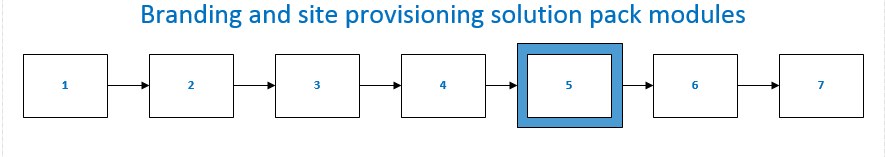
Module 5: SharePoint 2013 site provisioning

# Code sample inventory

This section introduces the site provisioning samples in the [sample pack](http://code.msdn.microsoft.com/SharePoint-2013-Sample-81b03d1e) companion to the solution pack documents. The SharePoint 2013 CSOM provides APIs for creating subsites, OneDrive for Business sites, and site collections.



The samples listed in this table demonstrate how to provision each kind of site. The table also indicates which UI and request processing patterns each sample uses.

**Table 1. Site provisioning code samples and resources**

|  |  |  |
| --- | --- | --- |
| **Sample** | **Description and additional resources** | **Pattern** |
| [Office365 - Apply automatically custom branding to personal site (OneDrive for Business)](http://blogs.msdn.com/b/vesku/archive/2013/11/25/office365-apply-automatically-custom-branding-to-personal-site-skydrive-pro.aspx) (Vesa Juvonen)  [OneDriveForBusinessSiteBranding](http://code.msdn.microsoft.com/SharePoint-2013-Brand-a-6da627cb) | Applies custom branding elements to a OneDrive for Business site. | On-demand provisioning of OneDrive for Business sites on Office 365 |
| [Provisioning site collections using SP App model in on-premises with just CSOM](http://blogs.msdn.com/b/vesku/archive/2014/06/09/provisioning-site-collections-using-sp-app-model-in-on-premises-with-just-csom.aspx) | Provisioning a site collection with CSOM in an on-premises or Dedicated farm |  |
| [Site provisioning techniques and remote provisioning in SharePoint 2013](http://blogs.msdn.com/b/vesku/archive/2013/08/23/site-provisioning-techniques-and-remote-provisioning-in-sharepoint-2013.aspx) (Vesa Juvonen) |  | On-demand provisioning of subsites |
| [Self-Service Site Provisioning using Apps for SharePoint 2013](http://blogs.msdn.com/b/richard_dizeregas_blog/archive/2013/04/04/self-service-site-provisioning-using-apps-for-sharepoint-2013.aspx) (Richard diZerega) |  | On-demand provisioning of subsites and site collections, with form for desired customizations |
| [SharePoint 2013: Provision dedicated and on-premises sites with the app model](http://code.msdn.microsoft.com/SharePoint-2013-Provision-523764b8) | Provision site collections in an on-premises installation of SharePoint 2013 or SharePoint Online Dedicated by using the client object model. | Asynchronous on-demand provisioning of site collections. |
| [Batch Provisioning](http://code.msdn.microsoft.com/Provision-sites-in-batches-fcf31bc6) | Provision site collections in a console app. | Site collection provisioning in a console app |
| [SiteProvisioningWorkflow](http://code.msdn.microsoft.com/SharePoint-2013-Use-e2ee88dd) | Provision site collections with a workflow on the app web and a remote event receiver  [SharePoint 2013 workflow: Create a custom action](http://code.msdn.microsoft.com/sharepoint/SharePoint-2013-workflow-41e5c0f9) | Approval workflow on the app web with site collection provisioning in a remote event receiver. |
| [SiteProvisioningWorkflowAppWeb](http://code.msdn.microsoft.com/SharePoint-2013-Use-2b96feb7) | Provision site collections with a workflow on the host web and a remote event receiver. | Site collection provisioning in a remote event receiver, with a workflow (identical to the app web workflow) deployed to the host web. |
| [Use sideloading to provision a provider-hosted app](http://code.msdn.microsoft.com/SharePoint-2013-Use-315c531b) | Use the SharePoint 2013 side loading feature to install a provider-hosted app. | This method bypasses the regular governance measures associated with deploying apps and doesn't require site owners to deploy apps from the app catalog |

# Site Collection Provisioning: Three patterns

The samples fall into three general patterns.

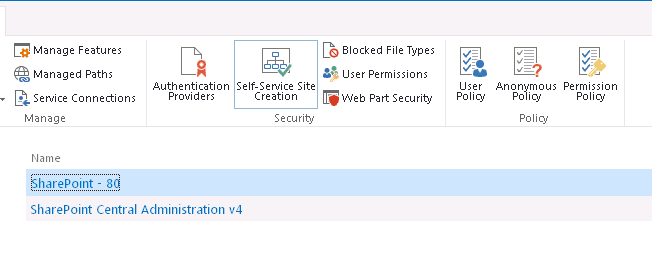
1. On-demand, with desired customizations passed through a form.
2. Approval workflow (with site provisioning code executed in a remote event receiver)
3. Batch processing in a console app or app-based remote timer job

The APIs for creating site collections, subsites, and OneDrive for Business sites are different. Only the on-demand pattern applies to OneDrive for Business sites, because the code that provisions personal sites must run under the identity of the user who owns the site. You can apply the other two patterns to creation of all other types of SharePoint sites**.** The [Self-Service Site Provisioning using Apps for SharePoint 2013](http://blogs.msdn.com/b/richard_dizeregas_blog/archive/2013/04/04/self-service-site-provisioning-using-apps-for-sharepoint-2013.aspx)sample by Richard diZerega demonstrates this by enabling creation of both subsites and site collections through a [customization form](http://blogs.msdn.com/b/richard_dizeregas_blog/archive/2013/04/04/self-service-site-provisioning-using-apps-for-sharepoint-2013.aspx).

The first three samples follow the on-demand model and are described more fully in their associated blog posts. The on-demand samples demonstrate how to use a custom form that runs on a remote (non-SharePoint) web site to pass desired customizations to the code that creates the site. Once you have created a custom form, you can override the default Self Service Site Creation form for your site collection with your own by setting a value for the Custom Form URL in the Self-Service Suite Creation settings for the web application.

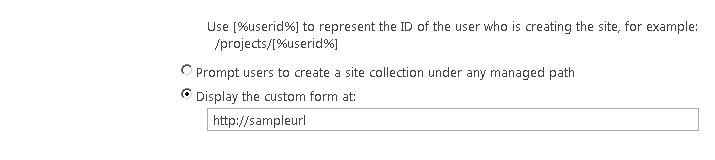
You can find this setting by opening your Central Administration site and navigating to the **Web Applications Management** page (**Application Management** > **Manage Web Applications**).  Select the web application that you want to modify and then click the **Self-Service Site Creation** button (Figure 1).

**Figure 1. Self Service Site Creation button for a web application**



That button launches the Self-Service Site Creation Management form. Click the check box labelled Display Custom form at: and enter the URL for your custom form (Figure 2).

**Figure 2. Self Service Site Creation button for a web application**



You can also override the default Self Service Site Creation form by setting a value for the [SPWebApplication.SelfServiceSiteCustomFormUrl](http://msdn.microsoft.com/en-us/library/office/microsoft.sharepoint.administration.spwebapplication.selfservicesitecustomformurl(v=office.15).aspx) property. You can set that value with this PowerShell script:

$webApp = Get-SPWebApplication “https://<siteurl>”

$webApp.ShowStartASiteMenuItem = $true

$webApp.SelfServiceSiteCustomFormUrl = “https://<customformurl>”

$webApp.Update()

The two workflow samples and the batch provisioning console app demonstrate how to build frameworks around the code that provisions site collections in order to make that code work in scenarios that require approval workflows and batch processing of site creation requests.

**Important** Before running the approval workflow or batch provisioning samples, enable the app-only policy by opening the AppManifest.xml editor and checking the **Allow the app to make app-only calls to SharePoint** check box.

The samples that use the app-only policy (workflow and batch processing) require full control permission at the Tenant scope. You also need to create client context with the URL for the tenant admin site.

## Provision site collections with a workflow on the app web and a remote event receiver

This sample demonstrates:

* Declarative creation of workflow to the app web.
* Declarative creation of custom list to the app web.
* Declarative association of remote event receiver with custom list on app web.
* Approval workflow triggers site collection provisioning.
* Remote site collection provisioning in a remote event receiver using the app-only policy.

This sample deploys a simple approval workflow and an associated custom list to the app web. List item creation triggers a new workflow and assigns the approval task to the person or group in the **Approver** field of the custom list. If the **Approver** approves the request, the **Approved** field value changes from false to true. When an item in the list is updated, the remote event receiver provisions a site collection and gives it the name supplied by the user in the **Title** field.

Since the workflow and custom list are deployed to the app web, the sample creates both declaratively. It also associates the remote event receiver with the list declaratively. When you use a workflow that is deployed to the host web, you need to write code in order to associate the remote event receiver with the list. The second workflow sample shows how to do this.

The workflow resets the value of the **Title** field to its original value so that this value will be delivered to the remote event receiver (on the item updated event). This allows you to avoid having to retrieve the list item in the remote event receiver. The sample assumes that you are storing the customizations in the custom list that records the site creation request. You could also store the site metadata in a remote data store.

The user interface consists solely of the default list display and editing page on the app web. You could also build a form that runs on a remotely hosted page, as in the second workflow sample.

## Provision site collections with a workflow on the host web and a remote event receiver

This sample demonstrates:

* Using the app installed event to associate a remote event receiver with a custom list on the host web.
* Remote site collection provisioning in a remote event receiver using the app-only policy.

This sample assumes that you’ve created on the host web an approval workflow and associated list that are identical to the ones deployed to the app web in the **Provision site collections with a workflow on the app web and a remote event receiver** sample. It uses a remote event receiver that is identical to the one in that sample.

The sample uses the app installed event to associate the remote event receiver with the custom list on the host web. You create the app installed event receiver by checking the **Handle App Installed** property in the app’s project properties. You can’t make this association declaratively when the list is on the host web.

The code in snippet B shows how to associate a remote event receiver with the item updated event in a custom list.

You could also use the app installed event receiver to provision the custom list and the workflow on the host web. [How to: Complete basic operations using SharePoint 2013 client library code](http://msdn.microsoft.com/en-us/library/office/fp179912.aspx#BasicOps_SPListTasks) shows how to create a custom list, and [Working with the SharePoint 2013 Workflow Services Client Side Object Model](http://msdn.microsoft.com/en-us/library/office/dn481315.aspx) shows how to create a workflow with the client object model.

The code in the remote event receiver is nearly identical to the code in the first workflow sample. It sets the owner of the site collection to a default value. You could update the form to make the site owner a customizable value.

The user interface consists of a simple form that runs on the remote web application. It allows the user to set the name of the site collection. This form could be enhanced to allow the user to set the person or group who will own the site collection.

## Provision site collections in a console app

The [Batch Provisioning](http://code.msdn.microsoft.com/Provision-sites-in-batches-fcf31bc6) sample demonstrates:

* Configuration of a remote app for SharePoint that runs as a console application.
* Remote site collection provisioning in a console application using the app-only policy.

Before you test this app, you must first create a client id and client secret with the 15/\_layouts/appregnew.aspx page. Place these values in the app.config file of the console application. This configuration file is equivalent to the web.config file in the web application project. Place the URLs for the new site collections that you want to create in the Sites.xml file of the console application. You’ll also have to enter values in the Program.cs file for your admin site URL and the admin account that you’ll want to associate with the new site collection or collections.

The **Batch ProvisioningWeb** project is part of the standard provider-hosted app solution template in Visual Studio 2013. It is not required for this pattern, but you can use it to make sure that you have correctly configured your app. Add the client id and secret to the web.config file, and you can verify that those values are correct by using F5 debugging. If the site name appears on the page that loads after you click **Yes** on the **Trust It** page, you can be sure that you are using the correct client id and secret. Place these values in the app.config file of the **Batch Provisioning Console** project.

The **Batch ProvisioningWeb** project also contains the TokenHelper.cs file that you will need to copy into the **Batch Provisioning Console** project.

## Use batch provisioning to manage changes to SharePoint site branding elements

You can use the [Batch Provisioning](http://code.msdn.microsoft.com/Provision-sites-in-batches-fcf31bc6) sample to update and manage at scale changes to the branding elements of SharePoint sites.

When applying branding elements to sites using SharePoint solutions and the feature framework, you may have requested that branding updates and changes be applied to the hive and shared across site collections. This made it possible to update all branding assets in a single location.

With the app model, use the same approach.

* Minimize the impact of changes introduced to site branding elements by storing CSS, JavaScript, and image files in a single site collection.
* Reference them using a relative path. Using this approach, the only file that spans site collections is the master page.

You can run batch provisioning in a console application that you can host remotely—it uses the app-only permission policy.

