



Enterprise Content Management apps in SharePoint 2013 and SharePoint Online solution pack (Module 4 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.RecordsManagement 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 how you can control the in-place records management settings for a site or list. | Consider using this sample when:   * You want to configure in-place records management settings during your custom site provisioning process. | This sample app shows how you can use a provider-hosted app to control in-place records management settings for sites or lists. This sample app uses OfficeDevPnP.Core, which provides a wrapper class to configure records management easily. In SharePoint 2013, you cannot configure records management easily using the client side object model. |

**Related samples**:

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

[ECM.DocumentLibraries](https://github.com/OfficeDev/PnP/tree/master/Scenarios/ECM.DocumentLibraries)

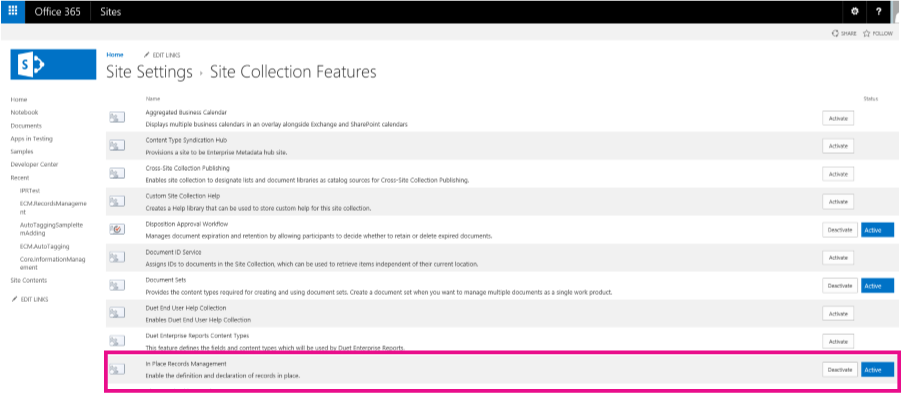
[ECM.AutoTagging](https://github.com/OfficeDev/PnP/tree/master/Scenarios/ECM.AutoTagging)

# Before you begin…

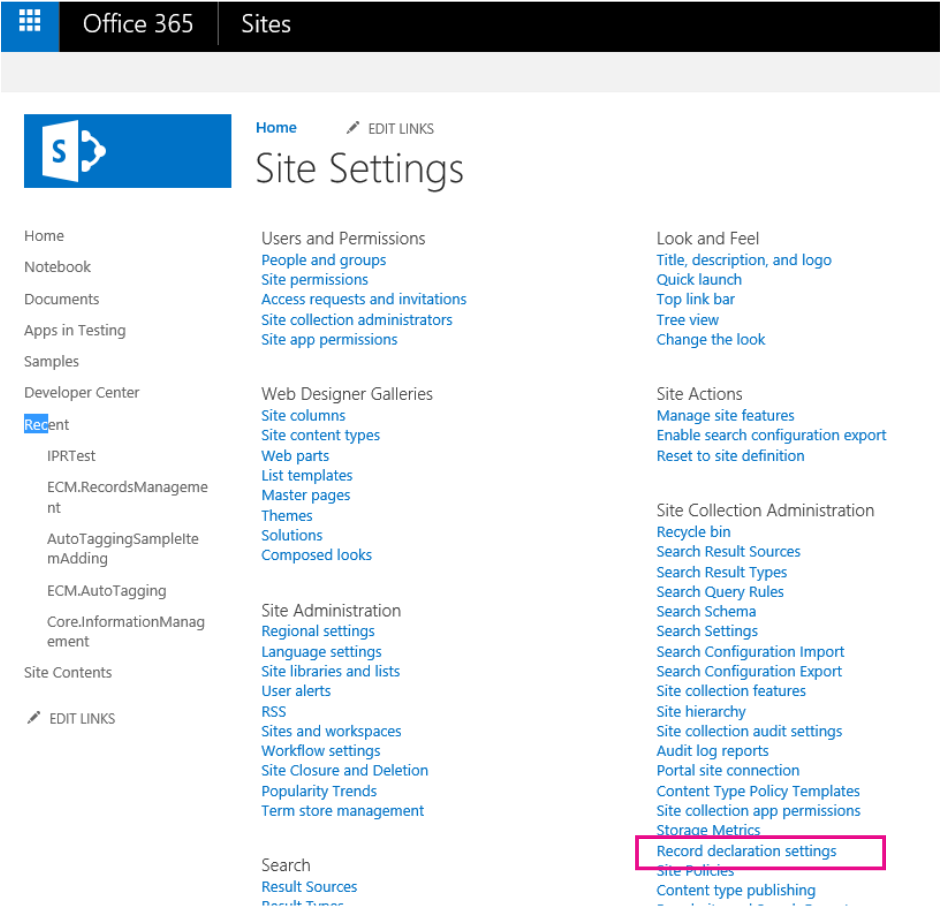
Ensure that you have performed the following steps before you run this app:

