



Enterprise Content Management apps in SharePoint 2013 and SharePoint Online solution pack (Module 2 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.Autotagging app

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this sample** | **How this app works** |
| This sample app for SharePoint uses a provider-hosted app to show you how to automatically tag content when it is added to a library using remote event receivers.  This sample also shows you how to:   * Create fields, content types, and document libraries. * Retrieve the value of a custom user profile property. * Set taxonomy fields. | Consider using this sample when:   * You want to implement event receivers in SharePoint Online. * Improve search results by attaching additional metadata to the content at the time of creation. * Help classify your content. * Used event receivers in the past and now need to modernize your code before migrating to a newer version of SharePoint. | This app demonstrates the use of a remote event receiver on a list. The remote event receiver is hosted on an Azure Website. When a new document is added to the list, the remote event receiver updates a custom field in the list with data sourced from a custom user profile property. |

**Related samples**:

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

# Before you begin…

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

1. Create an Azure Website and deploy the ECM.AutoTaggingWeb project to the Azure Website.
2. Register your app using the **Appregnew.aspx** page in Office 365.
3. This app uses app only permissions. You need to assign app-only permissions using the **AppInv.aspx** page in Office 365. Copy the following XML from the **AppManifest.xml** file to the **Permission Request XML** textbox on the **AppInv.aspx** page, as shown in **Figure 1**.

<AppPermissionRequests AllowAppOnlyPolicy="true">

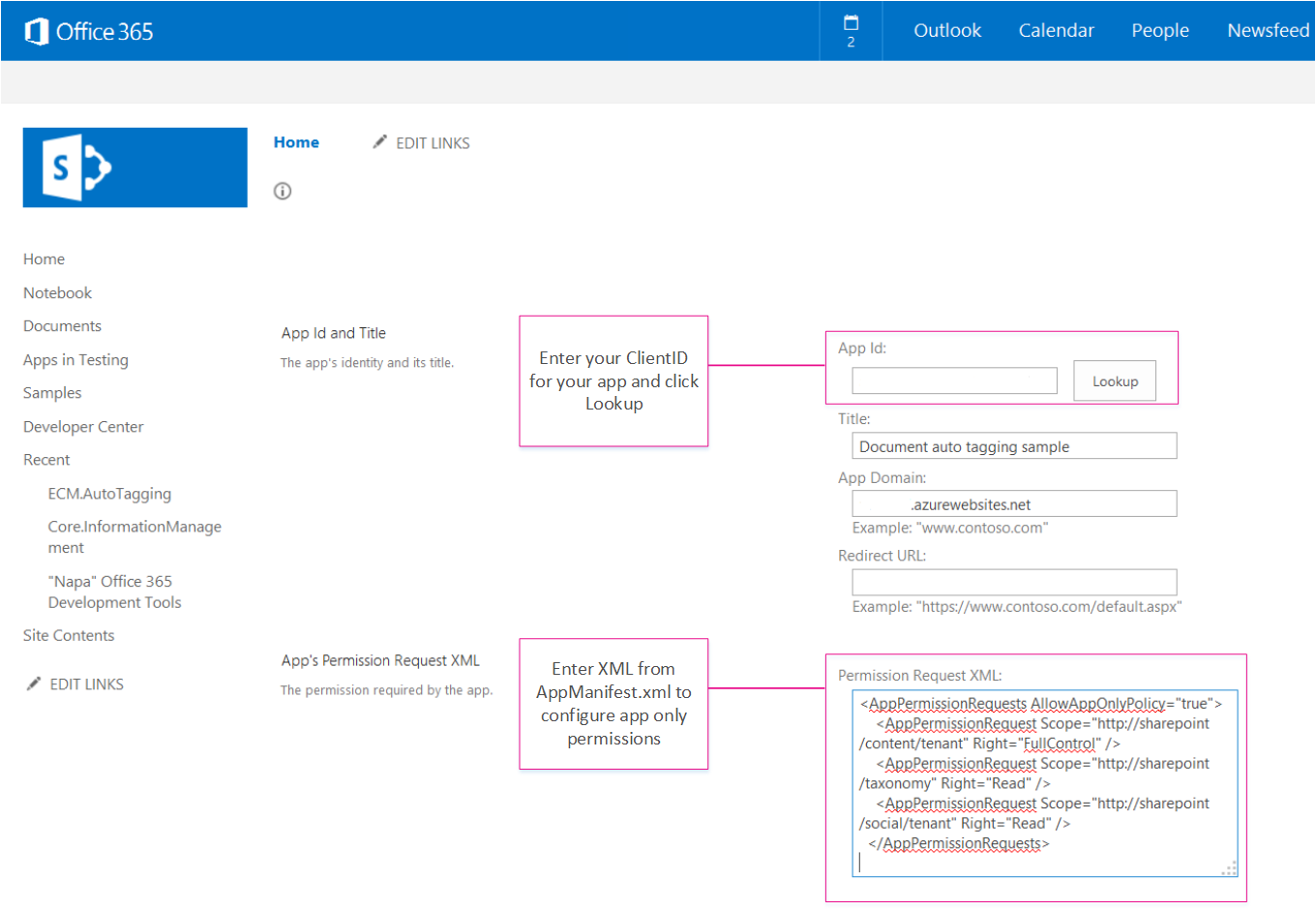
<AppPermissionRequest Scope="http://sharepoint/content/tenant" Right="FullControl" />

