



Enterprise Content Management apps in SharePoint 2013 and SharePoint Online solution pack (Module 6 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.BulkDocumentUploader](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.BulkDocumentUploader)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this sample** | **How this app works** |
| This code sample shows how to upload documents to SharePoint document libraries (including OneDrive for Business) using REST APIs.  **Important:** This code sample demonstrates how to upload one file to a document library. You need to extend this sample to upload multiple files. | Consider using this sample when:   * You want to upload files to SharePoint Online. * You want to migrate to Office 365 and move your files using your own custom migration tool.   **Note:** This sample can work for any document library in SharePoint, including the OneDrive for Business document library. | This code sample uses a console application to upload files to document libraries using REST API calls. This sample uploads a company policy document to a user’s OneDrive document library. Configuration settings are specified in a XML file and a CSV file.  **Note:** This sample does not use OfficeDevPnP.Core. |

**Related samples**:

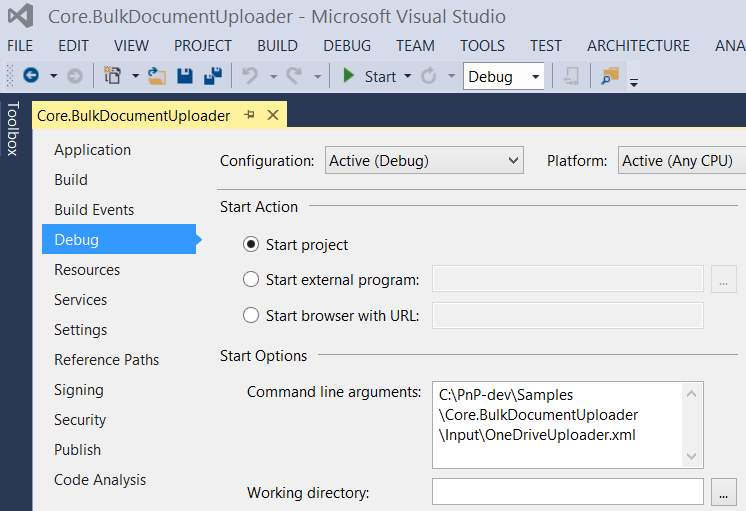
[Core.LargeFileUpload](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.LargeFileUpload)

# Before you begin…

Perform the following steps before you run this code sample:

1. Edit the **OneDriveUploader.xml** file with the following information:
   * The location where you want to save your text and .csv log files.
   * The file path to your CSV mapping file (for example, C:\PnP\Samples\Core.BulkDocumentUploader\Input\SharePointSites.csv).
   * The location of the company policy files to upload (for example, C:\PnP\Samples\Core.BulkDocumentUploader\Input\OneDriveFiles).
   * Your SharePoint Online credentials.
   * The document action to perform (either upload or delete).
   * The new filename to apply to the file after the file has been uploaded to the document library (for example, **COMPANY POLICY DOCUMENT.xlsx**).
2. In the **SharePointSites.csv** mapping file, list the document library URL to upload files to, and the name of the company policy file to upload.
3. Add the file path of the OneDriveUploader.xml file as a command line argument by opening the project properties on the **Core.BulkDocumentUploader** projectin Solution Explorer, and then selecting **Properties > Debug** as shown in Figure 1.

**Figure 1. Set OneDriveUploader.xml as a command line argument.**



# Understanding the code…

From the **Main** method in **Program.cs**, the **RecurseActions** method calls the **Run** method in **OneDriveMapper.cs.** The **Run** methodgets the location of the file to upload from **SharePointSites.csv**,and then calls the **IterateCollection** method.

public override void Run(BaseAction parentAction, DateTime CurrentTime, LogHelper logger)

{

CsvProcessor csvProcessor = new CsvProcessor();

logger.LogVerbose(string.Format("Attempting to read mapping CSV file '{0}'", this.UserMappingCSVFile));

using (StreamReader reader = new StreamReader(this.UserMappingCSVFile))

{

csvProcessor.Execute(reader, (entries, y) => { IterateCollection(entries, logger); }, logger);

}

}

The SharePointSite.csv file lists a file to upload and the document library to upload that file to. The **IterateCollection** method uploads the file to the document library by performing the following tasks:

1. Gets the file to upload.
2. Ensures the user has permissions to add items.
3. Creates the HttpWebRequest object with the authentication cookie, the REST string request to upload the document, and the HTTP request action method.
4. Performs the file upload (or deletion).