* Activate the **In-Place Records Management** feature on the site collection (see Figure 1).
* In site settings, verify that you see **Record declaration settings** in **Site Collection Administration** (see Figure 2).

**Figure 1. Activate In-Place Records Management on your site collection**



**Figure 2. Record declaration settings in Site Settings**



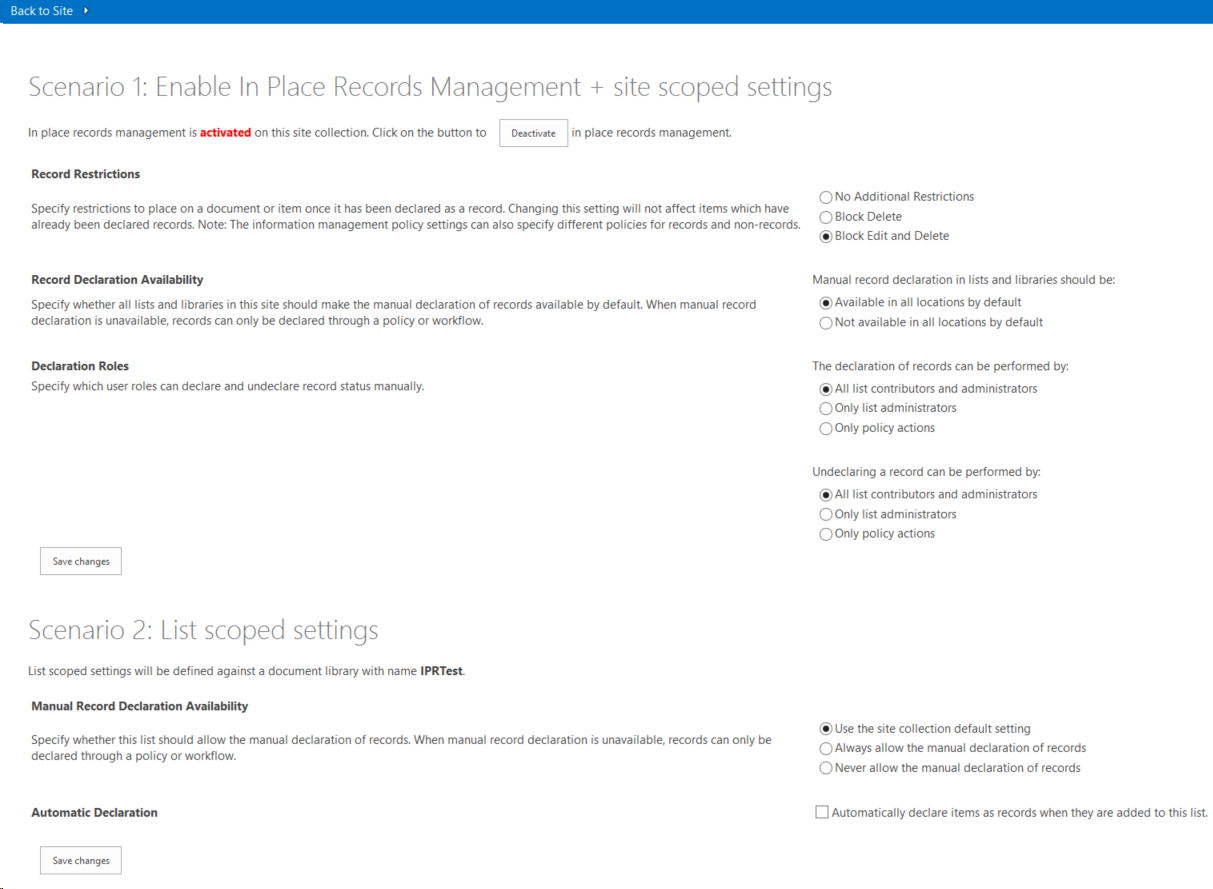
# Understanding the code…

When you start the app, the launch page displays the following information (as shown in Figure 3):