# Completing site provisioning tasks with CSOM

This section includes code that demonstrates:

* Creating a site collection
* Associating a remote event receiver with a custom list on the host web in the app installed event

## Use the TenantAdministration CSOM to create a site collection on SharePoint Online

This code demonstrates how to use the [TenantAdministration CSOM](http://msdn.microsoft.com/en-us/library/office/microsoft.online.sharepoint.tenantadministration.aspx) to create a site collection on SharePoint Online.

**Note** This code only works on an Office 365 Multi-Tenant (MT) SharePoint site. See Section 4 for instructions on how to do nearly the same thing on an on-premises or Dedicated SharePoint farm.

//The assembly for this is in Program Files\SharePoint Client Components\Assemblies

using Microsoft.Online.SharePoint.TenantAdministration;

//This is hard-coded for SharePoint Online (ie - all tenants)

string SHAREPOINT\_PID = "00000003-0000-0ff1-ce00-000000000000";

//The app must have tenant-level permissions and can be installed on any site

//in the tenancy. You must use the tenant

//admin site URL to get client context.

Uri sharePointUrl = new Uri("https://<site>-admin.sharepoint.com");

string myRealm = TokenHelper.GetRealmFromTargetUrl(sharePointUrl);

try

{

string accessToken = TokenHelper.GetAppOnlyAccessToken(SHAREPOINT\_PID, sharePointUrl.Authority, myRealm).AccessToken;

using (ClientContext clientContext = TokenHelper.GetClientContextWithAccessToken(sharePointUrl.ToString(), accessToken))

{

if (clientContext != null)

{

//Retrieved from a list item property that has been reset by the workflow.

//ListItemUpdated activity

var requestTitle = properties.ItemEventProperties.AfterProperties["Title"];

var tenant = new Tenant(clientContext);

var newSite = new SiteCreationProperties()

{

Url = "https://<site name>.sharepoint.com/sites/" + requestTitle,

Owner = "administrator@<site>.onmicrosoft.com",

Template = "STS#0",

Title = "Workflow provisioning test site",

StorageMaximumLevel = 1000,

StorageWarningLevel = 500,

TimeZoneId = 7,

UserCodeMaximumLevel = 7,

UserCodeWarningLevel = 1,

};

var spoOperation = tenant.CreateSite(newSite);

clientContext.Load(spoOperation);

clientContext.ExecuteQuery();

while (!spoOperation.IsComplete)

{

System.Threading.Thread.Sleep(2000);

clientContext.Load(spoOperation);

clientContext.ExecuteQuery();

}

}

}

}

catch (Exception ex)

{

//Handle exception;

}

}

## Associate a remote event receiver with a custom list on the host web in the app installed event

This example associates a remote event receiver on the SharePoint host web in the app-installed event.

SPRemoteEventResult result = new SPRemoteEventResult();

using (ClientContext clientContext = TokenHelper.CreateAppEventClientContext(properties, false))

{

if (clientContext != null)

{

clientContext.Load(clientContext.Web);

clientContext.ExecuteQuery();

var requestProperty = (System.ServiceModel.Channels.HttpRequestMessageProperty)System.ServiceModel.OperationContext.Current.IncomingMessageProperties[System.ServiceModel.Channels.HttpRequestMessageProperty.Name];

string opContext = System.ServiceModel.OperationContext.Current.Channel.LocalAddress.Uri.AbsoluteUri.Substring(0, System.ServiceModel.OperationContext.Current.Channel.LocalAddress.Uri.AbsoluteUri.LastIndexOf("/"));

string remoteUrl = string.Format("{0}/CreateSite.svc", opContext);

//string remoteUrl = string.Format("{0}/CreateSite2.svc", System.ServiceModel.OperationContext.Current.Channel.LocalAddress.Uri.DnsSafeHost + "/services");

var appWebUrl = "https://" + requestProperty.Headers[System.Net.HttpRequestHeader.Host];

List createSiteRequests = clientContext.Web.Lists.GetByTitle("SiteCreationRequests");

if (properties.EventType == SPRemoteEventType.AppInstalled)

{

EventReceiverDefinitionCreationInformation newEventReceiver = new EventReceiverDefinitionCreationInformation()

{

EventType = EventReceiverType.ItemUpdated,

ReceiverName = "CreateSite",

ReceiverUrl = remoteUrl,

SequenceNumber = 1000 //Should be higher number if lower priority, particularly for async events

};

createSiteRequests.EventReceivers.Add(newEventReceiver);

clientContext.ExecuteQuery();

}

}

}

return result;

}

# How do I provision site collections with CSOM on an on-premises or Office 365 Dedicated SharePoint farm?

If you have are working with an on-premises or Office 365 Dedicated (D) SharePoint farm, you can use nearly identical code to create site collections, but you have to prepare your farm in order to make the code work. You can see a fuller version of these instructions in Vesa Juvonen’s blog post, [Provisioning site collections using SP App model in on-premises with just CSOM](http://blogs.msdn.com/b/vesku/archive/2014/06/09/provisioning-site-collections-using-sp-app-model-in-on-premises-with-just-csom.aspx).

You’ll need to do these four things in order to prepare your farm for site creation with CSOM:

1. Install the SharePoint 2013 April 2014 CU. Stefan Goßner provides links and additional context for this CU on his blog post, [April 2014 CU for SharePoint 2013 has finally been released](http://blogs.technet.com/b/stefan_gossner/archive/2014/05/08/april-2014-cu-for-sharepoint-2013-has-finally-been-released.aspx).
2. Enable site collection creation with CSOM by running the following PowerShell script:  
     
   # Enable remote site collection creation for on-premises and Dedicated farms

# at the web application level

# If this is not done, you will get an unknown object exception

#

$WebApplicationUrl = http://dev.contoso.com

$snapin = Get-PSSnapin | Where-Object {$\_.Name -eq 'Microsoft.SharePoint.Powershell'}

if ($snapin -eq $null)

{

Write-Host "Loading SharePoint Powershell Snapin"

Add-PSSnapin "Microsoft.SharePoint.Powershell"

}

$webapp=Get-SPWebApplication $WebApplicationUrl

$newProxyLibrary = New-Object "Microsoft.SharePoint.Administration.SPClientCallableProxyLibrary"

$newProxyLibrary.AssemblyName = "Microsoft.Online.SharePoint.Dedicated.TenantAdmin.ServerStub, Version=15.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c"

$newProxyLibrary.SupportAppAuthentication = $true

$webapp.ClientCallableSettings.ProxyLibraries.Add($newProxyLibrary)

$webapp.Update()

Write-Host "Successfully added TenantAdmin ServerStub to ClientCallableProxyLibrary."

# Reset the memory of the web application

Write-Host "IISReset..."

Restart-Service W3SVC,WAS -force

Write-Host "IISReset complete."

1. Download and install the [SharePoint Online Management Shell](http://www.microsoft.com/en-us/download/details.aspx?id=35588). Install it on the computer where you do your remote development with CSOM. You’ll need to reference the Microsoft.Online.SharePoint.Client.Tenant.dll assembly, which is installed by default to the C:\Program Files\SharePoint Online Management Shell\Microsoft.Online.SharePoint.PowerShell folder.
2. Enable the **AdministrationSiteType** property in one site collection in your on-premises or Office 365 D farm. This site will serve as the tenant admin site during site collection creation. You’ll use this site to get client context. You could, for example, use the root site in the root site collection of the web application where you are going to create new site collections. This doesn’t have any other effect on your environment.  
   Note that if you need to provision host named site collections, this site must also be a host named site collection.  
   Use the following PowerShell script to make the site you choose the tenant admin site:  
     
   # Set the AdministrationSiteType property on a site collection of any site collection type.

# This site collection is used to get ClientContext for CSOM when creating site collections

# on on-premises and Dedicated sites

#

$siteColUrl = http://dev.contoso.com

$snapin = Get-PSSnapin | Where-Object {$\_.Name -eq 'Microsoft.SharePoint.Powershell'}

if ($snapin -eq $null)

{

Write-Host "Loading SharePoint Powershell Snapin"

Add-PSSnapin "Microsoft.SharePoint.Powershell"

}

$site = get-spsite -Identity $siteColUrl

$site.AdministrationSiteType = [Microsoft.SharePoint.SPAdministrationSiteType]::TenantAdministration

Once you’ve prepared your on-premises or Office 365 D farm for site collection creation with CSOM, you can use code that is nearly identical to the code that creates site collections remotely on Office 365 MT. The significant difference is that you get the SharePoint principal ID from the site collection that you’ve designated as your tenant admin site. The following sample assumes that you’ve assigned the role of tenant admin site to the root site collection of your farm, but you can use any site collection on the farm that want.

private string CreateSiteCollection(string hostWebUrl, string url, string template, string title, string adminAccount)

{

// Resolve root site collection URL from host web. We assume that this has been set as the "TenantAdminSite"

string rootSiteUrl = hostWebUrl.Substring(0, 8 + hostWebUrl.Substring(8).IndexOf("/"));

//Resolve URL for the new site collection

var webUrl = string.Format("{0}/sites/{1}", rootSiteUrl, url);

// This assumes that AdministrationSiteType as been set as TenantAdministration for root site collection

// If this tenant admin URI points to a site collection that is a host named site collection,

// this code will create a host named site collection as well

var tenantAdminUri = new Uri(rootSiteUrl);

string realm = TokenHelper.GetRealmFromTargetUrl(tenantAdminUri);

var token = TokenHelper.GetAppOnlyAccessToken(TokenHelper.SharePointPrincipal, tenantAdminUri.Authority, realm).AccessToken;

using (var adminContext = TokenHelper.GetClientContextWithAccessToken(tenantAdminUri.ToString(), token))

{

var tenant = new Tenant(adminContext);

var properties = new SiteCreationProperties()

{

Url = webUrl,

Owner = adminAccount,

Title = title,

Template = template

};

//Start the SPO operation to create the site

SpoOperation op = tenant.CreateSite(properties);

adminContext.Load(op, i => i.IsComplete);

adminContext.ExecuteQuery();

}

return webUrl;

}

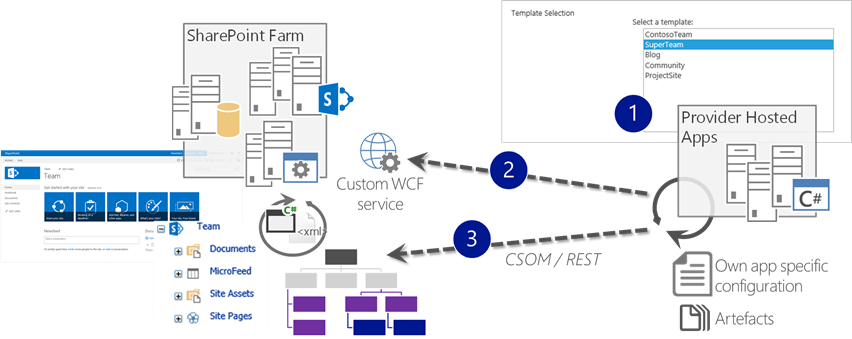
# What if the API I want to use to provision site branding isn't in CSOM?

In some cases, you may want to provision some element of site branding, but the method or property in question that exists in server-side code does not exist in CSOM. You can expose an otherwise unavailable capability as a custom extension point by deploying a custom Windows Communication Foundation (WCF) endpoint to the SharePoint farm. This functionality is demonstrated in the [ServicesSiteManager](http://code.msdn.microsoft.com/SharePoint-2013-Use-apps-9094e012) sample. Although the [ServicesSiteManager](http://code.msdn.microsoft.com/SharePoint-2013-Use-apps-9094e012) sample provisions a site collection, Office 365 Dedicated and on-premises customers should use the approach described in Section 4 of this document to provision site collections remotely. The WCF service pattern illustrated by the sample is still valid for cases where a capability in server-side code does not exist in CSOM.

While you are technically deploying additional server-side code to the farm, it has no long-term supportability impact--it simply exists to expose required SSOM APIs that aren't available in CSOM by default to a provider-hosted app that processes requests by calling specified parameters. This solution does not contain any customizations that need to be maintained in SharePoint.

**Important** For Office 365 D customers, Microsoft supports the WCF service pattern only when the CSOM code you need to build your solution is available in the server-side object model (SSOM), but not in CSOM. This restriction does not apply to on-prem deployments. Since the WCF service pattern requires server-side code, this workaround is not available in the SharePoint multi-tenant (MT). Want to see specific SSOM code exposed in CSOM? Submit your request to [User Voice](http://officespdev.uservoice.com/) and let us know.

**Figure 1. Deploying a custom WCF endpoint to a SharePoint farm**



Since the remote provisioning pattern enables you to provision a site using existing CSOM APIs in an on-premises SharePoint farm, you can easily replace functionality that the WCF endpoint solution enables once the functionality becomes available in the SharePoint CSOM. The exposed API set needs to be changed infrequently, so we recommend maintaining the code in the provider-hosted app for SharePoint, rather than in SharePoint.