**Note:** The filename is overwritten with the value of FileUploadName specified in OneDriveUploader.xml.

public override void IterateCollection(Collection<string> entries, LogHelper logger)

{

Stopwatch IterationSW = new Stopwatch();

IterationSW.Start();

logger.LogVerbose(string.Format(CultureInfo.CurrentCulture, "Establishing context object to: '{0}'", entries[this.SiteIndex]));

try

{

// Use context of current iteration URL for current user item.

using (ClientContext context = new ClientContext(entries[this.SiteIndex]))

{

using (SecureString password = new SecureString())

{

foreach (char c in this.Password.ToCharArray())

{

password.AppendChar(c);

}

context.Credentials = new SharePointOnlineCredentials(this.UserName, password);

// Get file to upload from directory.

FileInfo theFileToUpload = new FileInfo(Path.Combine(this.DirectoryLocation + "\\", entries[this.FileIndex] + ".xlsx"));

logger.LogVerbose(string.Format(CultureInfo.CurrentCulture, "Attempting to {0} file {1}", this.DocumentAction, theFileToUpload));

// Ensure account has permissions to access.

BasePermissions perm = new BasePermissions();

perm.Set(PermissionKind.AddListItems);

ConditionalScope scope = new ConditionalScope(context, () => context.Web.DoesUserHavePermissions(perm).Value);

using(scope.StartScope())

{

Stopwatch tempSW = new Stopwatch();

tempSW.Start();

int success = 0;

while(tempSW.Elapsed.TotalSeconds < 20)

{

var digest = context.GetFormDigestDirect();

string cookie = ((SharePointOnlineCredentials)context.Credentials).GetAuthenticationCookie(new Uri(entries[this.SiteIndex])).TrimStart("SPOIDCRL=".ToCharArray());

using (Stream s = theFileToUpload.OpenRead())

{

// Define REST string request to upload document to context. This string specifies the Documents folder, but you can specify another document library.

string theTargetUri = string.Format(CultureInfo.CurrentCulture, "{0}/\_api/web/lists/getByTitle('Documents')/RootFolder/Files/add(url='{1}',overwrite='true')?", entries[this.SiteIndex], this.FileUploadName);

// Define REST HTTP request object.

HttpWebRequest SPORequest = (HttpWebRequest)HttpWebRequest.Create(theTargetUri);

// Define HTTP request action method.

if (this.DocumentAction == "Upload")

{

SPORequest.Method = "POST";

}

else if (this.DocumentAction == "Delete")

{

SPORequest.Method = "DELETE";

}

else

{

logger.LogVerbose(string.Format(CultureInfo.CurrentCulture, "There was a problem with the HTTP request in DocumentAction attribute of XML file"));

throw new Exception("The HTTP Request operation is not supported, please check the value of DocumentAction in the XML file");

}

// Build out additional HTTP request details.

SPORequest.Accept = "application/json;odata=verbose";

SPORequest.Headers.Add("X-RequestDigest", digest.DigestValue);

SPORequest.ContentLength = s.Length;

SPORequest.ContentType = "application/octet-stream";

// Handle authentication to context through cookie.

SPORequest.CookieContainer = new CookieContainer();

SPORequest.CookieContainer.Add(new Cookie("SPOIDCRL", cookie, string.Empty, new Uri(entries[this.SiteIndex]).Authority));

// Perform file upload/deletion.

using (Stream requestStream = SPORequest.GetRequestStream())

{

s.CopyTo(requestStream);

}

// Get HTTP response to determine success of operation.

HttpWebResponse SPOResponse = (HttpWebResponse)SPORequest.GetResponse();

logger.LogVerbose(string.Format(CultureInfo.CurrentCulture, "Successfully '{0}' file {1}", this.DocumentAction, theFileToUpload));

logger.LogOutcome(entries[this.SiteIndex], "SUCCCESS");

success = 1;

// Dispose of the HTTP response.

SPOResponse.Close();

break;

}

}

tempSW.Stop();

if (success != 1)

{

throw new Exception("The HTTP Request operation exceeded the timeout of 20 seconds");

}

}

}

}

}

catch(Exception ex)

{

logger.LogVerbose(string.Format(CultureInfo.CurrentCulture, "There was an issue performing '{0}' on to the URL '{1}' with exception: {2}", this.DocumentAction, entries[this.SiteIndex], ex.Message));

logger.LogOutcome(entries[this.SiteIndex], "FAILURE");

}

finally

{

IterationSW.Stop();

logger.LogVerbose(string.Format(CultureInfo.CurrentCulture, "Completed processing URL:'{0}' in {1} seconds", entries[this.SiteIndex], IterationSW.ElapsedMilliseconds/1000));

}

}