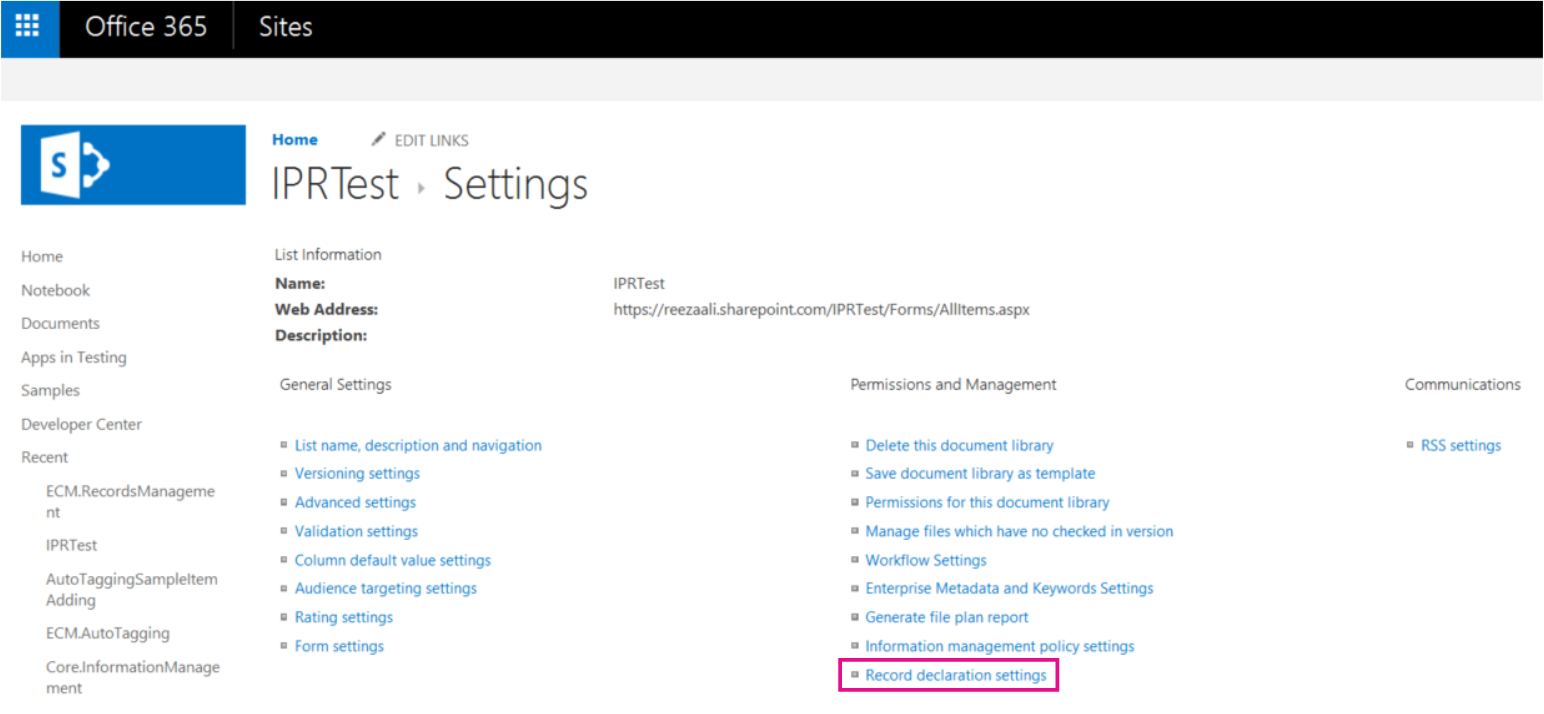
* In **Scenario 1**, this sample shows you how to build a UI to control the records management settings on your site collection. The UI in this sample is similar to the UI found in **Records declaration settings** in **Site Settings** (see Figure 2). In Scenario 1, you can also activate or deactivate the In-Place Records Management feature on your site collection.
* In **Scenario 2**, this sample shows you how to build a UI to control the records management settings on lists. The UI in this sample is similar to the UI found in **Records declaration settings** in the library settings on your list (see Figure 4 and Figure 5).