The following example code is an excerpt from the [ServicesSiteManager](http://code.msdn.microsoft.com/SharePoint-2013-Use-apps-9094e012) sample. It is compatible only with [SharePoint Online Dedicated](http://technet.microsoft.com/en-us/library/dn151711.aspx).

using Microsoft.SharePoint.Client;

using System;

using System.Collections.Generic;

using System.Configuration;

using System.IO;

using System.Linq;

using System.ServiceModel;

using System.Web;

using System.Web.Hosting;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace Contoso.Services.SiteManager.AppWeb

{

public partial class Default : System.Web.UI.Page

{

protected void Page\_PreInit(object sender, EventArgs e)

{

Uri redirectUrl;

switch (SharePointContextProvider.CheckRedirectionStatus(Context, out redirectUrl))

{

case RedirectionStatus.Ok:

return;

case RedirectionStatus.ShouldRedirect:

Response.Redirect(redirectUrl.AbsoluteUri, endResponse: true);

break;

case RedirectionStatus.CanNotRedirect:

Response.Write("An error occurred while processing your request.");

Response.End();

break;

}

}

protected void Page\_Load(object sender, EventArgs e)

{

// Set the URL presentation.

lblHostUrl.Text = ConfigurationManager.AppSettings["WebApplicationUrl"];

// Add options to the template listing.

listSites.Items.Add(new System.Web.UI.WebControls.ListItem("Contoso Team", "STS#0"));

listSites.Items.Add(new System.Web.UI.WebControls.ListItem("Contoso Blog", "BLOG#0"));

listSites.Items.Add(new System.Web.UI.WebControls.ListItem("Contoso Community", "COMMUNITY#0"));

}

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

{

// Create the site collection using impersonation.

SiteManager.SiteManagerClient managerClient = GetSiteManagerClient();

SiteManager.SiteData newSite = new SiteManager.SiteData()

{

Description = txtDescription.Text,

LcId = "1033",

OwnerLogin = txtAdminPrimary.Text,

SecondaryContactLogin = txtAdminSecondary.Text,

Title = txtTitle.Text,

Url = string.Format("sites/" + txtUrl.Text),

WebTemplate = listSites.SelectedValue

};

// Create the site collection by calling the WCF endpoint in the SharePoint farm.

string url = managerClient.CreateSiteCollection(newSite);

// Use app identity to brand the newly created site collection properly.

// Use app identity since we don't know if the requestor account

// has permissions to just created site collection.

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

// Connect the site collection just created and upload the branding.

using (var ctx = CreateAppOnlyClientContextForUrl(spContext, url))

{

// Deploy the theme to the web, so that we can set it for the site.

Web web = ctx.Web;

ctx.Load(web);

ctx.ExecuteQuery();

DeployThemeToWeb(ctx, web);

//Set the properties for applying custom theme that was just uploaded.

string spColorURL = URLCombine(web.ServerRelativeUrl, "/\_catalogs/theme/15/contoso.spcolor");

string spFontURL = URLCombine(web.ServerRelativeUrl, "/\_catalogs/theme/15/contoso.spfont");

string backGroundImage = URLCombine(web.ServerRelativeUrl, "/\_catalogs/theme/15/contosobg.jpg");

// Use the Red theme for demonstration

web.ApplyTheme(spColorURL,

spFontURL,

backGroundImage,

false);

ctx.ExecuteQuery();

// Redirect to just created site

Response.Redirect(url);

}

}

/// <summary>

/// Creates context to the location provided as a URL.

/// </summary>

/// <param name="spContext"></param>

/// <param name="url"></param>

/// <returns></returns>

private ClientContext CreateAppOnlyClientContextForUrl(SharePointContext spContext, string url)

{

return TokenHelper.GetClientContextWithAccessToken(url, spContext.AppOnlyAccessTokenForSPHost);

}

/// <summary>

/// Dynamically configures from the WCF end point using code.

/// </summary>

/// <returns>Needed proxy client with impersonation information.</returns>

private SiteManager.SiteManagerClient GetSiteManagerClient()

{

BasicHttpBinding binding = new BasicHttpBinding();

if (ConfigurationManager.AppSettings["WebApplicationUrl"].Contains("https://"))

{

binding.Security.Mode = BasicHttpSecurityMode.Transport;

}

else

{

binding.Security.Mode = BasicHttpSecurityMode.TransportCredentialOnly;

}

binding.Security.Transport.ClientCredentialType = HttpClientCredentialType.Ntlm;

EndpointAddress endPoint = new EndpointAddress(ConfigurationManager.AppSettings["WebApplicationUrl"] + "/\_vti\_bin/contoso.services.sitemanager/sitemanager.svc");

//Sets time outs, since site collection creation could take a while.

//Time out is also set on server side.

binding.ReceiveTimeout = TimeSpan.FromMinutes(15);

binding.CloseTimeout = TimeSpan.FromMinutes(15);

binding.OpenTimeout = TimeSpan.FromMinutes(15);

binding.SendTimeout = TimeSpan.FromMinutes(15);

//Create proxy instance.

SiteManager.SiteManagerClient managerClient = new SiteManager.SiteManagerClient(binding, endPoint);

managerClient.ClientCredentials.Windows.AllowedImpersonationLevel = System.Security.Principal.TokenImpersonationLevel.Impersonation;

// Sets impersonation information.

var impersonator = new System.Net.NetworkCredential(ConfigurationManager.AppSettings["Service\_UserId"],

ConfigurationManager.AppSettings["Service\_Pwd"],

ConfigurationManager.AppSettings["Service\_Domain"]);

managerClient.ClientCredentials.Windows.ClientCredential = impersonator;

return managerClient;

}

#region THEME DEPLOYMENT

/// <summary>

/// Deploys theme files to the web and creates the theme option.

/// </summary>

/// <param name="clientContext"></param>

/// <param name="web"></param>

private void DeployThemeToWeb(ClientContext clientContext, Web web)

{

// Deploys files one by one to the proper location.

DeployFileToThemeFolderSite(clientContext, web, "DeploymentFiles/Theme/Contoso.spcolor");

DeployFileToThemeFolderSite(clientContext, web, "DeploymentFiles/Theme/Contoso.spfont");

DeployFileToThemeFolderSite(clientContext, web, "DeploymentFiles/Theme/contosobg.jpg");

// Adds entries to the Theme catalog. This is optional, but adds visibility

// for the theme option, if the theme is change manually.

AddNewThemeOptionToSite(clientContext, web);

}

/// <summary>

/// Deploys the specified file to the theme folder.

/// </summary>

/// <param name="clientContext"></param>

/// <param name="web"></param>

/// <param name="sourceAddress"></param>

private void DeployFileToThemeFolderSite(ClientContext clientContext, Web web, string sourceAddress)

{

// Gets the path to the file about to be deployed.

string file = HostingEnvironment.MapPath(string.Format("~/{0}", sourceAddress));

List themesList = web.GetCatalog(123);

// get the theme list

clientContext.Load(themesList);

clientContext.ExecuteQuery();

Folder rootfolder = themesList.RootFolder;

clientContext.Load(rootfolder);

clientContext.Load(rootfolder.Folders);

clientContext.ExecuteQuery();

Folder folder15 = rootfolder;

foreach (Folder folder in rootfolder.Folders)

{

if (folder.Name == "15")

{

folder15 = folder;

break;

}

}

// Uses CSOM to upload the file into.

FileCreationInformation newFile = new FileCreationInformation();

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

newFile.Url = folder15.ServerRelativeUrl + "/" + Path.GetFileName(sourceAddress);

newFile.Overwrite = true;

Microsoft.SharePoint.Client.File uploadFile = folder15.Files.Add(newFile);

clientContext.Load(uploadFile);

clientContext.ExecuteQuery();

}

/// <summary>

/// Creates new options in the Look and Feel section.

/// </summary>

/// <param name="clientContext"></param>

/// <param name="web"></param>

private void AddNewThemeOptionToSite(ClientContext clientContext, Web web)

{

// Gets instances to the composite look gallery.

List themesOverviewList = web.GetCatalog(124);

clientContext.Load(themesOverviewList);

clientContext.ExecuteQuery();

// Is the item already in the list?

if (!ContosoThemeEntryExists(clientContext, web, themesOverviewList))

{

// Create a new theme entry. Theme selection is not available from the UI

// in OneDrive for Business sites, so this code is only to maintain consistency.

ListItemCreationInformation itemInfo = new ListItemCreationInformation();

Microsoft.SharePoint.Client.ListItem item = themesOverviewList.AddItem(itemInfo);

item["Name"] = "Contoso";

item["Title"] = "Contoso";

item["ThemeUrl"] = URLCombine(web.ServerRelativeUrl, "/\_catalogs/theme/15/contoso.spcolor"); ;

item["FontSchemeUrl"] = URLCombine(web.ServerRelativeUrl, "/\_catalogs/theme/15/contoso.spfont"); ;

item["ImageUrl"] = URLCombine(web.ServerRelativeUrl, "/\_catalogs/theme/15/contosobg.jpg");

// This code uses the out-of-the-box .master.

item["MasterPageUrl"] = URLCombine(web.ServerRelativeUrl, "/\_catalogs/masterpage/seattle.master");

item["DisplayOrder"] = 0;

item.Update();

clientContext.ExecuteQuery();

}

}

/// <summary>

/// Verifies whether the theme option already exists on the site.

/// </summary>

/// <param name="clientContext"></param>

/// <param name="web"></param>

/// <param name="themeList"></param>

/// <returns></returns>

private bool ContosoThemeEntryExists(ClientContext clientContext, Web web, List themeList)

{

CamlQuery query = new CamlQuery();

query.ViewXml = @"

<View>

<Query>

<Where>

<Eq>

<FieldRef Name='Name' />

<Value Type='Text'>Contoso</Value>

</Eq>

</Where>

</Query>

</View>";

var found = themeList.GetItems(query);

clientContext.Load(found);

clientContext.ExecuteQuery();

if (found.Count > 0)

{

return true;

}

return false;

}

private string URLCombine(string baseUrl, string relativeUrl)

{

if (baseUrl.Length == 0)

return relativeUrl;

if (relativeUrl.Length == 0)

return baseUrl;

return string.Format("{0}/{1}", baseUrl.TrimEnd(new char[] { '/', '\\' }), relativeUrl.TrimStart(new char[] { '/', '\\' }));

}

#endregion

}

}

# Additional samples for provisioning and branding

This section describes and links to samples that demonstrate best practices for addressing the following ten scenarios for provisioning and branding sites by a variety of applications: provider-hosted apps, console applications, and, in the last example, a sandboxed solution.

* Site classification
* Refresh branding of existing sites and subsites
* Modify host web lists at list creation time
* Modify SharePoint Designer settings using CSOM
* Create content types using CSOM
* Create document content types using CSOM
* Modify site permissions and external users
* Manage Users and Groups
* Yammer group creation and replacement of existing feed
* Prevent site deletion

## [Site classification](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/Core.SiteClassification)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample demonstrates how to implement a site classification solution, as well as leverage site policies. | **Typical scenarios**:   * Identify sensitive data * Eliminate excessive site proliferation * Enforce SharePoint Site Policies * Eliminate outdated and irrelevant search results | Site categories are first created. Then a console application adds a new custom action for site classification, and a web application is used to display the categories created. |

**Related samples**:

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

**Alternative approaches**:

* Use extension method **Web.AddindexedPropertyBagKey** in **Office365PnP.Cor**e to store the classification values in site property bags instead of a list. The method enables property bags to be crawled or searchable.

