 3. Example of Term Groups in the Managed Metadata Service**



# Scenario 1 - Move Term Group

When the user selects Move Term Group, the app prompts the user to enter a Term Group to synchronize and then calls the **CopyNewTermGroups** method in **MMSSyncManager.cs**. **CopyNewTermGroups** copies a term group from the source term store to the target term store. **CopyNewTermGroups** performs the following tasks:

1. Retrieves the source and target Term Store objects.
2. Verifies that the languages of the source and target term stores match.
3. Verifies that the source term group doesn’t exist in the target term store, and then copies the source term group to the target term store using **CreateNewTargetTermGroup**.

**Note:** You can set the TermGroupExclusions, TermGroupToCopy, and TermSetInclusions parameters to filter which terms get processed.

The following code shows the **CopyNewTermGroups** and **CreateNewTargetTermGroup** methods from **MMSSyncManager.cs**.

public bool CopyNewTermGroups(ClientContext sourceContext, ClientContext targetContext, List<string> termGroupExclusions = null, string termGroupToCopy = null)

{

TermStore sourceTermStore = GetTermStoreObject(sourceContext);

TermStore targetTermStore = GetTermStoreObject(targetContext);

List<int> languagesToProcess = null;

if (!ValidTermStoreLanguages(sourceTermStore, targetTermStore, out languagesToProcess))

{

Log.Internal.TraceError((int)EventId.LanguageMismatch, "The target termstore default language is not available as language in the source term store, syncing cannot proceed.");

return false;

}

// Get a list of term groups to process, exclude site collection scoped groups and system groups.

IEnumerable<TermGroup> termGroups = sourceContext.LoadQuery(sourceTermStore.Groups.Include(g => g.Name,

g => g.Id,

g => g.IsSiteCollectionGroup,

g => g.IsSystemGroup))

.Where(g => g.IsSystemGroup == false && g.IsSiteCollectionGroup == false);

sourceContext.ExecuteQuery();

foreach (TermGroup termGroup in termGroups)

{

// Skip term group if you only want to copy one particular term group.

if (!String.IsNullOrEmpty(termGroupToCopy))

{

if (!termGroup.Name.Equals(termGroupToCopy, StringComparison.InvariantCultureIgnoreCase))

{

continue;

}

}

// Skip term groups that you do not want to copy.

if (termGroupExclusions != null && termGroupExclusions.Contains(termGroup.Name, StringComparer.InvariantCultureIgnoreCase))

{

Log.Internal.TraceInformation((int)EventId.CopyTermGroup\_Skip, "Skipping {0} as this is a system termgroup", termGroup.Name);

continue;

}

// About to start copying a term group.

TermGroup sourceTermGroup = GetTermGroup(sourceContext, sourceTermStore, termGroup.Name);

TermGroup targetTermGroup = GetTermGroup(targetContext, targetTermStore, termGroup.Name);

if (sourceTermGroup == null)

{

continue;

}

if (targetTermGroup != null)

{

if (sourceTermGroup.Id != targetTermGroup.Id)

{

// Term group exists with a different ID...can't sync

Log.Internal.TraceWarning((int)EventId.CopyTermGroup\_IDMismatch, "The term groups have different ID's. I don't know how to work it.");

}

else

{

// Do nothing as this term group was previously copied. Term group changes need to be

// picked up by the change log processing.

