# Office AMS: App Script part pattern

|  |  |
| --- | --- |
| Summary: | Applies to: |
| This sample shows how to provide app script part to SharePoint, which is using JavaScript from provider hosted app for building up the UI in SharePoint | * Office 365 Multi Tenant (MT) * Office 365 Dedicated (D) * SharePoint 2013 on-premises |
| Solution: | Core.AppScriptPart, version 1.0 |
| Author: | Vesa Juvonen (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.  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |

# General comments

This pattern is commonly used cross the industry to integrate systems between the others using JavaScript embedding capabilities. This scenario sample shows how to achieve the similar structure using typical SharePoint provider hosted pattern and how we can provide our extension to be easily available for end users using the web part gallery.

Good examples of the pattern usage is for example Yammer embedding mechanism or how interactive maps are integrated to the sites using Bing or Google

* [Yammer embedding capability](https://developer.yammer.com/connect/)
* [Bing map embed](http://www.microsoft.com/web/post/using-the-bing-maps-api)
* [Google map embed](https://developers.google.com/maps/documentation/javascript/tutorial#Loading_the_Maps_API)

In each of the above scenarios we reference to JavaScript and we use specific div for actually dynamically then contain the actual referenced functionality. Here’s an example of Yammer embed command with JavaScript reference and the div marker to define the location of the capability in the page.

<script type="text/javascript" src="https://assets.yammer.com/assets/platform\_embed.js"></script>

<div id="embedded-feed" style="height:400px;width:500px;"></div>

<script>

yam.connect.embedFeed(

{ container: '#embedded-feed',

network: 'veskuonline.com'  // network permalink

});

</script>

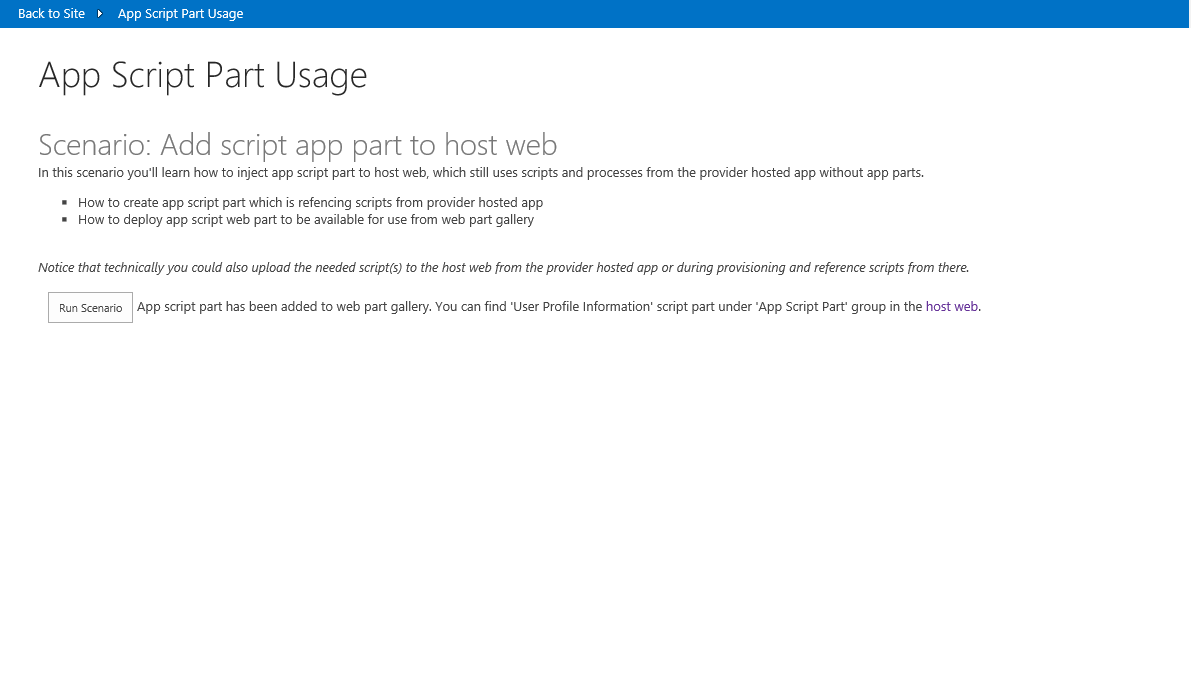
This provides more seamless and dynamic integration option than using app part which are IFrames. This also means that this is suitable option for example for responsive user interface design. So this could be definitely something to evaluate also from app model perspective. What if we would actually just simply deploy redefined script web parts to the SharePoint sites which would have the reference and needed html for provider hosted app reference. This would give the end users opportunity to add additional functionalities to the sites as needed using simply normal SharePoint user experience.

Notice thought that if you would need to provide complex parameterization for each instance on the page, this could be complex to achieve, but not impossible. You could pretty easily recognize when the page is in edit mode and then provide needed parameterization options from embedded JavaScript. Any configuration could be stored for example to the provider hosted app side. It’s good to notice that since we are injecting new web part option to the web part gallery, deployment of the web part definition (.webpart file) requires tenant administration permissions, so this model is not available for apps hosted in the app store.