<AppPermissionRequest Scope="http://sharepoint/taxonomy" Right="Read" />

<AppPermissionRequest Scope="http://sharepoint/social/tenant" Right="Read" />

</AppPermissionRequests>

**Figure 1. Assign app-only permissions using the AppInv.aspx page in Office 365**

****

1. In the **ECM.AutoTaggingWeb** project, in the **ReceiverHelper.cs** file, in the **CreateEventReciever** method, you need to update the ReceiverUrl property with the URL of your Azure Website.

public static EventReceiverDefinitionCreationInformation CreateEventReciever(string receiverName, EventReceiverType type)

{

EventReceiverDefinitionCreationInformation \_rer = new EventReceiverDefinitionCreationInformation();

\_rer.EventType = type;

\_rer.ReceiverName = receiverName;

\_rer.ReceiverClass = "ECM.AutoTaggingWeb.Services.AutoTaggingService";

\_rer.ReceiverUrl = "https://<Your domain>.azurewebsites.net/Services/AutoTaggingService.svc";

\_rer.Synchronization = EventReceiverSynchronization.Synchronous;

return \_rer;

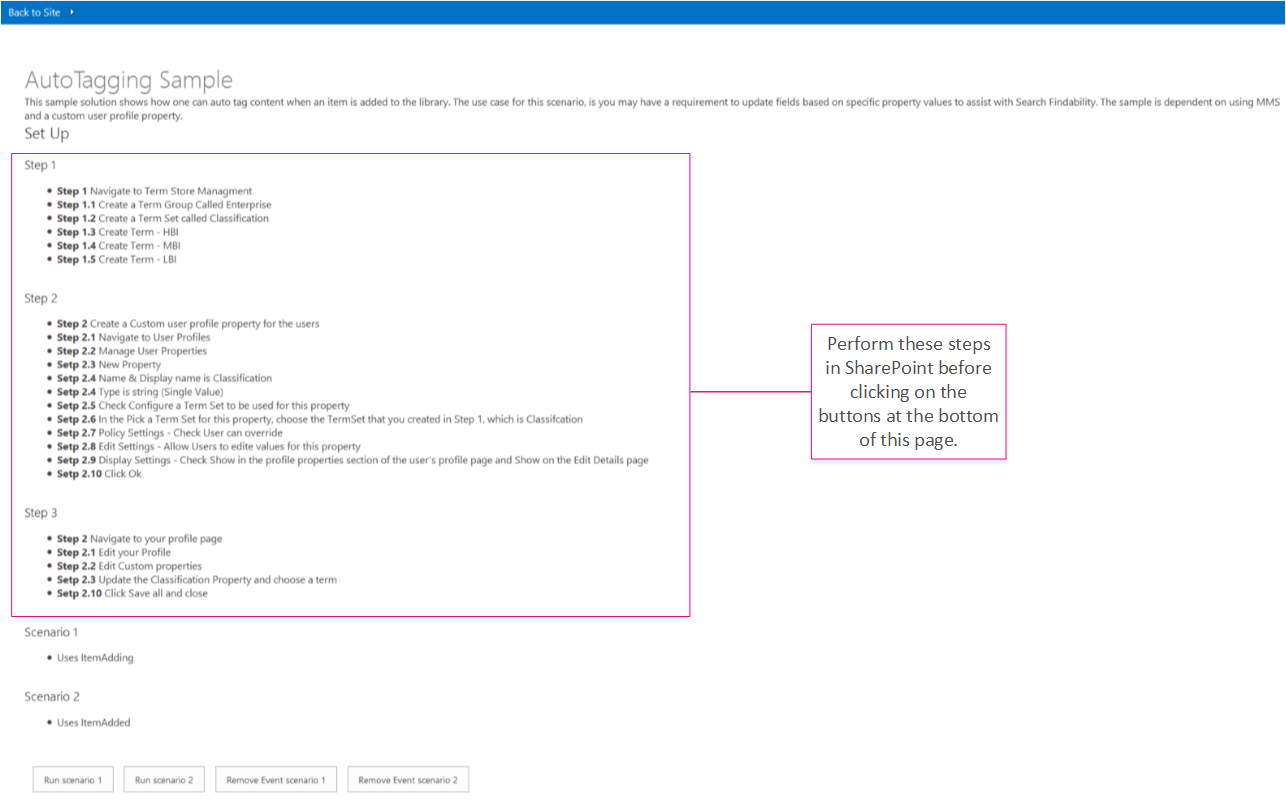
}

1. Package and deploy your app.

For more information, see [Setting up provider hosted app to Windows Azure for Office 365 tenant](http://blogs.msdn.com/b/vesku/archive/2013/11/25/setting-up-provider-hosted-app-to-windows-azure-for-office365-tenant.aspx).

When you start the app, the launch page of the Document Auto Tagging provider hosted app displays, as shown in Figure 2. The launch page shows some additional configuration steps you need to perform before assigning or removing the event receivers.

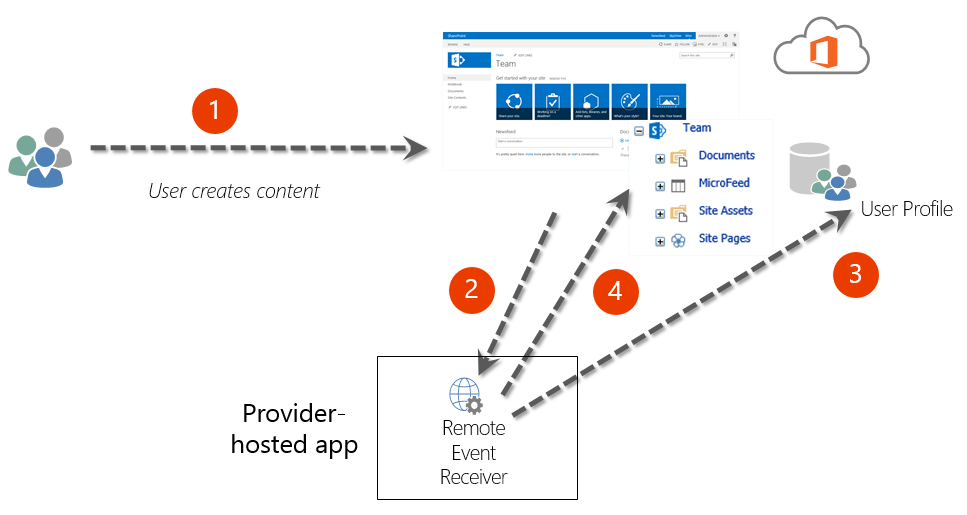
**Figure 2. Launch page of the app, highlighting additional configuration steps to be performed in SharePoint**