From your SharePoint site, you can navigate to the app, which runs on the remote host, by choosing **Recent > ECM.RecordsManagement**. To return to your SharePoint site, choose **Back to Site.**

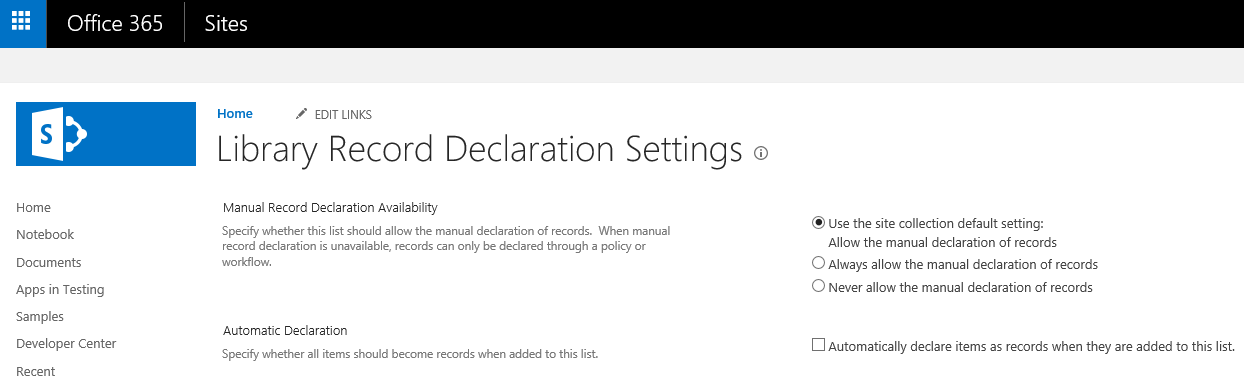
**Figure 3. ECM.RecordsManagement launch page**



**Figure 4. How to find Record declaration settings on a list**

****

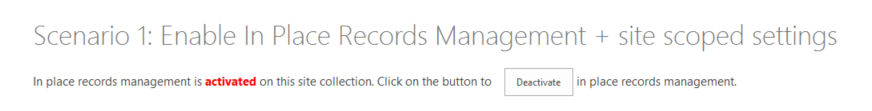
**Figure 5. Record declaration settings on a list**



# Scenario 1

Scenario 1 addresses in-place records management features and settings for sites. At the top of the app, there is a **Deactivate** (or **Activate**) button, as shown in Figure 6. Choosing this button deactivates (or activates) the In-Place Records Management feature on the site collection.

**Figure 6. Using the Deactivate button to deactivate the In-Place Records Management feature on your site collection**



The following code activates or deactivates the In-Place Records Management feature on the site collection. The **DisableInPlaceRecordsManagementFeature** and **EnableSiteForInPlaceRecordsManagement** methods are part of the **AppModelExtensions\RecordsManagementExtensions.cs** file in the **OfficeDevPnP.Core.**

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

{

if (cc.Site.IsInPlaceRecordsManagementActive())

{

cc.Site.DisableInPlaceRecordsManagementFeature();

IPRStatusUpdate(false);

}

else

{

cc.Site.EnableSiteForInPlaceRecordsManagement();

IPRStatusUpdate(true);

}

}

OfficeDevPnP.Core includes extension methods to get and set all site scoped in-place records management settings. The following code from the **EnableSiteForInPlaceRecordsManagement** method shows how to use these extension methods to set restrictions, and specify who can declare or undeclare records on your site.

public static void EnableSiteForInPlaceRecordsManagement(this Site site)

{

// Activate the "In-Place Records Management" feature if not yet enabled.

if (!site.IsFeatureActive(new Guid(INPLACE\_RECORDS\_MANAGEMENT\_FEATURE\_ID)))

{

// Note: this also sets the ECM\_SITE\_RECORD\_RESTRICTIONS value to "BlockDelete, BlockEdit".

site.ActivateInPlaceRecordsManagementFeature();

}

// Enable in-place records management in all locations.

site.SetManualRecordDeclarationInAllLocations(true);

// Set restrictions to default values after enablement (this is also done at feature activation).

EcmSiteRecordRestrictions restrictions = EcmSiteRecordRestrictions.BlockDelete | EcmSiteRecordRestrictions.BlockEdit;

site.SetRecordRestrictions(restrictions);

// Set record declaration to default value.

site.SetRecordDeclarationBy(EcmRecordDeclarationBy.AllListContributors);

// Set record undeclaration to default value.

site.SetRecordUnDeclarationBy(EcmRecordDeclarationBy.OnlyAdmins);

}

When the user changes their in-place records management settings and chooses the **Save changes** button, the following code in the **btnSaveSiteScopedIPRSettings\_Click** method runs.

