# OfFICE AMS: SUBSite Creation Sample

|  |  |
| --- | --- |
| Summary: | Applies to: |
| This sample shows how to override out of the box sub site creation functionality for providing dynamic templates also for sub sites using so called “remote provisioning” pattern. | * Office 365 Multi Tenant (MT) * Office 365 Dedicated (D) * SharePoint 2013 on-premises |
| Solution: | Provisioning.SubSiteCreationApp, version 1.0 |
| Author: | Vesa Juvonen ,Bert Jansen & Frank Marasco (Microsoft) |
| //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // THIS CODE IS PROVIDED \*AS IS\* WITHOUT WARRANTY OF  // ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY  // IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR  // PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

# Introduction

This solution uses so called remote provisioning pattern to provide as flexible sub site template system as possible. Old models using either site definitions, web templates or feature stapling will cause challenges with the evergreen model, meaning more frequent releases of new capability to the cloud and on-premises.

Using remove provisioning pattern will initially require additional code, but since sites created using it are always based on out of the box site definitions, we don’t need to perform any updates when new capabilities are stapled to out of the box templates. This will have significantly reduce the long term maintenance costs for the solution.

Additional information related on the different site provisioning options can be found from following blog posts.

* [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)
* [SharePoint 2013 site provisioning techniques presentation video recording](http://blogs.msdn.com/b/vesku/archive/2013/09/09/sharepoint-2013-site-provisioning-techniques-presentation-video-recording.aspx)

# Scenario: Sub site creation

This scenario shows how we can build the actual sub site creation experience and demonstration of possible options or configuration model for the templates.

## Implementing custom UI for site creation

Scenario is using pretty simple user interface design, which is mimicking the out of the box experience in high level. This is done so that end users don’t get confused and don’t even notice that we’ve taken them to separate provider hosted app, which is actually performing the sub site creation.

## Creating sub site based on selected template

Actual sub site creation is using SharePoint 2013 client side object model. If there’s any modules or files which the site is using, those are uploaded to the site using also client side object model, so that we don’t need to use any feature framework or sandbox elements.

Actual sub site creation is pretty simple with few lines of code using the *WebCreationInformation* object.

// Create web creation configuration

WebCreationInformation information = new WebCreationInformation();

information.WebTemplate = template;

information.Description = description;

information.Title = title;

information.Url = txtUrl;

// currently all english, could be extended to be configurable based on language pack usage

information.Language = 1033;

Microsoft.SharePoint.Client.Web newWeb = null;

newWeb = hostWeb.Webs.Add(information);

ctx.ExecuteQuery();

ctx.Load(newWeb);

ctx.ExecuteQuery();

After the site has been then created, we apply all other needed changes on top of it.

// Add sub site link override. We are going to add some custom javascript

new LabHelper().AddJsLink(ctx, newWeb, this.Request);

// Set oob theme to the just created site

new LabHelper().SetThemeBasedOnName(ctx, newWeb, hostWeb, "Orange");

// LabHelper AddJsLink Impl

string scriptUrl = String.Format("{0}://{1}:{2}/Resources", request.Url.Scheme,

request.Url.DnsSafeHost, request.Url.Port);

string revision = Guid.NewGuid().ToString().Replace("-", "");

string jsLink = string.Format("{0}/{1}?rev={2}", scriptUrl, "CustomInjectedJS.js", revision);

StringBuilder scripts = new StringBuilder(@"

var headID = document.getElementsByTagName('head')[0];

var");

scripts.AppendFormat(@"

newScript = document.createElement('script');

newScript.type = 'text/javascript';

newScript.src = '{0}';

headID.appendChild(newScript);", jsLink);

string scriptBlock = scripts.ToString();

var existingActions = web.UserCustomActions;

ctx.Load(existingActions);

ctx.ExecuteQuery();

var actions = existingActions.ToArray();

foreach (var action in actions)

{

if (action.Description == "scenario1" && action.Location == "ScriptLink")

{

action.DeleteObject();

ctx.ExecuteQuery();

}

}

var newAction = existingActions.Add();

newAction.Description = "scenario1";

newAction.Location = "ScriptLink";

newAction.ScriptBlock = scriptBlock;

newAction.Update();

ctx.Load(web, s => s.UserCustomActions);

ctx.ExecuteQuery();

// LabHelper SetThemeBasedOnName Impl

// Let's get instance to the composite look gallery

List themeList = rootWeb.GetCatalog(124);

ctx.Load(themeList);

ctx.ExecuteQuery();