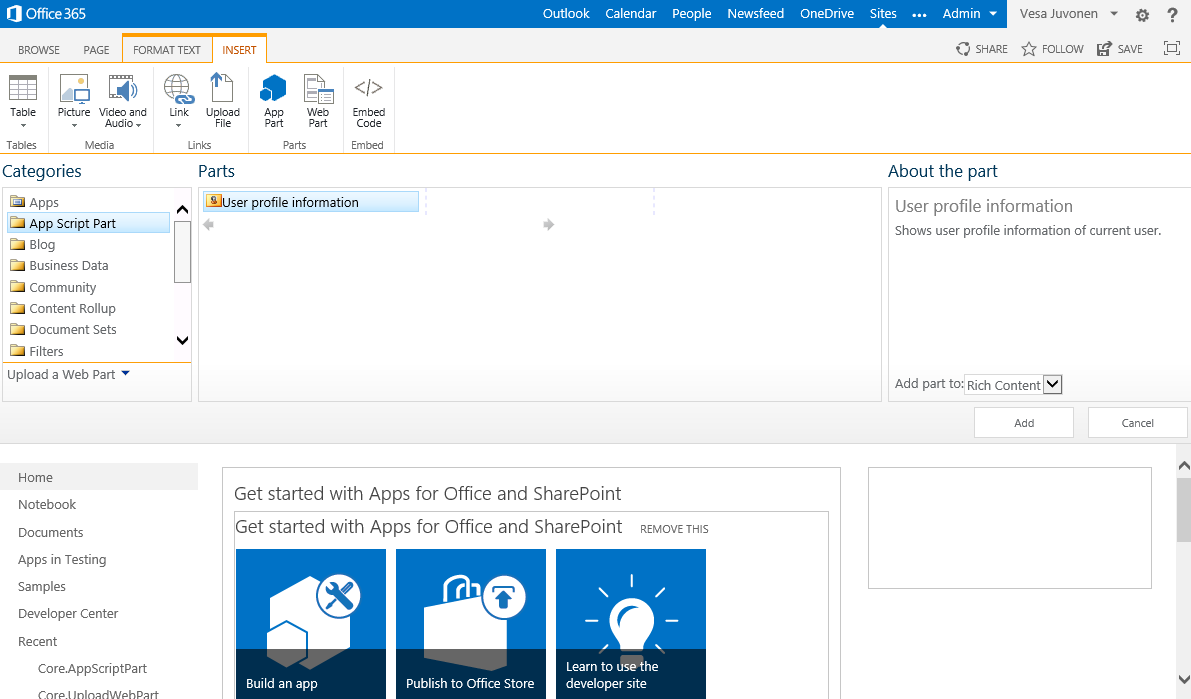
In production we could be running these scripts easily from Windows Azure or from any other centralized location where they can be referenced from the SharePoint pages. This also gives us easy way to update the script, since it’s not stored in the actual SharePoint page, it’s rather loaded completely from the provider hosted environment.

# Scenario: Add App Script part to the web

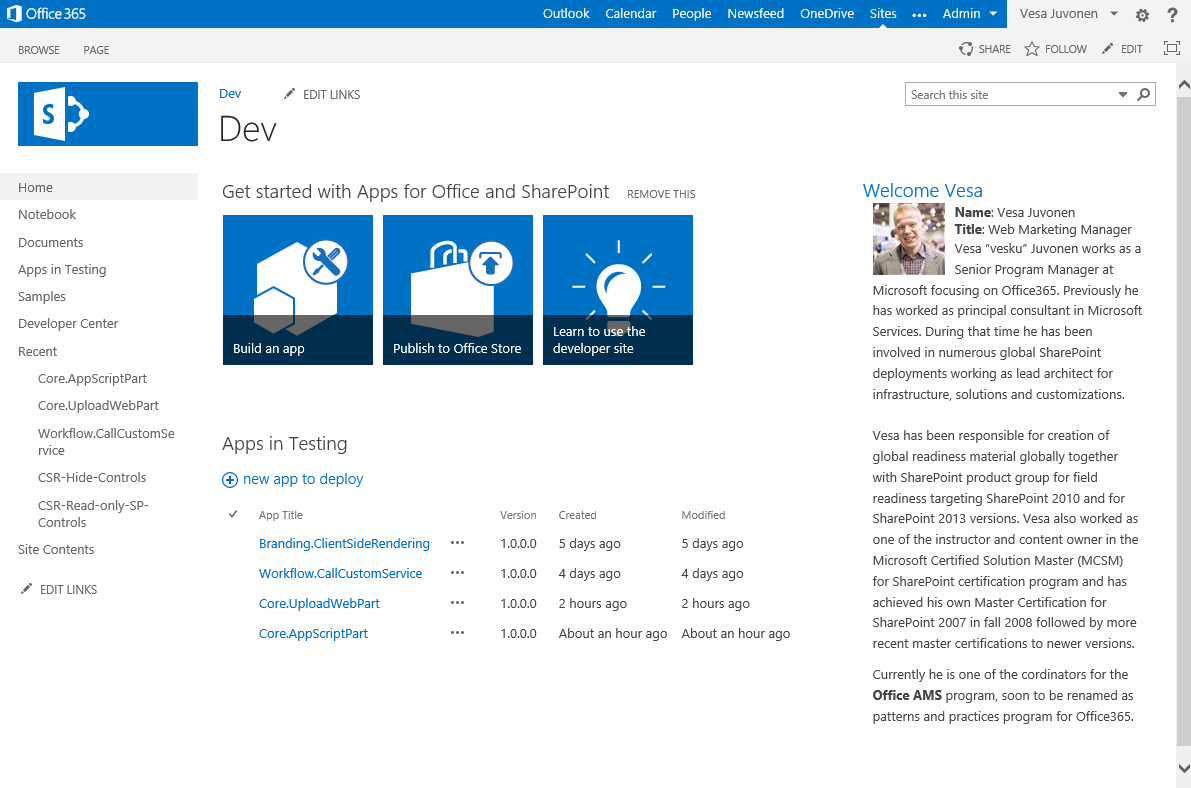
This scenario shows how to modify host web by adding new option as app script part. App script part is out of the box script web part which has been however configured to reference script from the provider hosted app side. In this sample scenario we reference script from the IIS express, so functionality only works when you’re running the Visual Studio debugger, but in real production usage you’d be using some specific URL for the provider hosted app for example hosted from Microsoft Azure platform.