Log.Internal.TraceInformation((int)EventId.CopyTermGroup\_AlreadyCopied, "Termgroup {0} was already copied...changes to it will need to come from changelog processing.", termGroup.Name);

}

}

else

{

Log.Internal.TraceInformation((int)EventId.CopyTermGroup\_Copying, "Copying termgroup {0}...", termGroup.Name);

this.CreateNewTargetTermGroup(sourceContext, targetContext, sourceTermGroup, targetTermStore, languagesToProcess);

}

}

return true;

}

private void CreateNewTargetTermGroup(ClientContext sourceClientContext, ClientContext targetClientContext, TermGroup sourceTermGroup, TermStore targetTermStore, List<int> languagesToProcess)

{

TermGroup destinationTermGroup = targetTermStore.CreateGroup(sourceTermGroup.Name, sourceTermGroup.Id);

if (!string.IsNullOrEmpty(sourceTermGroup.Description))

{

destinationTermGroup.Description = sourceTermGroup.Description;

}

TermSetCollection sourceTermSetCollection = sourceTermGroup.TermSets;

if (sourceTermSetCollection.Count > 0)

{

foreach (TermSet sourceTermSet in sourceTermSetCollection)

{

sourceClientContext.Load(sourceTermSet,

set => set.Name,

set => set.Description,

set => set.Id,

set => set.Contact,

set => set.CustomProperties,

set => set.IsAvailableForTagging,

set => set.IsOpenForTermCreation,

set => set.CustomProperties,

set => set.Terms.Include(

term => term.Name,

term => term.Description,

term => term.Id,

term => term.IsAvailableForTagging,

term => term.LocalCustomProperties,

term => term.CustomProperties,

term => term.IsDeprecated,

term => term.Labels.Include(label => label.Value, label => label.Language, label => label.IsDefaultForLanguage)));

sourceClientContext.ExecuteQuery();

TermSet targetTermSet = destinationTermGroup.CreateTermSet(sourceTermSet.Name, sourceTermSet.Id, targetTermStore.DefaultLanguage);

targetClientContext.Load(targetTermSet, set => set.CustomProperties);

targetClientContext.ExecuteQuery();

UpdateTermSet(sourceClientContext, targetClientContext, sourceTermSet, targetTermSet);

foreach (Term sourceTerm in sourceTermSet.Terms)

{

Term reusedTerm = targetTermStore.GetTerm(sourceTerm.Id);

targetClientContext.Load(reusedTerm);

targetClientContext.ExecuteQuery();

Term targetTerm;

if (reusedTerm.ServerObjectIsNull.Value)

{

try

{

targetTerm = targetTermSet.CreateTerm(sourceTerm.Name, targetTermStore.DefaultLanguage, sourceTerm.Id);

targetClientContext.Load(targetTerm, term => term.IsDeprecated,

term => term.CustomProperties,

term => term.LocalCustomProperties);

targetClientContext.ExecuteQuery();

UpdateTerm(sourceClientContext, targetClientContext, sourceTerm, targetTerm, languagesToProcess);

}

catch (ServerException ex)

{

if (ex.Message.IndexOf("Failed to read from or write to database. Refresh and try again.") > -1)

{

// This exception was due to caching issues and generally is thrown when there's term reuse across groups

targetTerm = targetTermSet.ReuseTerm(reusedTerm, false);

}

else

{

throw ex;

}

}

}

else

{

targetTerm = targetTermSet.ReuseTerm(reusedTerm, false);

}

targetClientContext.Load(targetTerm);

targetClientContext.ExecuteQuery();

targetTermStore.UpdateCache();

// Refresh session and term store references to force reload of the term just added. This is

//needed because there can be an update change event following next and without this trick

// the newly created term set cannot be obtained from the server.

targetTermStore = GetTermStoreObject(targetClientContext);

// Recursively add the other terms.

ProcessSubTerms(sourceClientContext, targetClientContext, targetTermSet, targetTerm, sourceTerm, languagesToProcess, targetTermStore.DefaultLanguage);

}

}

}

targetClientContext.ExecuteQuery();

}

# Scenario 2 – Process Changes

When the user selects Process Changes, the app prompts the user to enter a Term Group to synchronize, and then calls the **ProcessChanges** method in **MMSSyncManager.cs**. **ProcessChanges** uses the **GetChanges** method of the ChangedInformation class to retrieve all changes made to groups, term sets and terms in the source Managed Metadata Service. Changes are then applied to the target Managed Metadata Service.