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

{

EcmSiteRecordRestrictions restrictions = (EcmSiteRecordRestrictions)Convert.ToInt32(rdRestrictions.SelectedValue);

cc.Site.SetRecordRestrictions(restrictions);

cc.Site.SetManualRecordDeclarationInAllLocations(Convert.ToBoolean(rdAvailability.SelectedValue));

EcmRecordDeclarationBy declareBy = (EcmRecordDeclarationBy)Convert.ToInt32(rdDeclarationBy.SelectedValue);

cc.Site.SetRecordDeclarationBy(declareBy);

EcmRecordDeclarationBy unDeclareBy = (EcmRecordDeclarationBy)Convert.ToInt32(rdUndeclarationBy.SelectedValue);

cc.Site.SetRecordUnDeclarationBy(unDeclareBy);

}

In the previous code, a call is made to the **SetRecordRestrictions** method in **RecordsManagementExtensions.cs.** The **SetRecordRestrictions** method shows how to set restrictions on the records.

public static void SetRecordRestrictions(this Site site, EcmSiteRecordRestrictions restrictions)

{

string restrictionsProperty = "";

if (restrictions.Has(EcmSiteRecordRestrictions.None))

{

restrictionsProperty = EcmSiteRecordRestrictions.None.ToString();

}

else if (restrictions.Has(EcmSiteRecordRestrictions.BlockEdit))

{

// BlockEdit is always used in conjunction with BlockDelete.

restrictionsProperty = EcmSiteRecordRestrictions.BlockDelete.ToString() + ", " + EcmSiteRecordRestrictions.BlockEdit.ToString();

}

else if (restrictions.Has(EcmSiteRecordRestrictions.BlockDelete))

{

restrictionsProperty = EcmSiteRecordRestrictions.BlockDelete.ToString();

}

// Set property bag entry.

site.RootWeb.SetPropertyBagValue(ECM\_SITE\_RECORD\_RESTRICTIONS, restrictionsProperty);

}

# Scenario 2

Scenario 2 shows how to interact with in-place records management settings for lists. When this app installs, the app creates a document library called **IPRTest**. When you use this sample app to change and save the in-place records management settings, the changes are applied to IPRTest.

**Note:** To use in-place records management settings on a list, you must activate the **In-place Records Management** feature on your site collection.

The following code in **Default.aspx.cs** runs when a user chooses the **Save Changes** button.

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

{

List ipr = cc.Web.GetListByTitle(IPR\_LIBRARY);

EcmListManualRecordDeclaration listManual = (EcmListManualRecordDeclaration)Convert.ToInt32(rdListAvailability.SelectedValue);

ipr.SetListManualRecordDeclaration(listManual);

ipr.SetListAutoRecordDeclaration(chbAutoDeclare.Checked);

// Refresh the settings as AutoDeclare changes the manual settings.

if (ipr.IsListRecordSettingDefined())

{

rdListAvailability.SelectedValue = Convert.ToString((int)ipr.GetListManualRecordDeclaration());

chbAutoDeclare.Checked = ipr.GetListAutoRecordDeclaration();

rdListAvailability.Enabled = !chbAutoDeclare.Checked;

}

}

The code calls the following two methods in the **RecordsManagementExtensions.cs** file of **OfficeDevPnP.Core**:

* **SetListManualRecordDeclaration** – Defines the manual records declaration setting for this list.
* **SetListAutoRecordDeclaration** – Automatically declare items added to this list as a record. If records declaration is set to automatic on this list, the manual records declaration settings on the list no longer apply. Event receivers are added to the list to start specific records management actions when events occur.

public static void SetListManualRecordDeclaration(this List list, EcmListManualRecordDeclaration settings)

{

if (settings == EcmListManualRecordDeclaration.UseSiteCollectionDefaults)

{

// If you set list record declaration back to the default values, you also need to

// turn off auto record declaration. Other property bag values are left as is; when

// settings are changed again these properties are also again usable.

if (list.PropertyBagContainsKey(ECM\_AUTO\_DECLARE\_RECORDS))

{

list.SetListAutoRecordDeclaration(false);

}

// Set the property that dictates custom list record settings to false.

list.SetPropertyBagValue(ECM\_IPR\_LIST\_USE\_LIST\_SPECIFIC, false.ToString());

}

else if (settings == EcmListManualRecordDeclaration.AlwaysAllowManualDeclaration)

{

list.SetPropertyBagValue(ECM\_ALLOW\_MANUAL\_DECLARATION, true.ToString());

