



Enterprise Content Management apps in SharePoint 2013 and SharePoint Online solution pack (Module 7 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.LargeFileUpload](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.LargeFileUpload)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this sample** | **How this app works** |
| This code sample shows how to upload large files to SharePoint, and how to bypass the 2MB file upload limit. | Consider using this sample when you want to:   * Upload files larger than 2MB to SharePoint. | This code sample runs as a console application that demonstrates how to upload large files to a document library using either the:   * SaveBinaryDirect method on the Microsoft.SharePoint.Client.File object. * ContentStream property on the FileCreationInformation object. |

**Related samples**:

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

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

# Understanding the code…

When you start this app, a console application displays as shown in Figure 1. You must supply a SharePoint Online site collection URL and your login credentials for Office 365.

**Figure 1. Core.LargeFileUpload console application.**



After authentication, the console application displays an error as shown in Figure 2. This is because the **UploadDocumentContent** method in **FileUploadService.cs** tries to upload a file larger than 2MB using the **FileCreationInformation.Content** property and throws an exception.

**Tip:** Avoid using the FileCreationInformation.Content property in your code because it restricts your file upload size to a maximum of 2MB.

public void UploadDocumentContent(ClientContext ctx, string libraryName, string filePath)

{

Web web = ctx.Web;

// Ensure that target library exists, create if it is missing

if (!LibraryExists(ctx, web, libraryName))

{

CreateLibrary(ctx, web, libraryName);

}

FileCreationInformation newFile = new FileCreationInformation();

// The next line of code causes an exception to be thrown for files > 2MB.

newFile.Content = System.IO.File.ReadAllBytes(filePath);

newFile.Url = System.IO.Path.GetFileName(filePath);

// Get instances to the given library.

List docs = web.Lists.GetByTitle(libraryName);

// Add file to the library.

Microsoft.SharePoint.Client.File uploadFile = docs.RootFolder.Files.Add(newFile);

ctx.Load(uploadFile);

ctx.ExecuteQuery();

}

**Note:** UploadDocumentContent creates a document library called **Docs** if it does not already exist. This document library is used later in this code sample.

**Figure 2. Using the FileCreationInformation.Content property to upload files larger than 2MB throws an exception**



The Core.LargeFileUpload sample then presents two options you can use to upload large files to a document library. These two options include using:

1. The **SaveBinaryDirect** method.
2. The **UploadDocumentContentStream** method.

In the following code, **SaveBinaryDirect** in **FileUploadService.cs** uses the **Microsoft.SharePoint.Client.File.SaveBinaryDirect** method with a FileStream object to upload files to a document library.

public void SaveBinaryDirect(ClientContext ctx, string libraryName, string filePath)

{

Web web = ctx.Web;

// Ensure that target library exists, create if it is missing

if (!LibraryExists(ctx, web, libraryName))

{

CreateLibrary(ctx, web, libraryName);

}

using (FileStream fs = new FileStream(filePath, FileMode.Open))

{

Microsoft.SharePoint.Client.File.SaveBinaryDirect(ctx, string.Format("/{0}/{1}", libraryName, System.IO.Path.GetFileName(filePath)), fs, true);

}

}

In the following code, **UploadDocumentContentStream** in **FileUploadService.cs** uses the **FileCreationInformation.ContentStream** property with the **Microsoft.SharePoint.Client.File** object to upload a file to a document library. You must assign a **FileStream** object to the **FileCreationInformation.ContentStream** property.

public void UploadDocumentContentStream(ClientContext ctx, string libraryName, string filePath)

{

Web web = ctx.Web;

// Ensure that target library exists, create if it is missing

if (!LibraryExists(ctx, web, libraryName))

{

CreateLibrary(ctx, web, libraryName);

}

using (FileStream fs = new FileStream(filePath, FileMode.Open))

{

FileCreationInformation flciNewFile = new FileCreationInformation();

// This is the key difference for the first case - using ContentStream property

flciNewFile.ContentStream = fs;

flciNewFile.Url = System.IO.Path.GetFileName(filePath);

flciNewFile.Overwrite = true;

List docs = web.Lists.GetByTitle(libraryName);

Microsoft.SharePoint.Client.File uploadFile = docs.RootFolder.Files.Add(flciNewFile);

ctx.Load(uploadFile);

ctx.ExecuteQuery();

}

}

When the console application closes, you can navigate to the **Docs** document library by choosing **Recent > Docs.** As shown in Figure 3, the document library contains two large files.

**Figure 3. Docs document library with two large files.**