**Description of the sample:**

Overview

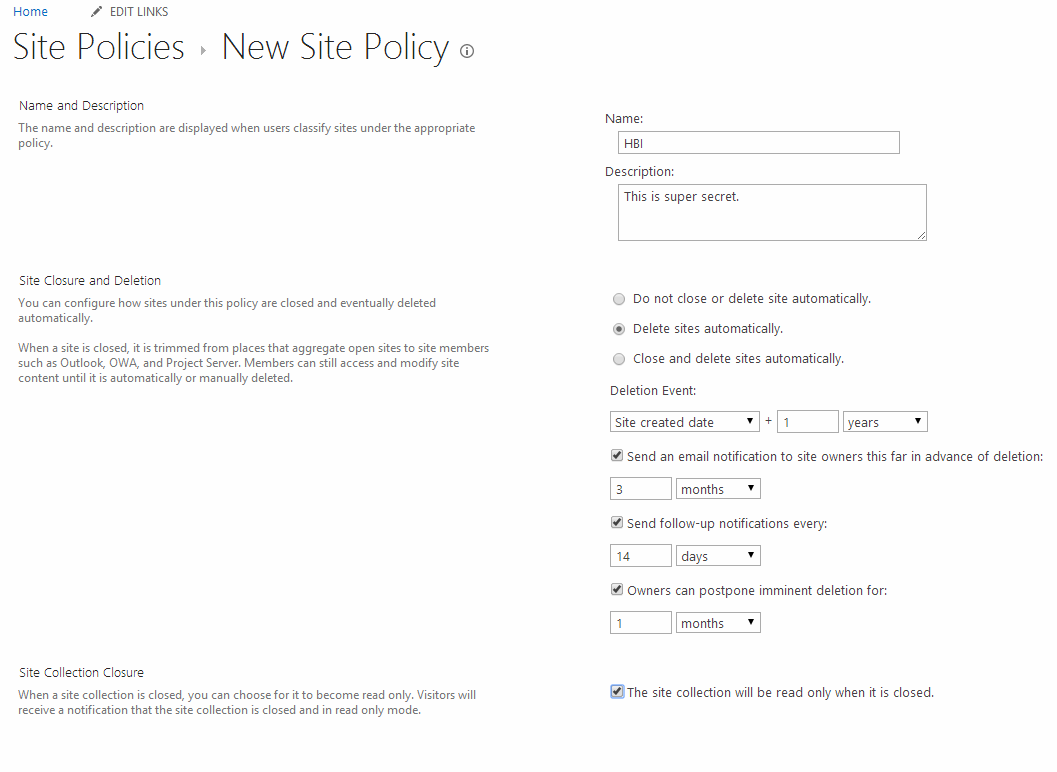
Even with good governance, SharePoint sites can proliferate and grow out of control. Sites are created as they are needed, but sites are rarely deleted. Many organizations have search crawl burdened by unused site collections, difficulty with outdated and irrelevant results. Site classification allows sensitive data in the environment to be identified. This scenario demonstrates how to implement a site classification solution that leverages many of the existing Solutions Pack samples, as well as use SharePoint Site Policies to enforce deletion. This solution can also be integrated into your existing site provisioning solution to address your governance needs.

Setup

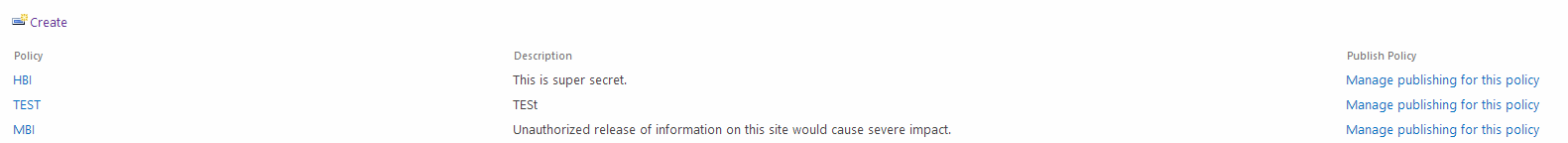
The first step is to define the site policies that will be available in all your site collections. This sample defines and sets site policies in the Content Type Hub using SharePoint Online MT, but the same approach is available in SharePoint Online Dedicated as well as SharePoint on-premises. If your environment is hosted in SharePoint Online MT, your content type hub would be located at the following URL: <https://[tenanatname]/sites/contentTypeHub>. Navigate to Settings, then Site Policies under Site Collection Administration, and then finally create.

Note: See Overview of site policies in SharePoint 2013 at <http://technet.microsoft.com/en-US/library/jj219569(v=office.15).aspx> for an overview of Site Policies.

Create three site policies: HBI, MBI and LBI. For example, enter the HBI policy as follows:



Repeat the above setup two more times for the MBI and LBI policies. You should now have the following:



The next step is to publish the three policies.

Scenario 1: Insert a custom action

Here we are going to add the custom action to the Settings page and the SharePoint gear. That is only available to users with **ManageWeb** Permission.

|  |  |
| --- | --- |
|  |  |

/// <summary>

/// Adds a custom Action to a Site Collection

/// </summary>

/// <param name="ctx">The Authenticaed client context.</param>

/// <param name="hostUrl">The Provider hosted URL for the Application</param>

static void AddCustomAction(ClientContext ctx, string hostUrl)

{

var \_web = ctx.Web;

ctx.Load(\_web);

ctx.ExecuteQuery();

//we only want the action to show up if you have manage web permissions

BasePermissions \_manageWebPermission = new BasePermissions();

\_manageWebPermission.Set(PermissionKind.ManageWeb);

CustomActionEntity \_entity = new CustomActionEntity()

{

Group = "SiteTasks",

Location = "Microsoft.SharePoint.SiteSettings",

Title = "Site Classification",

Sequence = 1000,

Url = string.Format(hostUrl, ctx.Url),

Rights = \_manageWebPermission,

};

CustomActionEntity \_siteActionSC = new CustomActionEntity()

{

Group = "SiteActions",

Location = "Microsoft.SharePoint.StandardMenu",

Title = "Site Classification",

Sequence = 1000,

Url = string.Format(hostUrl, ctx.Url),

Rights = \_manageWebPermission

};

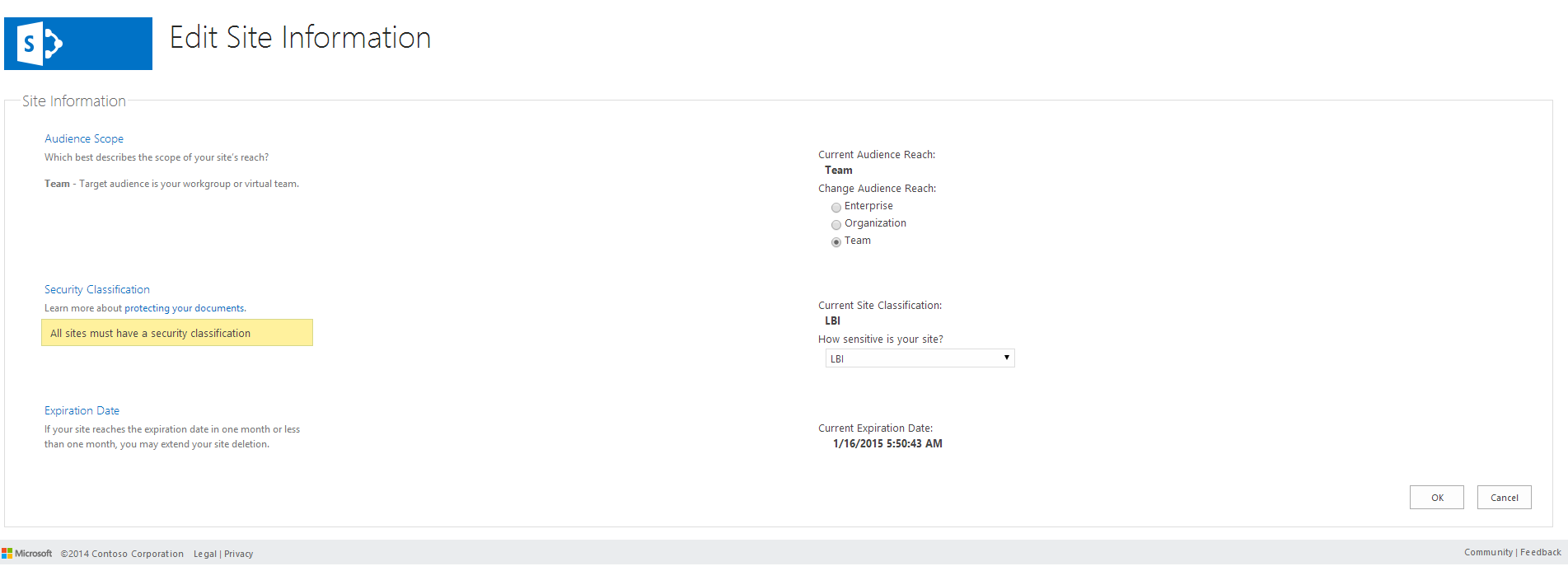
\_web.AddCustomAction(\_entity);

\_web.AddCustomAction(\_siteActionSC);

}

See [here](http://msdn.microsoft.com/en-us/library/office/bb802730(v=office.15).aspx) for more information on the custom action settings.

Scenario 2: Custom site classification



The preceding page defines the necessary options that are available. Here we define the intended **Audience** **Reach** of the Site as well as the defined **Site Policy**. We also show the **Expiration Date** of the site, which is based on the site policy that you created earlier. Both the Audience Reach and **Site Classification** are searchable and will have managed properties associated after a crawl has taken place. You can then use these properties to search for specific types of sites.



These are searchable via a custom hidden list that is implemented in the Site Collection. This is implemented in the **Core.SiteClassification.Common** project in the **SiteManagerImpl** class.

private void CreateSiteClassificationList(ClientContext ctx)

{

var \_newList = new ListCreationInformation()

{

Title = SiteClassificationList.SiteClassificationListTitle,

Description = SiteClassificationList.SiteClassificationDesc,

TemplateType = (int)ListTemplateType.GenericList,

Url = SiteClassificationList.SiteClassificationUrl,

QuickLaunchOption = QuickLaunchOptions.Off

};

if(!ctx.Web.ContentTypeExistsById(SiteClassificationContentType.SITEINFORMATION\_CT\_ID))

{

//ct

ContentType \_contentType = ctx.Web.CreateContentType(SiteClassificationContentType.SITEINFORMATION\_CT\_NAME,

SiteClassificationContentType.SITEINFORMATION\_CT\_DESC,

SiteClassificationContentType.SITEINFORMATION\_CT\_ID,

SiteClassificationContentType.SITEINFORMATION\_CT\_GROUP);

FieldLink \_titleFieldLink = \_contentType.FieldLinks.GetById(new Guid("fa564e0f-0c70-4ab9-b863-0177e6ddd247"));

\_titleFieldLink.Required = false;

\_contentType.Update(false);

//Key Field

ctx.Web.CreateField(SiteClassificationFields.FLD\_KEY\_ID,

SiteClassificationFields.FLD\_KEY\_INTERNAL\_NAME,

FieldType.Text,

SiteClassificationFields.FLD\_KEY\_DISPLAY\_NAME,

SiteClassificationFields.FIELDS\_GROUPNAME);

//value field

ctx.Web.CreateField(SiteClassificationFields.FLD\_VALUE\_ID,

SiteClassificationFields.FLD\_VALUE\_INTERNAL\_NAME,

FieldType.Text,

SiteClassificationFields.FLD\_VALUE\_DISPLAY\_NAME,

SiteClassificationFields.FIELDS\_GROUPNAME);

//Add Key Field to content type

ctx.Web.AddFieldToContentTypeById(SiteClassificationContentType.SITEINFORMATION\_CT\_ID,

SiteClassificationFields.FLD\_KEY\_ID.ToString(),

true);

//Add Value Field to content type

ctx.Web.AddFieldToContentTypeById(SiteClassificationContentType.SITEINFORMATION\_CT\_ID,

SiteClassificationFields.FLD\_VALUE\_ID.ToString(),

true);

}

var \_list = ctx.Web.Lists.Add(\_newList);

\_list.Hidden = true;

\_list.ContentTypesEnabled = true;

\_list.Update();

ctx.Web.AddContentTypeToListById(SiteClassificationList.SiteClassificationListTitle, SiteClassificationContentType.SITEINFORMATION\_CT\_ID, true);

this.CreateCustomPropertiesInList(\_list);

ctx.ExecuteQuery();

this.RemoveFromQuickLaunch(ctx, SiteClassificationList.SiteClassificationListTitle);

}

By default, when you create a list either using out-of-box or if you are using CSOM, the list will be available in the **Recent** menu. Now, this is not desired -- it’s supposed to be hidden. We need some simple code to remove the item from the **Recent** menu:

Note: For more information, see: <http://blogs.technet.com/b/speschka/archive/2014/05/07/create-a-list-in-the-host-web-when-your-sharepoint-app-is-installed-and-remove-it-from-the-recent-stuff-list.aspx>

private void RemoveFromQuickLaunch(ClientContext ctx, string listName)

{

Site \_site = ctx.Site;

Web \_web = \_site.RootWeb;

ctx.Load(\_web, x => x.Navigation, x => x.Navigation.QuickLaunch);

ctx.ExecuteQuery();

var \_vNode = from NavigationNode \_navNode in \_web.Navigation.QuickLaunch

where \_navNode.Title == "Recent"

select \_navNode;

NavigationNode \_nNode = \_vNode.First<NavigationNode>();

ctx.Load(\_nNode.Children);

ctx.ExecuteQuery();

var vcNode = from NavigationNode cn in \_nNode.Children

where cn.Title == listName

select cn;

NavigationNode \_cNode = vcNode.First<NavigationNode>();

\_cNode.DeleteObject();

ctx.ExecuteQuery();

}

The sample provides for the possibility that a Site Administrator or someone with permission can remove the new list. When this page is accessed the list is re-created; however, the sample doesn’t set the properties back. If the values in the list are not present, we know that someone deleted the list and you can leverage the Office 365 PnP sample **Core.SiteEnumeration** to do checks on the list and send nasty emails to your site administrators. You may also extend this sample and modify the permissions on the list so that only site collections administrators have access.