Note: This document includes only some portions of the **ProcessChanges** method. To review the entire **ProcessChanges** method, open the Core.MMSSync solution using Visual Studio 2013.

The **ProcessChanges** method starts by creating a **TaxonomySession** object.

Log.Internal.TraceInformation((int)EventId.TaxonomySession\_Open, "Opening the taxonomy session");

TaxonomySession sourceTaxonomySession = TaxonomySession.GetTaxonomySession(sourceClientContext);

TermStore sourceTermStore = sourceTaxonomySession.GetDefaultKeywordsTermStore();

sourceClientContext.Load(sourceTermStore,

store => store.Name,

store => store.DefaultLanguage,

store => store.Languages,

store => store.Groups.Include(group => group.Name, group => group.Id));

sourceClientContext.ExecuteQuery();

Next, we retrieve changes using the ChangeInformation object, and setting the start date on the ChangeInformation object. In this sample, we retrieve all changes that were made within the last year.

Log.Internal.TraceInformation((int)EventId.TermStore\_GetChangeLog, "Reading the changes");

ChangeInformation changeInformation = new ChangeInformation(sourceClientContext);

changeInformation.StartTime = startFrom;

ChangedItemCollection termStoreChanges = sourceTermStore.GetChanges(changeInformation);

sourceClientContext.Load(termStoreChanges);

sourceClientContext.ExecuteQuery();

The **GetChanges** method returns a **ChangedItemCollection**, which we use to enumerate all changes occurring in the term store as shown in the following code segment. The last line of the following code segment checks whether the ChangedItem was a Term Group. **ProcessChanges** includes code segments to perform similar checks on the **ChangedItem** for term sets and terms.

foreach (ChangedItem \_changeItem in termStoreChanges)

{

if (\_changeItem.ChangedTime < startFrom)

{

Log.Internal.TraceVerbose((int)EventId.TermStore\_SkipChangeLogEntry, "Skipping item {1} changed at {0}", \_changeItem.ChangedTime, \_changeItem.Id);

continue;

}

Log.Internal.TraceVerbose((int)EventId.TermStore\_ProcessChangeLogEntry, "Processing item {1} changed at {0}. Operation = {2}, ItemType = {3}", \_changeItem.ChangedTime, \_changeItem.Id, \_changeItem.Operation, \_changeItem.ItemType);

#region Group changes

if (\_changeItem.ItemType == ChangedItemType.Group)

The changed item type might be a term group, term set, or term. Each changed item type has different operations you can perform on it. The following table lists the operations that you can perform on each changed item type.

|  |  |
| --- | --- |
| **What changed? (ChangedItemType)** | **Operations you can perform on the changed item type (ChangedOperationType)** |
| Group | Delete Group  Add Group  Edit Group |
| TermSet | Delete term set  Move term set  Copy term set  Add term set  Edit term set |
| Term | Delete term  Move term  Copy term  Path change term  Merge term  Add term  Edit term |

The following code segment shows how to perform a delete operation when a term group was deleted in the source Managed Metadata Service.

#region Delete group

if (\_changeItem.Operation == ChangedOperationType.DeleteObject)

{

TermGroup targetTermGroup = targetTermStore.GetGroup(\_changeItem.Id);

targetClientContext.Load(targetTermGroup, group => group.Name);

targetClientContext.ExecuteQuery();

if (!targetTermGroup.ServerObjectIsNull.Value)

{

if (termGroupExclusions == null || !termGroupExclusions.Contains(targetTermGroup.Name, StringComparer.InvariantCultureIgnoreCase))

{

Log.Internal.TraceInformation((int)EventId.TermGroup\_Delete, "Deleting group: {0}", targetTermGroup.Name);

targetTermGroup.DeleteObject();

targetClientContext.ExecuteQuery();

}

}

}

#endregion