# Overview of the ECM.Autotagging app

This sample uses a remote event receiver to automatically tag documents (add metadata to documents) added to a document library, with data from a custom user profile property. The process flow for auto tagging documents using the remote event receiver is shown in Figure 3.

**Figure 3. Process flow for tagging documents in a document library using a remote event receiver**



As shown in Figure 3, the steps to assign metadata to the newly created document in the document library with a remote event receiver are:

1. A user creates or uploads new content to a document library. There is a remote event receiver assigned to handle ItemAdding or ItemAdded events on this document library.
2. A call is made to the remote event receiver (either the ItemAdding or ItemAdded method).
3. The provider-hosted eapp fetches the value of a custom user profile property in the User Profile Service of SharePoint for that user. In this code sample, the **Classification** custom user profile property that was added previously is retrieved.
4. The remote event receiver updates the metadata on the new document (tags the document) added to the document library with the value of the custom user profile property for that user.

Run Scenario 1

When you choose the button **Run Scenario 1**, the app performs the following steps:

1. Creates a document library.
2. Creates the remote event receiver for the ItemAdding event (Note: this document discusses the ItemAdding event receiver type. Generally, the ItemAdding event receiver type performs better than the ItemAdded event receiver type. The ECM.Autotagging solution provides code for both the ItemAdding and ItemAdded event receiver types).
3. Adds the remote event receiver to the document library.

The following code performs the above steps and can be found in the **btnScenario1\_Click** method of the **Default.aspx.cs** page in the **ECM.AutoTaggingWeb** project.

protected void btnScenario1\_Click(object sender, EventArgs e)

{

var \_libraryToCreate = this.GetLibaryInformationItemAdding();

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

using (var ctx = spContext.CreateUserClientContextForSPHost())

{

try

{

if(!ctx.Web.ListExists(\_libraryToCreate.Title))

{

ScenarioHandler \_scenario = new ScenarioHandler();

\_scenario.CreateContosoDocumentLibrary(ctx, \_libraryToCreate);

}

List \_list = ctx.Web.Lists.GetByTitle(\_libraryToCreate.Title);

EventReceiverDefinitionCreationInformation \_rec = ReceiverHelper.CreateEventReciever(ScenarioHandler.AUTOTAGGING\_ITEM\_ADDING\_RERNAME, EventReceiverType.ItemAdding);

ReceiverHelper.AddEventReceiver(ctx, \_list, \_rec);

}

catch(Exception \_ex)

{

}

}

}

As shown in the previous code, a call is made to the **CreateContosoDocumentLibrary** method. The following code in the **ScenarioHandler.cs** file uses methods from OfficeDevPnP.Core to create a custom document library with a custom content type. The default content type in the document library is removed.

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);

}

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.RemoveContentTypeByName("Document");

\_list.Update();

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

ctx.Web.Context.ExecuteQuery();

}

else

{

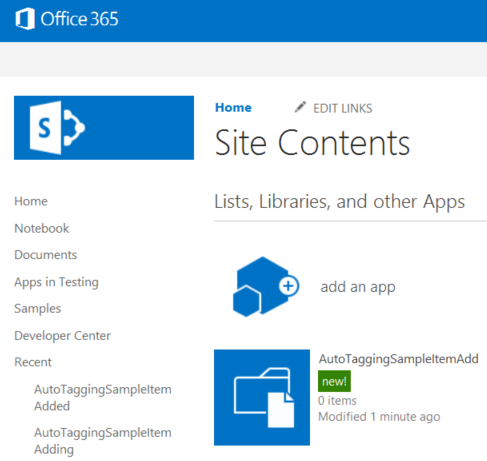
throw new Exception("A list, survey, discussion board, or document library with the specified title already exists in this Web site. Please choose another title.");

}

}

After this code completes, the **AutoTaggingSampleItemAdding** document library is created in **Site Contents**, as shown in Figure 4.