The list verification check is also implemented in the **SiteManagerImpl** class, in the **Initialize** member:

internal void Initialize(ClientContext ctx)

{

try {

var \_web = ctx.Web;

var lists = \_web.Lists;

ctx.Load(\_web);

ctx.Load(lists, lc => lc.Where(l => l.Title == SiteClassificationList.SiteClassificationListTitle));

ctx.ExecuteQuery();

if (lists.Count == 0) {

this.CreateSiteClassificationList(ctx);

}

}

catch(Exception \_ex)

{

}

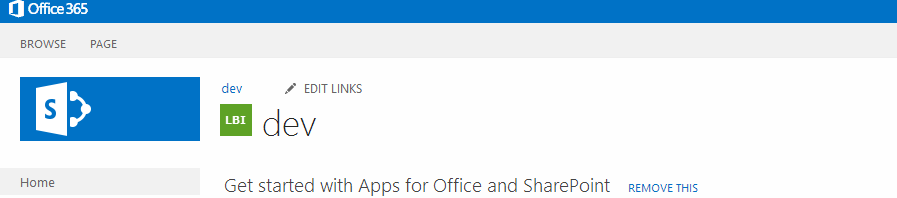
}

}

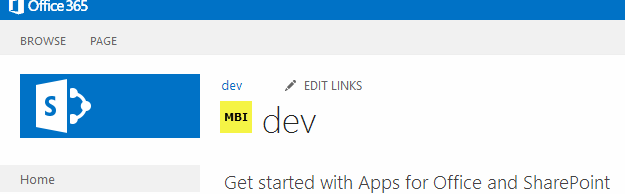
Scenario 3: Site classification display

In the final scenario, an indicator is added to show site classification. Here, the sample injects an image next to the **Site Title**. In older versions this was done using a Delegate control or a custom master page along with JavaScript. Delegate controls are Server Side implemented, so those are avoided here. Modifying the master page would work, but this sample uses a JavaScript injection pattern. When you change the **Site Policy** in the **Edit Site Information** page, this will change the site indicator, as in the following:

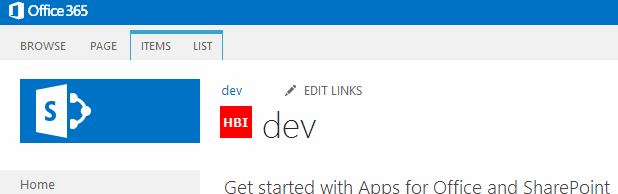
**LBI**



**MBI**



**HBI**



The following method is defined in the **Core.SiteClassificationWeb** project, scripts and classifier.js. I chose to store the images in an Azure Web Site, so you will have to change the hard-coded URLs to match your environment.

function setClassifier() {

if (!classified)

{

var clientContext = SP.ClientContext.get\_current();

var query = "<View><Query><Where><Eq><FieldRef Name='SC\_METADATA\_KEY'/><Value Type='Text'>sc\_BusinessImpact</Value></Eq></Where></Query><ViewFields><FieldRef Name='ID'/><FieldRef Name='SC\_METADATA\_KEY'/><FieldRef Name='SC\_METADATA\_VALUE'/></ViewFields></View>";

var list = clientContext.get\_web().get\_lists().getByTitle("Site Information");

clientContext.load(list);

var camlQuery = new SP.CamlQuery();

camlQuery.set\_viewXml(query);

var listItems = list.getItems(camlQuery);

clientContext.load(listItems);

clientContext.executeQueryAsync(Function.createDelegate(this, function (sender, args) {

var listItemInfo;

var listItemEnumerator = listItems.getEnumerator();

while (listItemEnumerator.moveNext()) {

listItemInfo = listItemEnumerator.get\_current().get\_item('SC\_METADATA\_VALUE');

var pageTitle = $('#pageTitle')[0].innerHTML;

if (pageTitle.indexOf("img") > -1) {

classified = true;

}

else {

var siteClassification = listItemInfo;

if (siteClassification == "HBI") {

var img = $("<a href='http://insertyourpolicy' target=\_blank><img id=classifer name=classifer src='https://spmanaged.azurewebsites.net/content/img/hbi.png' title='Site contains personally identifiable information (PII), or unauthorized release of information on this site would cause severe or catastrophic loss to Contoso.' alt='Site contains personally identifiable information (PII), or unauthorized release of information on this site would cause severe or catastrophic loss to Contoso.'></a>");

$('#pageTitle').prepend(img);

classified = true;

}

else if (siteClassification == "MBI") {

var img = $("<a href='http://insertyourpolicy' target=\_blank><img id=classifer name=classifer src='https://spmanaged.azurewebsites.net/content/img/mbi.png' title='Unauthorized release of information on this site would cause severe impact to Contoso.' alt='Unauthorized release of information on this site would cause severe impact to Contoso.'></a>");

$('#pageTitle').prepend(img);

classified = true;

}

else if (siteClassification == "LBI") {

var img = $("<a href='http://insertyourpolicy' target=\_blank><img id=classifer name=classifer src='https://spmanaged.azurewebsites.net/content/img/lbi.png' title='Limited or no impact to Contoso if publically released.' alt='Limited or no impact to Contoso if publically released.'></a>");

$('#pageTitle').prepend(img);

classified = true;

}

}

}

}));

}

Dependencies

* Microsoft.SharePoint.Client.dll
* Microsoft.SharePoint.Client.Runtime.dll
* Microsoft.Office.Client.Policy
* OfficeAMS.Core

## [Refresh branding of existing sites and subsites](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/Branding.Refresh)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This scenario shows how you can refresh the branding of existing site collections and sites. | This sample provides a convenient way to identify and update the branding on existing sites. A typical scenario is when moving to a new version. | This console application first fetches a list of sites using search. Then, the application uses the web property bag to determine particular theme changes. |

**Related samples**:

**Alternative approaches**:

* Use a remote timer job pattern to refresh branding. See [Core.SimpleTimerJob](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.SimpleTimerJob).

**Description of the sample:**

Overview

This scenario uses the Office 365 PnP Core Library to iterate over existing sites and their subsites with a purpose to verify and update the applied branding. The sample shows how to upgrade the site branding, but the same concept can be used for other purposes – such as deploying a new library to a list of sites, or to upgrade a custom action that was deployed at provisioning time. The scenario may be used for any operation in which existing sites need to be moved to a newer version.

Step 1: Getting the sites you want to operate on

First you need to acquire a list of sites and/or subsites to perform changes against. The sample shows how to do this via search, but other options to fetch this list are either reading from a site directory, or providing a management UI where administrators can specify the list. Here is the code using search:

// Get a list of sites: search is one way to obtain this list, alternative can be a site directory

List<SiteEntity> sites = cc.Web.SiteSearchScopedByUrl("https://bertonline.sharepoint.com");

// Generic settings (apply changes on all webs or just root web

bool applyChangesToAllWebs = true;

// Optionally further refine the list of returned site collections

var filteredSites = from p in sites

where p.Url.Contains("13003")

select p;

List<SiteEntity> sitesAndSubSites = new List<SiteEntity>();

if (applyChangesToAllWebs)

{

// we want to update all webs, so the list of sites is extended with all subsites

foreach (SiteEntity site in filteredSites)

{

sitesAndSubSites.Add(new SiteEntity() { Url = site.Url,

Title = site.Title,

Template = site.Template });

GetSubSites(cc, site.Url, ref sitesAndSubSites);

}

sites = sitesAndSubSites;

}

The call to GetSubSites is a recursive call so that the complete subsite tree is fetched.

Note: Please note the amount of sites selected and verify they are correct before continuing.

Step 2: Upgrade the branding of the existing sites

Once a site has been selected for processing you can leverage Office 365 PnP Core methods to easily manipulate the site. The sample shows how this is done for branding, but any type of change can be processed in this manner.

In order to speed up the code, the sample uses a pattern that leverages the web property bag to store information about the current settings. The code first reads the web property bag values and takes the appropriate action based on each value:

// Check if we've a property bag entry

string themeName = cc.Web.GetPropertyBagValueString(BRANDING\_THEME, "");

if (!String.IsNullOrEmpty(themeName))

{

// No theme property bag entry, assume no theme has been applied

if (themeName.Equals(currentThemeName, StringComparison.InvariantCultureIgnoreCase))

{

// the used theme matches to the theme we want to update

int? brandingVersion = cc.Web.GetPropertyBagValueInt(BRANDING\_VERSION, 0);

if (brandingVersion < currentBrandingVersion)

{

DeployTheme(cc, currentThemeName);

// Set the web propertybag entries

cc.Web.SetPropertyBagValue(BRANDING\_THEME, currentThemeName);

cc.Web.SetPropertyBagValue(BRANDING\_VERSION, currentBrandingVersion);

}

}

else

{

if (forceBranding)

{

DeployTheme(cc, currentThemeName);

// Set the web propertybag entries

cc.Web.SetPropertyBagValue(BRANDING\_THEME, currentThemeName);

cc.Web.SetPropertyBagValue(BRANDING\_VERSION, currentBrandingVersion);

}

}

}

The code that actually updates the theme is pretty straightforward and based on Office 365 PnP Core methods:

string themeRoot = Path.Combine(AppRootPath, String.Format(@"Themes\{0}", themeName));

string spColorFile = Path.Combine(themeRoot, string.Format("{0}.spcolor", themeName));

string spFontFile = Path.Combine(themeRoot, string.Format("{0}.spfont", themeName));

string backgroundFile = Path.Combine(themeRoot, string.Format("{0}bg.jpg", themeName));

string logoFile = Path.Combine(themeRoot, string.Format("{0}logo.png", themeName));

if (IsThisASubSite(cc.Url))

{

// Retrieve the context of the root site of the site collection

using (ClientContext ccParent = new ClientContext(GetRootSite(cc.Url)))

{

ccParent.Credentials = cc.Credentials;

cc.Web.DeployThemeToSubWeb(ccParent.Web, themeName, spColorFile, spFontFile, backgroundFile, "");

cc.Web.SetThemeToSubWeb(ccParent.Web, themeName);

}

}

else

{

cc.Web.DeployThemeToWeb(themeName, spColorFile, spFontFile, backgroundFile, "");

cc.Web.SetThemeToWeb(themeName);

}

## [Modify host web lists at list creation time](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/Core.EventReceiversBasedModifications)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This scenario shows how you can modify a list created in the host web at list creation time. | This sample provides a way to modify a list at creation time for many uses, including adding a content type or enabling versioning. | This SharePoint Provider-hosted app handles the AppInstalled and AppUninstalling events to create a ListAdded event receiver. |

**Related samples**:

* [SharePoint 2013: Attach remote event receivers to lists in the host web](https://code.msdn.microsoft.com/office/SharePoint-2013-Attach-7cdd0f26)

**Alternative approaches**:

**Description of the sample:**

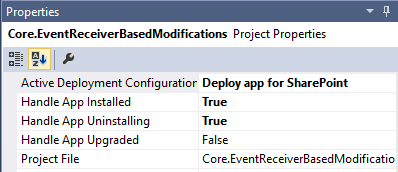
Overview

This example shows how you can modify lists when they are created in the host web. Whenever a user creates a new list in the host web a **ListAdded** remote event receiver gets fired and in this remote event receiver you can modify the list to your needs. Typical changes would be enabling versioning or adding a content type to the list, but in reality anything that can be done via CSOM can be used.

As we’re hooking up the list to the host web we’ll need to programmatically hookup the event receiver. In the sample we’ve opted to do this as part of an app: when the app is installed an **AppInstalled** event will fire and we’ll use this event to hookup the ListAdded event.

Ensuring that the AppInstalled and AppUninstalling events fire

The app events are set as properties of the SharePoint project:



Hooking up the ListAdded event

The remote event receiver that gets executed on app install will dynamically add the ListAdded event receiver. Below code snippet shows how this is done:

bool rerExists = false;

cc.Load(cc.Web.EventReceivers);

cc.ExecuteQuery();

foreach (var rer in cc.Web.EventReceivers)

{

if (rer.ReceiverName == RECEIVER\_NAME)

{

rerExists = true;

}

}

if (!rerExists)

{

EventReceiverDefinitionCreationInformation receiver = new EventReceiverDefinitionCreationInformation();

receiver.EventType = EventReceiverType.ListAdded;

//Get WCF URL where this message was handled

OperationContext op = OperationContext.Current;

Message msg = op.RequestContext.RequestMessage;

receiver.ReceiverUrl = msg.Headers.To.ToString();

receiver.ReceiverName = RECEIVER\_NAME;

receiver.Synchronization = EventReceiverSynchronization.Synchronous;

cc.Web.EventReceivers.Add(receiver);

cc.ExecuteQuery();

}

Customizing the added lists

When the **ListAdded** event handler is firing then the following code is executed. This is just a simple sample that leverages Office 365 PnP Core methods to set versioning, but in reality you can do any kind of manipulation that you might need.

private void HandleListAdded(SPRemoteEventProperties properties)

{

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

{

if (cc != null)

{

try

{

if (properties.ListEventProperties.TemplateId == (int)ListTemplateType.DocumentLibrary)

{

//set versioning

cc.Web.GetListByTitle(properties.ListEventProperties.ListTitle).UpdateListVersioning(true, true);

}

}

catch (Exception ex)

{

System.Diagnostics.Trace.WriteLine(ex.Message);

}

}

}

}

}

Dealing with Uninstall

When the app is uninstalled we’re also removing the event receiver. In order to make this work during debugging you’ll need to ensure that you navigate to the “Apps in testing” library and use the remove option on the app. This remove will trigger the app uninstalling event with the proper permissions to remove the created remote event handler. If you just close the browser or uninstall the app from the “site contents” then either the event receiver never fires or the **AppUninstalling** event receiver runs with insufficient permissions to remove the **ListAdded** event receiver. The reason for this behavior is differences in app deployment when the app gets side loaded which is what Visual Studio does when you press F5.