// Set the property that dictates custom list record settings to true.

list.SetPropertyBagValue(ECM\_IPR\_LIST\_USE\_LIST\_SPECIFIC, true.ToString());

}

else if (settings == EcmListManualRecordDeclaration.NeverAllowManualDeclaration)

{

list.SetPropertyBagValue(ECM\_ALLOW\_MANUAL\_DECLARATION, false.ToString());

// Set the property that dictates custom list record settings to true.

list.SetPropertyBagValue(ECM\_IPR\_LIST\_USE\_LIST\_SPECIFIC, true.ToString());

}

else

{

throw new ArgumentOutOfRangeException("settings");

}

}

public static void SetListAutoRecordDeclaration(this List list, bool autoDeclareRecords)

{

// Determine the SharePoint version based on the loaded CSOM library.

Assembly asm = Assembly.GetAssembly(typeof(Microsoft.SharePoint.Client.Site));

int sharePointVersion = asm.GetName().Version.Major;

if (autoDeclareRecords)

{

// Set the property that dictates custom list record settings to true.

list.SetPropertyBagValue(ECM\_IPR\_LIST\_USE\_LIST\_SPECIFIC, true.ToString());

// Prevent manual declaration.

list.SetPropertyBagValue(ECM\_ALLOW\_MANUAL\_DECLARATION, false.ToString());

// Hook up the needed event handlers.

list.Context.Load(list.EventReceivers);

list.Context.ExecuteQuery();

List<EventReceiverDefinition> currentEventReceivers = new List<EventReceiverDefinition>(list.EventReceivers.Count);

currentEventReceivers.AddRange(list.EventReceivers);

// Track changes to see if a list.Update is needed.

bool eventReceiverAdded = false;

// ItemUpdating receiver.

EventReceiverDefinitionCreationInformation newEventReceiver = CreateECMRecordEventReceiverDefinition(EventReceiverType.ItemUpdating, 1000, sharePointVersion);

if (!ContainsECMRecordEventReceiver(newEventReceiver, currentEventReceivers))

{

list.EventReceivers.Add(newEventReceiver);

eventReceiverAdded = true;

}

// ItemDeleting receiver.

newEventReceiver = CreateECMRecordEventReceiverDefinition(EventReceiverType.ItemDeleting, 1000, sharePointVersion);

if (!ContainsECMRecordEventReceiver(newEventReceiver, currentEventReceivers))

{

list.EventReceivers.Add(newEventReceiver);

eventReceiverAdded = true;

}

// ItemFileMoving receiver.

newEventReceiver = CreateECMRecordEventReceiverDefinition(EventReceiverType.ItemFileMoving, 1000, sharePointVersion);

if (!ContainsECMRecordEventReceiver(newEventReceiver, currentEventReceivers))

{

list.EventReceivers.Add(newEventReceiver);

eventReceiverAdded = true;

}

// ItemAdded receiver.

newEventReceiver = CreateECMRecordEventReceiverDefinition(EventReceiverType.ItemAdded, 1005, sharePointVersion);

if (!ContainsECMRecordEventReceiver(newEventReceiver, currentEventReceivers))

{

list.EventReceivers.Add(newEventReceiver);

eventReceiverAdded = true;

}

// ItemUpdated receiver.

newEventReceiver = CreateECMRecordEventReceiverDefinition(EventReceiverType.ItemUpdated, 1007, sharePointVersion);

if (!ContainsECMRecordEventReceiver(newEventReceiver, currentEventReceivers))

{

list.EventReceivers.Add(newEventReceiver);

eventReceiverAdded = true;

}

// ItemCheckedIn receiver.

newEventReceiver = CreateECMRecordEventReceiverDefinition(EventReceiverType.ItemCheckedIn, 1006, sharePointVersion);

if (!ContainsECMRecordEventReceiver(newEventReceiver, currentEventReceivers))

{

list.EventReceivers.Add(newEventReceiver);

eventReceiverAdded = true;

}

if (eventReceiverAdded)

{

list.Update();

list.Context.ExecuteQuery();

}

// Set the property that dictates the autodeclaration.

list.SetPropertyBagValue(ECM\_AUTO\_DECLARE\_RECORDS, autoDeclareRecords.ToString());

}

else

{

// Set the property that dictates the autodeclaration.

list.SetPropertyBagValue(ECM\_AUTO\_DECLARE\_RECORDS, autoDeclareRecords.ToString());

//Note: Existing list event handlers will just stay as they are, no need to remove them.

}

}