**Figure 4. AutoTaggingSampleItemAdding document library**



In the **ECM.AutoTaggingWeb** project, in the **ReceiverHelper.cs** file, the **CreateEventReciever** method creates the ItemAdding event receiver definition. In the ECM.AutoTaggingWeb project, in the Services folder, there is a web service called **AutoTaggingService.svc**. When you published the ECM.AutoTaggingWeb project to your Azure Website, this web service was also deployed to your Azure Website. The **CreateEventReciever** method assigns this web service as the remote event receiver on the document library. The following code from the **CreateEventReciever** method shows how to assign the web service to the remote event receiver.

public static EventReceiverDefinitionCreationInformation CreateEventReciever(string receiverName, EventReceiverType type)

{

EventReceiverDefinitionCreationInformation \_rer = new EventReceiverDefinitionCreationInformation();

\_rer.EventType = type;

\_rer.ReceiverName = receiverName;

\_rer.ReceiverClass = "ECM.AutoTaggingWeb.Services.AutoTaggingService";

\_rer.ReceiverUrl = "https://<Your domain>.azurewebsites.net/Services/AutoTaggingService.svc";

\_rer.Synchronization = EventReceiverSynchronization.Synchronous;

return \_rer;

}

The following code from the **AddEventReceiver** method assigns the remote event receiver to the document library.

public static void AddEventReceiver(ClientContext ctx, List list, EventReceiverDefinitionCreationInformation eventReceiverInfo)

{

if (!DoesEventReceiverExistByName(ctx, list, eventReceiverInfo.ReceiverName))

{

list.EventReceivers.Add(eventReceiverInfo);

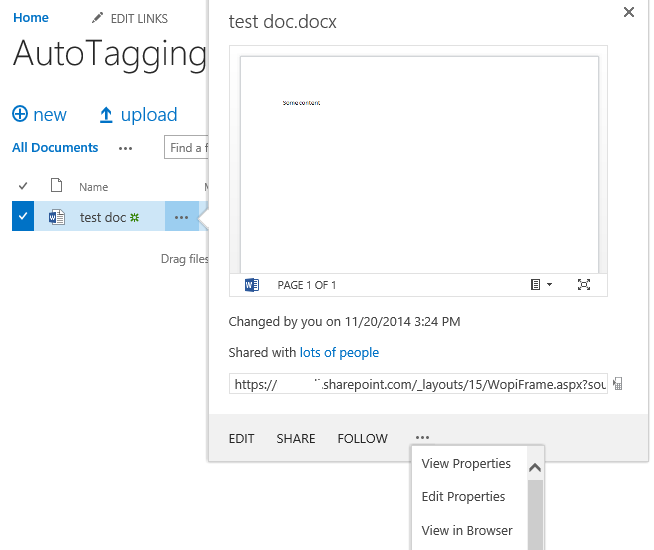
ctx.ExecuteQuery();

}

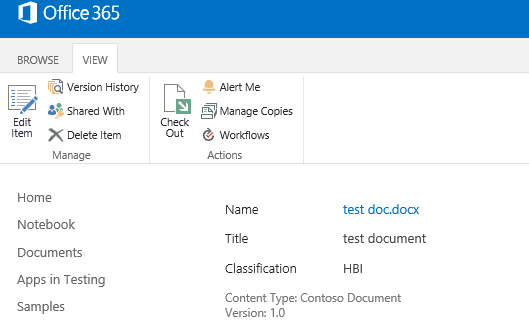
}

Now, the remote event receiver is added to the document library. When you upload documents to the **AutoTaggingSampleItemAdding** document library, the document will be tagged with the value of the Classification custom user profile property for that user. Figure 5 shows how to view the properties on a document. Figure 6 shows the document’s metadata with the Classification field.