Note: If you’ve been experimenting a lot it often helps to test this sample in a clean developer site.

## [Modify SharePoint Designer settings using CSOM](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.SPD)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample shows how modify SharePoint Designer Settings using CSOM. | This sample provides an offline method to modify SharePoint Designer settings as well as enable or disable SPD. | This console application programmatically modifies the SPD settings using CSOM. |

**Related samples**:

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

**Alternative approaches**:

**Description of the sample:**

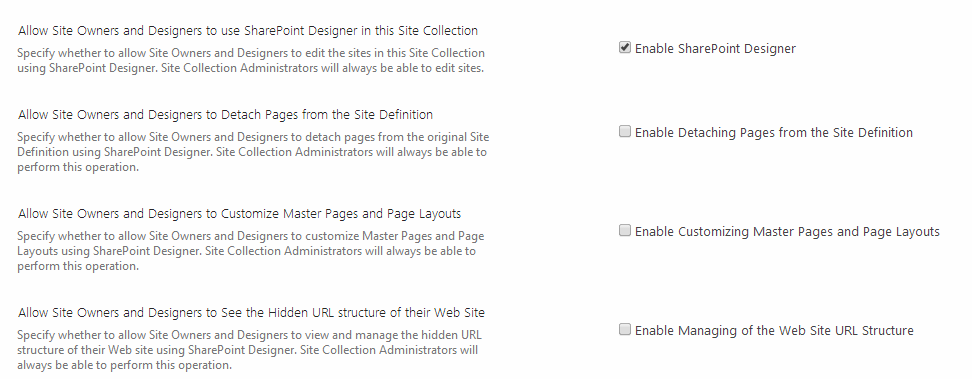
Overview

Maybe you have been getting complaints lately about the ability to use SharePoint Designer (SPD) in an enterprise SharePoint environment or your governance policy prohibits the use of SPD in your SharePoint environment either on-premises or in the Cloud. This sample shows a pattern for disabling the SharePoint Designer Settings using CSOM. Based on this sample, you can then implement a timer job-like solution that would apply these settings. See Office 365 PnP sample **Core.SimpleTimerJob** for additional information.

Scenario 1: Console application

This scenario is implemented as a console application that programmatically sets the SharePoint Designer settings using CSOM for your targeted site collection.

Note: Site Collection administrators can still enable this setting.



Disable Designer

/// <summary>

/// This method will disable SharePoint designer you must be a site collection administrator to perform this action

/// A UnauthorizedAccessException is thrown when attempting to set the property if either the user is not a Site Collection administrator or the setting is disabled at the

/// web application level.

/// Site Collection Administrators will always be able to edit sites.

/// </summary>

/// <param name="ctx"></param>

public static void DisableDesigner(ClientContext ctx)

{

try

{

Site \_site = ctx.Site;

ctx.Load(\_site);

//Allow Site Owners and Designers to use SharePoint Designer in this Site Collection

\_site.AllowDesigner = false;

//Allow Site Owners and Designers to Customize Master Pages and Page Layouts

\_site.AllowMasterPageEditing = false;

//Allow Site Owners and Designers to Detach Pages from the Site Definition

\_site.AllowRevertFromTemplate = false;

//Allow Site Owners and Designers to See the Hidden URL structure of their Web Site

\_site.ShowUrlStructure = false;

ctx.ExecuteQuery();

}

catch

{

throw;

}

}

}

Enable Designer

/// <summary>

/// This method will Enable SharePoint designer you must be a site collection administrator to perform this action

/// A UnauthorizedAccessException is thrown when attempting to set the property if either the user is not a Site Collection administrator or the setting is disabled at the

/// web application level.

/// Site Collection Administrators will always be able to edit sites.

/// </summary>

/// <param name="ctx"></param>

public static void EnableDesigner(ClientContext ctx)

{

try

{

Site \_site = ctx.Site;

ctx.Load(\_site);

\_site.AllowDesigner = true;

\_site.AllowMasterPageEditing = true;

\_site.AllowRevertFromTemplate = true;

\_site.ShowUrlStructure = true;

ctx.ExecuteQuery();

}

catch

{

throw;

}

}

Scenario 2: Hide “Designer settings”

Formerly, we would use Full Trust code and the **HideCustomAction** element to remove the “SharePoint Designer Settings” on the settings page. How can this be achieved using the Cloud App Model? We use the JavaScript injection pattern to hide the item. Just like using the **HideCustomAction** approach, a user with the appropriate permissions can still navigate to the page and change the settings.

Note: See Office 365 PnP Sample, Core.JavaScriptInjection for additional information.

|  |  |
| --- | --- |
| Before | After |
|  |  |

## [Create content types using CSOM](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.CreateContentTypes)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample shows how create content types using CSOM, and make localization changes using features introduced in SP2013 SP1. | This sample shows how site columns and content types, and their linkage, can be programmatically created. Also, the sample leverages new SP2013 SP1 CSOM APIs to add a specific content type identifier, as well as demonsrates how to localize content types, lists, and site titles. | This console application uses an inherited content type identifier to create a new content type on the O365 platform with all its existing properties. |

**Related content**:

* See a video showing this sample in action in the blog, “[FTC to CAM – Create Content Types with specific IDs using CSO](http://blogs.msdn.com/b/vesku/archive/2014/02/28/ftc-to-cam-create-content-types-with-specific-ids-using-csom.aspx).”
* [SharePoint Server 2013 Client Components SDK](http://www.microsoft.com/en-us/download/details.aspx?id=35585)

**Alternative approaches**:

**Description of the sample:**

Overview

This sample shows how you can create site columns and content types, then add site columns to a particular content type. It will also explain the new localization features that have been introduced in the SP2013 SP1 CSOM APIs. You can download the updated APIs from the [SharePoint Server 2013 Client Components SDK](http://www.microsoft.com/en-us/download/details.aspx?id=35585). Notice that the version of the **Microsoft.SharePoint.Client.dll** has been updated and the SP1 file version is set to 15.0.4569.1000.

Creation of content types and site columns

The following code snippet shows how to create a content type using the **ContentTypeCreationInformation** class. Also note that since SP1 you now can set the **Id**:

ContentTypeCollection contentTypes = web.ContentTypes;

cc.Load(contentTypes);

cc.ExecuteQuery();

foreach (var item in contentTypes)

{

if (item.StringId == "0x0101009189AB5D3D2647B580F011DA2F356FB2")

return;

}

// Create a Content Type Information object

ContentTypeCreationInformation newCt = new ContentTypeCreationInformation();

// Set the name for the content type

newCt.Name = "Contoso Document";

//Inherit from oob document - 0x0101 and assign

newCt.Id = "0x0101009189AB5D3D2647B580F011DA2F356FB2";

// Set content type to be avaialble from specific group

newCt.Group = "Contoso Content Types";

// Create the content type

ContentType myContentType = contentTypes.Add(newCt);

cc.ExecuteQuery();

Using **AddFieldAsXml** you can add fields to the **FieldCollection** of a site collection:

FieldCollection fields = web.Fields;

cc.Load(fields);

cc.ExecuteQuery();

string FieldAsXML = @"<Field ID='{4F34B2ED-9CFF-4900-B091-4C0033F89944}' Name='ContosoString' DisplayName='Contoso String' Type='Text' Hidden='False' Group='Contoso Site Columns' Description='Contoso Text Field' />";

Field fld = fields.AddFieldAsXml(FieldAsXML, true, AddFieldOptions.DefaultValue);

cc.Load(fields);

cc.Load(fld);

cc.ExecuteQuery();

Finally, the fields need to be linked to the content type, which is done by using the **FieldLinkCollection** and **FieldLinkCreationInformation** classes as is shown in the following sample:

FieldCollection fields = web.Fields;

Field fld = fields.GetByInternalNameOrTitle("ContosoString");

cc.Load(fields);

cc.Load(fld);

cc.ExecuteQuery();

FieldLinkCollection refFields = myContentType.FieldLinks;

cc.Load(refFields);

cc.ExecuteQuery();

foreach (var item in refFields)

{

if (item.Name == "ContosoString")

return;

}

// ref does nt

FieldLinkCreationInformation link = new FieldLinkCreationInformation();

link.Field = fld;

myContentType.FieldLinks.Add(link);

myContentType.Update(true);

cc.ExecuteQuery();

Localization of content types, list and site titles

If needed, you can localize the site title and site description using the following code snippet:

web.TitleResource.SetValueForUICulture("fi-FI", "Kielikäännä minut");

web.DescriptionResource.SetValueForUICulture("fi-FI", "Kielikäännetty saitti");

For a list, the same approach can be used to localize the title and description:

list.TitleResource.SetValueForUICulture("fi-FI", "Kielikäännä minut");

list.DescriptionResource.SetValueForUICulture("fi-FI", "Tämä esimerkki näyttää miten voit kielikääntää listoja.");

For content types, you have the option to localize the name and description, while for fields, you can localize the title and description values:

myContentType.NameResource.SetValueForUICulture("fi-FI", "Contoso Dokumentti");

myContentType.DescriptionResource.SetValueForUICulture("fi-FI", "Tämä on geneerinen Contoso dokumentti.");

fld.TitleResource.SetValueForUICulture("fi-FI", "Contoso Teksti");

fld.DescriptionResource.SetValueForUICulture("fi-FI", "Tää on niiku Contoso metadatalle.");

## [Create document content types using CSOM](https://github.com/OfficeDev/PnP/tree/dev/Samples/Core.CreateDocumentContentType)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample shows how create document content types using CSOM. | This sample provides a programmatic way to create a document content type by inheriting from an existing content type, then associating it to a custom document template. | This console application uses the features described in the previous sample, “Create content types using CSOM”. |

**Related samples**:

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

**Alternative approaches**:

**Description of the sample:**

Overview

This sample shows how to create document content types and add then associate a document template to the content type.

This sample adds a new content type called ‘Contoso Document’ to the site collection, which can be used in document libraries. This content type has a custom template associated to it when new document is created.

Creation of content types and site columns

ContentType ct = web.ContentTypes.GetById("0x0101009189AB5D3D2647B580F011DA2F356FB2");

cc.Load(ct); cc.ExecuteQuery();

string ctFolderServerRelativeURL = "\_cts/" + ct.Name;

Folder ctFolder = web.GetFolderByServerRelativeUrl(ctFolderServerRelativeURL);

cc.Load(ctFolder);

cc.ExecuteQuery();

string path = @"C:\Data\Test Documents\Doc CT File.docx";

string fileName = System.IO.Path.GetFileName(path);

byte[] filecontent = System.IO.File.ReadAllBytes(path);

using (System.IO.FileStream fs = new System.IO.FileStream(path, System.IO.FileMode.Open))

{

FileCreationInformation newFile = new FileCreationInformation();

newFile.Content = filecontent;

newFile.Url = ctFolderServerRelativeURL + "/" + fileName;

Microsoft.SharePoint.Client.File uploadedFile = ctFolder.Files.Add(newFile);

cc.Load(uploadedFile);

cc.ExecuteQuery();

//Microsoft.SharePoint.Client.File.SaveBinaryDirect(clientContext, ctFolderServerRelativeURL + "/" + fileName, fs, true);

cc.Load(ct); cc.ExecuteQuery();

ct.DocumentTemplate = fileName;

ct.Update(false);

cc.Load(ct); cc.ExecuteQuery();

Console.WriteLine("Content type updates");

}

## [Modify site permissions and external users](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/Core.SitePermissions)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| Two scenarios:   * Modify properties of site collection administrators using CSOM code. * Get the external sharing status and external users of a site collection or tenant. | This sample can be used to modify the site collection administrators on any site collection, including those for OneDrive for Business on Office 365 tenants. The sample also shows how to obtain external sharing status for Office 365 Multi-Tenant installations. | A console application that creates a **ClientContext** object to obtain permissions to list and/or modifiy administrators, and obtain external sharing status. Instructions on how to create a registered app using OAuth tokens are also provided. |

**Related samples**:

* [Provisioning.Pages](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/Provisioning.Pages) (page manipulation)