CamlQuery query = new CamlQuery();

string camlString = @"

<View>

<Query>

<Where>

<Eq>

<FieldRef Name='Name' />

<Value Type='Text'>{0}</Value>

</Eq>

</Where>

</Query>

</View>";

// Let's update the theme name accordingly

camlString = string.Format(camlString, themeName);

query.ViewXml = camlString;

var found = themeList.GetItems(query);

ctx.Load(found);

ctx.ExecuteQuery();

if (found.Count > 0)

{

Microsoft.SharePoint.Client.ListItem themeEntry = found[0];

//Set the properties for applying custom theme which was jus uplaoded

string spColorURL = null;

if (themeEntry["ThemeUrl"] != null && themeEntry["ThemeUrl"].ToString().Length > 0) {

spColorURL = MakeAsRelativeUrl((themeEntry["ThemeUrl"] as FieldUrlValue).Url);

}

string spFontURL = null;

if (themeEntry["FontSchemeUrl"] != null && themeEntry["FontSchemeUrl"].ToString().Length > 0)

{

spFontURL = MakeAsRelativeUrl((themeEntry["FontSchemeUrl"] as FieldUrlValue).Url);

}

string backGroundImage = null;

if (themeEntry["ImageUrl"] != null && themeEntry["ImageUrl"].ToString().Length > 0)

{

backGroundImage = MakeAsRelativeUrl((themeEntry["ImageUrl"] as FieldUrlValue).Url);

}

// Set theme for the web

web.ApplyTheme(spColorURL,

spFontURL,

backGroundImage,

false);

// Let's also update master page, if needed

if (themeEntry["MasterPageUrl"] != null && themeEntry["MasterPageUrl"].ToString().Length > 0)

{

web.MasterUrl = MakeAsRelativeUrl((themeEntry["MasterPageUrl"] as FieldUrlValue).Url); ;

}

ctx.ExecuteQuery();

# Scenario: Override sub site creation option

Since there’s currently no supported way to override the sub site creation experience, we’ll need to do this also using our code. This can be pretty easily achieved by injecting additional JavaScript file on the site, which is only executed when view contents page with the sub site creation link is shown.

## INJECTING Javascript file to the site

This could be done for example automatically when app is installed or it could be automatically done for the sites when they are created. To avoid issues with the MDS (Minimal Download Strategy), we’ll need to do some additional tricks while we inject the JavaScript to the site.

Key point is nevertheless to upload the needed JavaScript to the site and reference that in custom action code, which is added the site. This will ensure that our uploaded JavaScript file is executed one each page request.

As mentioned, these scripts will be then executed on each page request, so we’ll need to ensure that the code is written properly. Natively SharePoint also executes hundreds of lines of JavaScript code on each page request, so as long as this code is optimized, adding few more lines doesn’t really impact the page performance.

## Overriding sub site creation link

Following step is to ensure that the sub site link is overridden each time end user arrives to the *Site Contents* page where this link is located. First we’ll make sure that we only execute the code in the right page.

// Actual execution

function SubSiteOverride\_Inject() {

// Run injection only for site content

if ((window.location.href.toLowerCase().indexOf("viewlsts.aspx") > -1 && window.location.href.toLowerCase().indexOf("\_layouts/15") > -1))

{

SubSiteOverride\_OverrideLinkToAppUrl();

}

}

And second phase is to actually change the link accordingly. In this demo code we only change the link to point to our local debugging app, but in real implementation you’d implemented this either by storing the link to the property bag of the root site in site collection or by having custom configuration list, from where the link target would be taken for the script.

// Actual link override. Checking the right URL from root site collection of the tenant/web application

function SubSiteOverride\_OverrideLinkToAppUrl() {

//Update create new site link point to our custom page.

var link = document.getElementById('createnewsite');

var url = "https://localhost:44339/pages/default.aspx?SPHostUrl=" + encodeURIComponent(\_spPageContextInfo.webAbsoluteUrl);

if (link != undefined) {

// Could be get from SPSite root web property bag - now hardcdoded for demo purposes

link.href = url;

}

}

Notice that since this link update is based on JavaScript execution on client side, you might encounter challenges if the client browser has delays or slowness on the script execution. You can mitigate this by using having for example CSS in the site, which has this link hidden by default, but then it’s shown only after the JavaScript function has been executed.

# Dependencies

* Microsoft.SharePoint.Client.dll
* Microsoft.SharePoint.Client.Runtime.dll