**Figure 5. View document properties**



**Figure 6. A document’s metadata with the Classification field**



The **HandleAutoTaggingItemAdding** method uses the **GetProfilePropertyFor** method to retrieve the value of the **Classification** user profile property. The **HandleAutoTaggingItemAdding** method can be found in the **AutoTaggingService.svc.cs** file.

public void HandleAutoTaggingItemAdding(SPRemoteEventProperties properties,SPRemoteEventResult result)

{

using (ClientContext ctx = TokenHelper.CreateRemoteEventReceiverClientContext(properties))

{

if (ctx != null)

{

var itemProperties = properties.ItemEventProperties;

var \_userLoginName = properties.ItemEventProperties.UserLoginName;

var \_afterProperites = itemProperties.AfterProperties;

if(!\_afterProperites.ContainsKey(ScenarioHandler.FLD\_CLASSIFICATION\_INTERNAL\_NAME))

{

string \_classficationToSet = ProfileHelper.GetProfilePropertyFor(ctx, \_userLoginName, Constants.UPA\_CLASSIFICATION\_PROPERTY);

if(!string.IsNullOrEmpty(\_classficationToSet))

{

var \_formatTaxonomy = AutoTaggingHelper.GetTaxonomyFormat(ctx, \_classficationToSet);

result.ChangedItemProperties.Add(ScenarioHandler.FLD\_CLASSIFICATION\_INTERNAL\_NAME, \_formatTaxonomy);

}

}

}

}

}

**Important:** After retrieving the **Classification** value from the **GetProfilePropertyFor** method, the **Classification** value must be formatted in a certain way before it can be stored as metadata on the document. The **GetTaxonomyFormat** method in the **AutoTaggingHelper.cs** file shows how to format the Classification value.

public static string GetTaxonomyFormat(ClientContext ctx, string term)

{

if(string.IsNullOrEmpty(term))

{

throw new ArgumentException(string.Format(EXCEPTION\_MSG\_INVALID\_ARG, "term"));

}

string \_result = string.Empty;

var \_list = ctx.Web.Lists.GetByTitle(TAXONOMY\_HIDDEN\_LIST\_NAME);

CamlQuery \_caml = new CamlQuery();

\_caml.ViewXml = string.Format(TAXONOMY\_CAML\_QRY, term);

var \_listItemCollection = \_list.GetItems(\_caml);

ctx.Load(\_listItemCollection,

eachItem => eachItem.Include(

item => item,

item => item.Id,

item => item[TAXONOMY\_FIELDS\_IDFORTERM]));

ctx.ExecuteQuery();

if (\_listItemCollection.Count > 0)

{

var \_item = \_listItemCollection.FirstOrDefault();

var \_wssId = \_item.Id;

var \_termId = \_item[TAXONOMY\_FIELDS\_IDFORTERM].ToString(); ;

\_result = string.Format(TAXONOMY\_FORMATED\_STRING, \_wssId, term, \_termId);

}

return \_result;

}

Remove Event Scenario 1

When you choose the button **Remove Event Scenario 1**,the following code runs to remove the event receiver from the document library.

public static void RemoveEventReceiver(ClientContext ctx, List list, string receiverName)

{

ctx.Load(list, lib => lib.EventReceivers);

ctx.ExecuteQuery();

var \_rer = list.EventReceivers.Where(e => e.ReceiverName == receiverName).FirstOrDefault();

if(\_rer != null)

{

\_rer.DeleteObject();

ctx.ExecuteQuery();

}

}