**Alternative approaches**:

**Description of the sample:**

Overview

This sample focuses on two scenarios:

* Showing how you can easily modify properties of site collection administrators using CSOM code
* Showing how to get the external sharing status and external users of a site collection or tenant

Scenario 1: Working with Site Collection Administrators

This scenario uses only regular CSOM APIs, so having site collection permissions on the site is needed (your account must be a site collection administrator) in order to update the administrators of that site collection. The first step in the sample is to create a **ClientContext** object by a user with the proper permissions:

ClientContext cc = new AuthenticationManager().GetSharePointOnlineAuthenticatedContextTenant(String.Format("https://{0}.sharepoint.com/sites/{1}", tenantName, siteName), String.Format("{0}@{1}.onmicrosoft.com", userName, tenantName), password);

Using that ClientContext object you can then get a list of the current site collection administrators or update the site collection administrators as is shown in the following snippets:

List<UserEntity> admins = cc.Web.GetAdministrators();

List<UserEntity> adminsToAdd = new List<UserEntity>();

adminsToAdd.Add(new UserEntity() { LoginName = "i:0#.f|membership|user@domain" });

cc.Web.AddAdministrators(adminsToAdd);

UserEntity adminToRemove = new UserEntity() { LoginName = "i:0#.f|membership|user@domain" };

cc.Web.RemoveAdministrator(adminToRemove);

You can set the site collection administrators for site collections where you’re not already a site collection administrator by creating a **ClientContext** object using a registered app. Here, the ClientContext object is based on an **OAuth token** with tenant level permissions, as shown in the following snippet:

// Use (Get-MsolCompanyInformation).ObjectID to obtain Target/Tenant realm: <guid>

//

// Manually register an app via the appregnew.aspx page and generate an App ID and

// App Secret. The App title and App domain can be a simple string like "MyApp"

//

// Update the AppID in your worker role settings

//

// Add the AppSecret in your worker role settings

//

// Manually set the permission XML for you app via the appinv.aspx page:

// 1/ Lookup your app via its AppID

// 2/ Paste the permission XML and click on create

//

// Sample permission XML:

// <AppPermissionRequests AllowAppOnlyPolicy="true">

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

// </AppPermissionRequests>

//

// As you're granting tenant wide full control to an app the appsecret is as important

// as the password from your SharePoint administration account!

Once you’ve done that you can use the following code to obtain a ClientContext object for this app:

ClientContext cc = new AuthenticationManager().GetAppOnlyAuthenticatedContext("https://tenantname-my.sharepoint.com/personal/user2", "<your tenant realm>", "<appID>", "<appsecret>");

The provided URL is for the site collection for which you are changing the site collection administrators. Once you have the ClientContext you can use the same code as shown above, but now you can change the site collection administrators for any site collection, including the OneDrive for Business site collections that each of the Office 365 users have.

Scenario 2: Working with external sharing (Office 365 MT only)

This scenario shows how to get the external sharing status of a site collection, and get a list of external users for a specific site collection or for the whole tenant. Since these functionalities require the tenant CSOM libraries, you need to create a **ClientContext** against the tenant admin site collection as in the following snippet. Here the user account needs to be a tenant administrator account.

ClientContext ccTenant = new AuthenticationManager().GetSharePointOnlineAuthenticatedContextTenant(String.Format("https://{0}-admin.sharepoint.com/", tenantName), String.Format("{0}@{1}.onmicrosoft.com", userName, tenantName), password);

Once the **ClientContext** is ready you can use the following code to get the external sharing status and a list of external users:

ccTenant.Web.GetSharingCapabilitiesTenant(new Uri(String.Format("https://{0}.sharepoint.com/sites/{1}", tenantName, siteName)))

List<ExternalUserEntity> externalUsers = ccTenant.Web.GetExternalUsersForSiteTenant(new Uri(String.Format("https://{0}.sharepoint.com/sites/{1}", tenantName, siteName)));

List<ExternalUserEntity> externalUsers = ccTenant.Web.GetExternalUsersTenant();

## [Manage Users and Groups](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/Core.GroupManagement)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This scenario shows how you can manage users, groups, and permissions, within a site collection. | This sample can be used to add and remove users, groups, and permission levels from users/groups. | A SharePoint provider-hosted app that calls methods from the Office 365 PnP Core to manipulate users and groups. |

**Related samples**:

**Alternative approaches**:

**Description of the sample:**

Overview

This sample shows how you can add or remove groups and users within a given site collection. This sample shows the addition of users and groups, followed by giving them permission levels of access to SharePoint. The actual implementation of these group/user/permission level actions is implemented in Office 365 PnP Core as extension methods. This results in a very easy to use model which is shown via the following code snippets:

Adding groups and adding users to groups

cc.Load(cc.Web, web => web.CurrentUser);

cc.ExecuteQuery();

Microsoft.SharePoint.Client.User currentUser = cc.Web.CurrentUser;

if (!cc.Web.GroupExists("Test"))

{

Group group = cc.Web.AddGroup("Test", "Test group", true);

cc.Web.AddUserToGroup("Test", currentUser.LoginName);

}

Removing groups

if (cc.Web.GroupExists("Test"))

{

cc.Web.RemoveGroup("Test");

}

Removing users from groups

cc.Load(cc.Web, web => web.CurrentUser);

cc.ExecuteQuery();

Microsoft.SharePoint.Client.User currentUser = cc.Web.CurrentUser;

if (cc.Web.GroupExists("Test"))

{

if (cc.Web.IsUserInGroup("Test", currentUser.LoginName))

{

cc.Web.RemoveUserFromGroup("Test", currentUser.LoginName);

}

}

Add permission level to group

if (cc.Web.GroupExists("Test"))

{

cc.Web.AddPermissionLevelToGroup("Test", RoleType.Contributor);

}

Add permission level to user

cc.Load(cc.Web, web => web.CurrentUser);

cc.ExecuteQuery();

Microsoft.SharePoint.Client.User currentUser = cc.Web.CurrentUser;

cc.Web.AddPermissionLevelToUser(currentUser.LoginName, RoleType.Reader);

Remove permission level from group

if (cc.Web.GroupExists("Test"))

{

cc.Web.RemovePermissionLevelFromGroup("Test", RoleType.Reader);

}

Remove permission level from user

cc.Load(cc.Web, web => web.CurrentUser);

cc.ExecuteQuery();

Microsoft.SharePoint.Client.User currentUser = cc.Web.CurrentUser;

cc.Web.RemovePermissionLevelFromUser(currentUser.LoginName, RoleType.Reader);

## [Yammer group creation and replacement of existing feed](https://github.com/OfficeDev/PnP/tree/dev/Scenarios/Provisioning.Yammer)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample shows how to provision sites with new Yammer groups and associate them as default news feed for new collaboration sites. | This sample is used to add Yammer to a newly-provisioned collaborations site. | This provider-hosted SharePoint app uses new Yammer capabilities in the Office 365 PnP Core component to create a new Yammer group, and associate it as the discussion feed for the new site. |

**Related samples**:

**Alternative approaches**:

**Description of the sample:**

Overview

This sample shows how to use the newly-added Yammer capabilities in the core component to create a new Yammer group as part of site provisioning and to associate that group as the discussion feed for a newly-created collaborations site.

Scenario 1: Create subsite and replace news feed with Yammer group

To be able to use the sample, follow guidance for this URL to register an access token for your Yammer app. This access token is updated to the web.config of the provider hosted app.

* Get the access token from here: <https://developer.yammer.com/authentication>

Update access token to the web.config of the provider hosted app for the key called **YammerAccessToken**.

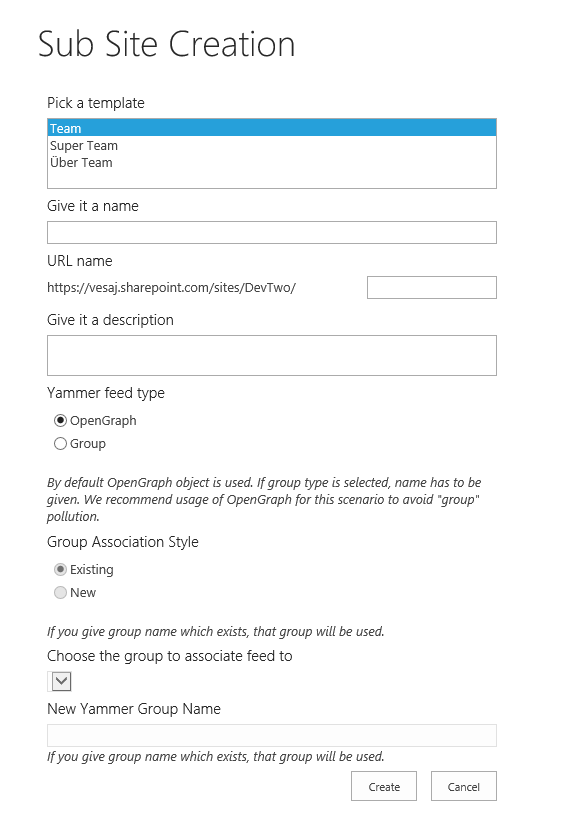
<!-- Details on how to get your access token - check following https://developer.yammer.com/authentication -->

<add key="YammerAccessToken" value="PutYourOwnYammerKeyHere" />

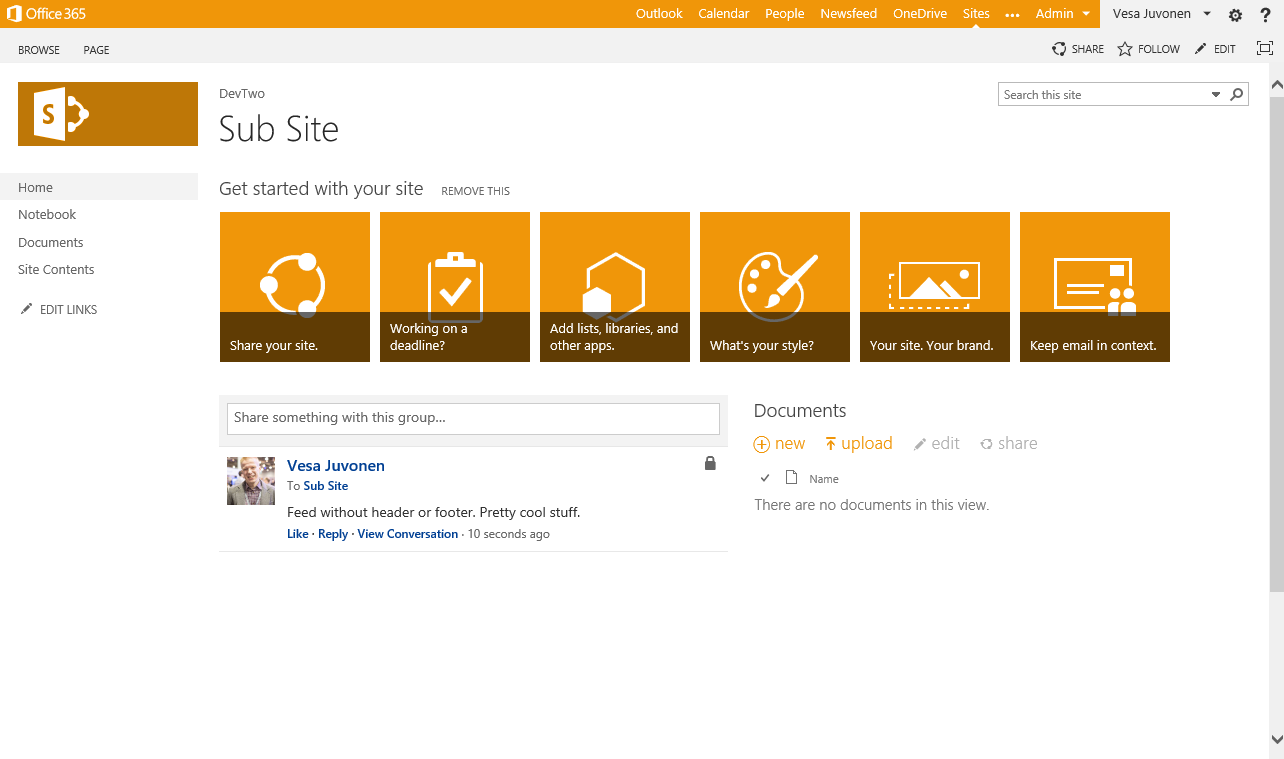
This sample provides simple provisioning UI for subsites where you are able to define if you want to use **OpenGraph** or group feed. If group feed option is selected, you can choose to associate the feed to any existing group or to create a new group for the site.

Note: We do recommend usage of **OpenGraph** objects with the team site feeds rather than using groups. If you automatically provision groups for each of the created team sites, you will end up having a massive amount of groups which will pollute your solution story in Yammer. Usage of the REST APIs for creating groups is also not documented and could have unexpected issues.

We recommend you use **OpenGraph** objects as the default feed technique.

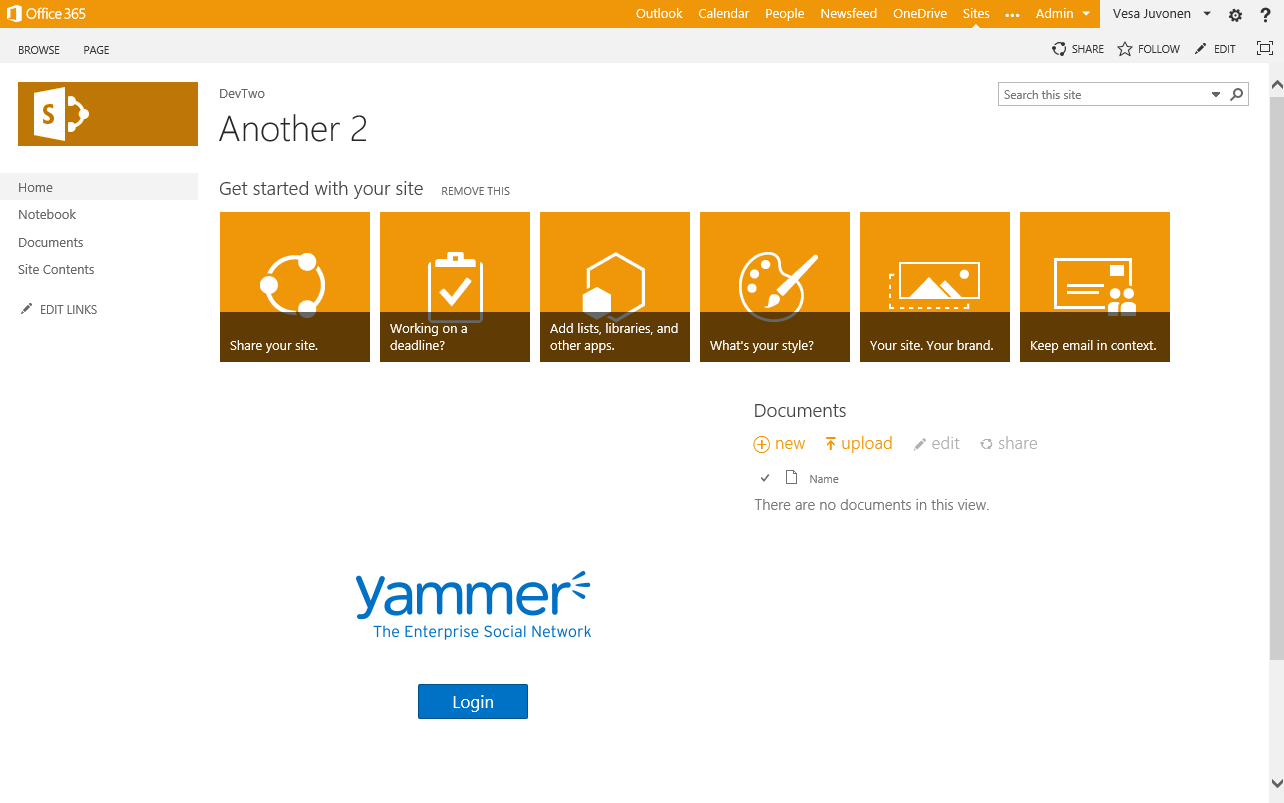


Once the provisioning is completed, you will see the following type of a site with associated Yammer feed running in it. Notice that you will need to be logged into the particular Yammer network in the browser session to show the newsfeed, since integration of the Yammer feed happens on the browser side.

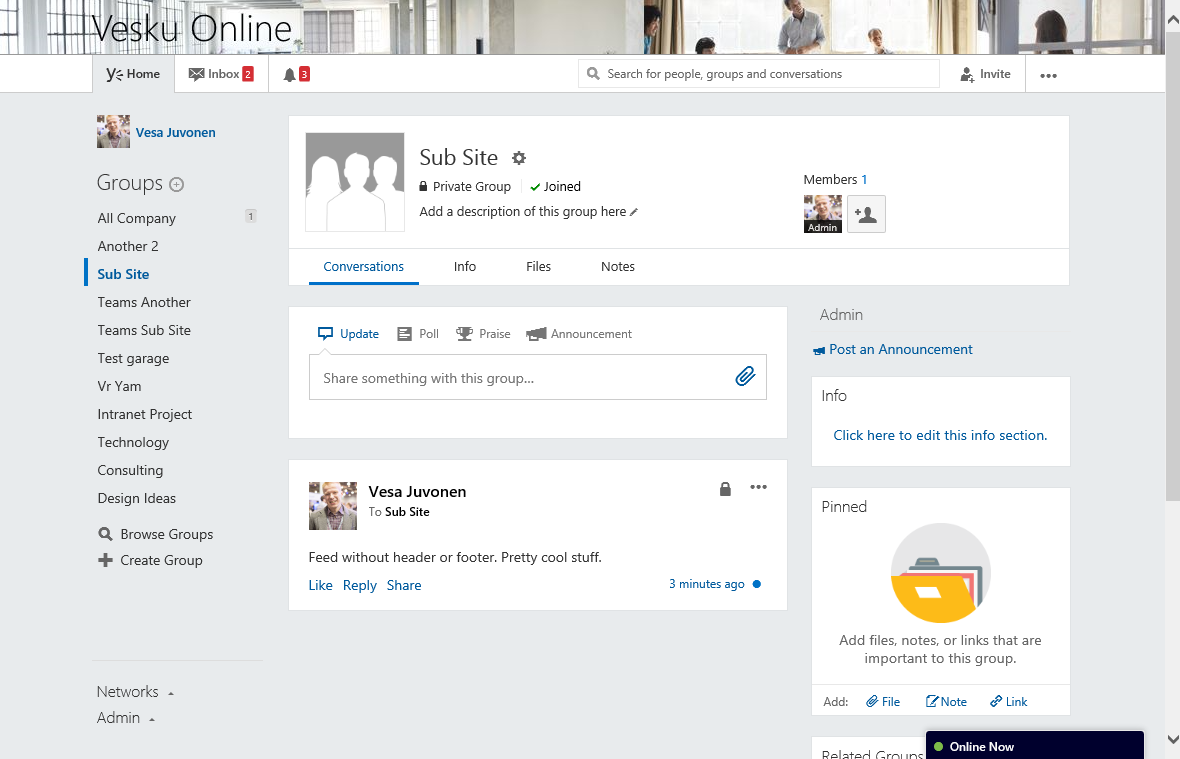


Note: by default, the feed is added without header or footer elements from Yammer, but that can be controlled from the code which is calling the embed capability.

This is how the UI looks if you have not signed to the Yammer network. You can click the login button, which will make the yammer feed to work again as expected.



This example creates new Yammer group for each of the team site. We could actually also create new **OpenGraph** objects if that’s preferred. Either way the group or OpenGraph object is visible also on the Yammer side as follows, and any updates from either side are updated to the feed. Here’s what gets created on the Yammer side as result of the site provisioning:



Notice also that since this configuration is dynamically applied during provisioning time, there’s no impact on removing the provisioning app away from the SharePoint side.

Core component extensions used

Actual provisioning logic and site modifications are using Office 365 PnP Core component extension methods. As you can see we can perform the required actions with only few lines of code due the encapsulated reusable methods from the core component.

public void CreateSubSite(Web hostWeb, string url, string template,

string title, string description, string yammerGroupName)

{

// Create new subsite

Web newWeb = hostWeb.CreateSite(title, url, description, template, 1033);

// Set theme for the site

newWeb.SetThemeToSubWeb(hostWeb, "Orange");

//Remove the "NewsFeed" web part

newWeb.DeleteWebPart("SitePages", "Site feed", "home.aspx");

// Get Yammer Group - Creates if does not exist

YammerGroup group =

YammerUtility.CreateYammerGroup(yammerGroupName, true, ConfigurationManager.AppSettings["YammerAccessToken"]);

// Get Yammer web part

WebPartEntity wpYammer = YammerUtility.GetYammerGroupDiscussionPart(group.network\_name, group.id, false, false);

// Add Yammer web part to the page

newWeb.AddWebPartToWikiPage("SitePages", wpYammer, "home.aspx", 2, 1, false);

}

## [Prevent site deletion](https://github.com/OfficeDev/PnP/tree/dev/Samples/Provisioning.PreventDeleteSites)

|  |  |  |
| --- | --- | --- |
| **What this demonstrates** | **Why you would want to use this**: | **How the app works** |
| This sample shows how to create a sandbox solution for preventing site deletion and using an app to deploy the solution with an application. | This sample provides a solution to the problem of remote event receivers not being able to prevent deletion of sites or other objects. | * Development of a simple sandboxed solution using a **SPSiteEventReceiver** to prevent site deletion. * Deployment of a solution to the Solution Gallery. * Activation, Deactivation, and Removal of a solution in the Solution gallery. |

**Related samples**:

**Alternative approaches**:

**Description of the sample:**

Overview

This sample shows how to create a sandbox solution for preventing site deletion and using an app to deploy the solution with an application. This solution combines a solution which is still partial-trust code (by using a sandbox solution) with a new provider-hosted application. One of the challenges of current remote event receivers is that they do not support a synchronous event processing. This means that when the **SiteDeleting**, **WebDeleting**, or **ListDeleting** events fired, their receivers would not be able to stop the object from being deleted.

This samples demonstrates:

* Development of a simple sandboxed solution using a **SPSiteEventReceiver** to prevent site deletion.
* Deployment of a solution to the Solution Gallery.
* Activation of a solution in the Solution gallery.
* Deactivation of a solution in the Solution gallery.
* Removal of a solution from the Solution gallery.

Security

Permissions for this solution require that the solution have Site Collection Full Control to be able to deploy and activate the sandboxed solution.

Features

The process outlined in the sample describes Deployment, Activation, Testing deletion, Deactivation, and Removal of the solution.

SandBox solutions

The sandbox solution is pretty simple. It consists of an event receiver wired up for **SiteDeleting** and **WebDeleting**. When the user (or system process) attempts to delete a site collection or web application, the event receiver cancels that action.

public override void SiteDeleting(SPWebEventProperties properties) {

properties.Cancel = true;

properties.ErrorMessage = "Site collection cannot be deleted";

}

public override void WebDeleting(SPWebEventProperties properties) {

properties.Cancel = true;

properties.ErrorMessage = "Site cannot be deleted";

}

Solution deployment and removal

The solution deployment consists of uploading the wsp to the Solution Gallery. This uses the **FileCreationInformation** object to upload the file to the gallery. Removing the site is done by finding the uploaded file and calling the **DeleteObject** method.

// get the file from the server path in the provider site

var filePath = Server.MapPath("~/PreventDeleteSites.wsp");

var file = new FileStream(filePath, FileMode.Op\*\*\*\*en);

// create the FileCreationInformation object and prepare

// to upload it to the solution gallery

var fileCI = new FileCreationInformation() {

ContentStream = file,

Url = "PreventDeleteSites.wsp",

Overwrite = true

};

// upload the solution to the gallery

var uploadedFile = solutionGallery.RootFolder.Files.Add(fileCI);

clientContext.Load(uploadedFile);

clientContext.ExecuteQuery();

Solution activation and Deactivation

Solution activation is done using the **DesignPackage** objects to determine the wsp solution package. **DesignPackage.Install** and **DesignPackage.Uninstall** are used to activate and deactivate the solution for the site collection, respectively.

// get the DesignPackageInfo (which is the same name for a sandbox solution)

var wsp = new DesignPackageInfo(){

// during deployment, the solution ID is not necessary

PackageGuid = Guid.Empty, // 4c16c0b9-0162-43ad-a8e9-a4b810e58a56

PackageName = "PreventDeleteSites"

};

// install the solution from the file url

var filerelativeurl = solutionGallery.RootFolder.ServerRelativeUrl + "/PreventDeleteSites.wsp";

DesignPackage.Install(clientContext, clientContext.Site, wsp, filerelativeurl);

clientContext.ExecuteQuery();

Dependencies

* Microsoft.SharePoint.Client.dll
* Microsoft.SharePoint.Client.Runtime.dll
* Microsoft.SharePoint.Client.Publishing.dll
* [Setting up provider hosted app to Windows Azure for Office365 tenant] http://blogs.msdn.com/b/vesku/archive/2013/11/25/setting-up-provider-hosted-app-to-windows-azure-for-office365-tenant.aspx)

# Conclusion

This module introduced site provisioning patterns. In Module 6, learn about strategies for migrating full-trust code (FTC) solutions to the app model.

# Updates

August 1, 2014:

Added **Development and Integration SKUs** section (Section 3) to Module 3.

Added link to new asynchronous site provisioning code sample -- SharePoint 2013: Provision dedicated and on-premises sites with the app model – to Module 1.

October 17, 2014:

Added **Additional samples for provisioning and branding** (Section 6) to Module 5.