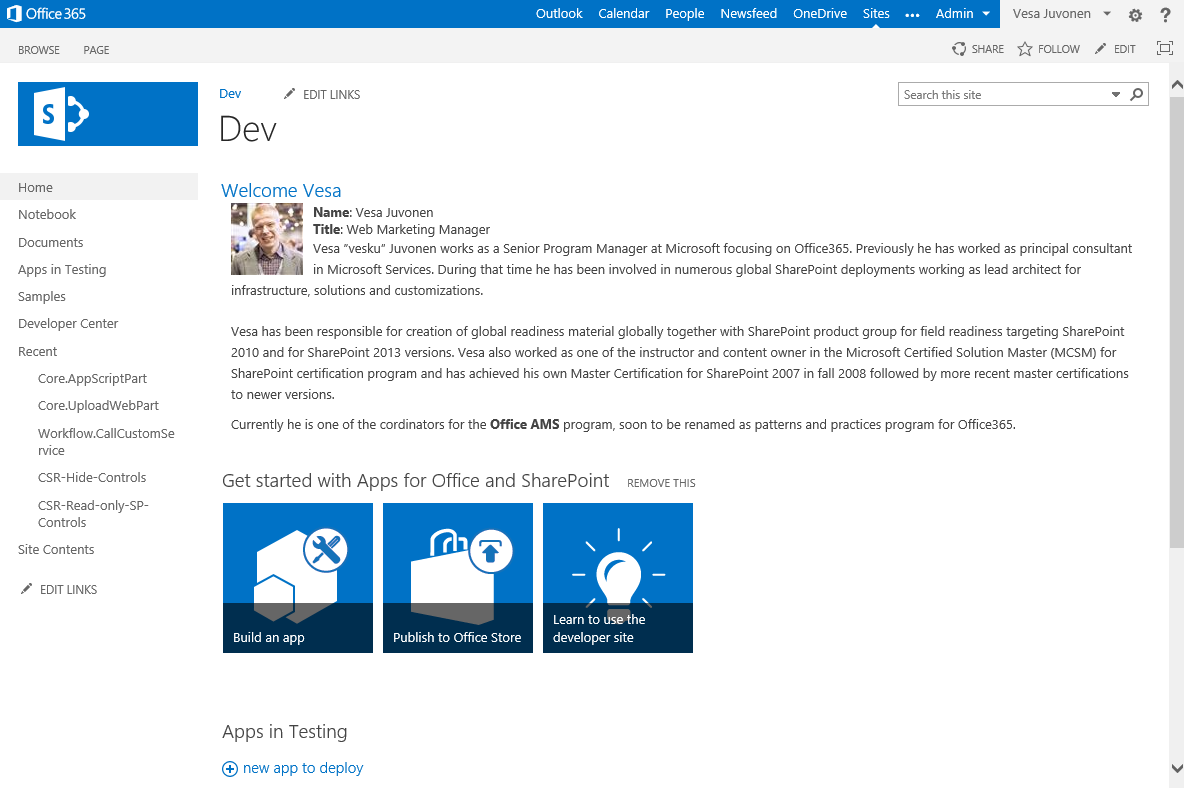
Once the modification has been executed, we can move to the host web and start editing the page for adding a Web Part. Notice that we have new category called “*App Script Part*” and we can locate new “*User Profile Information*” web part under that category.



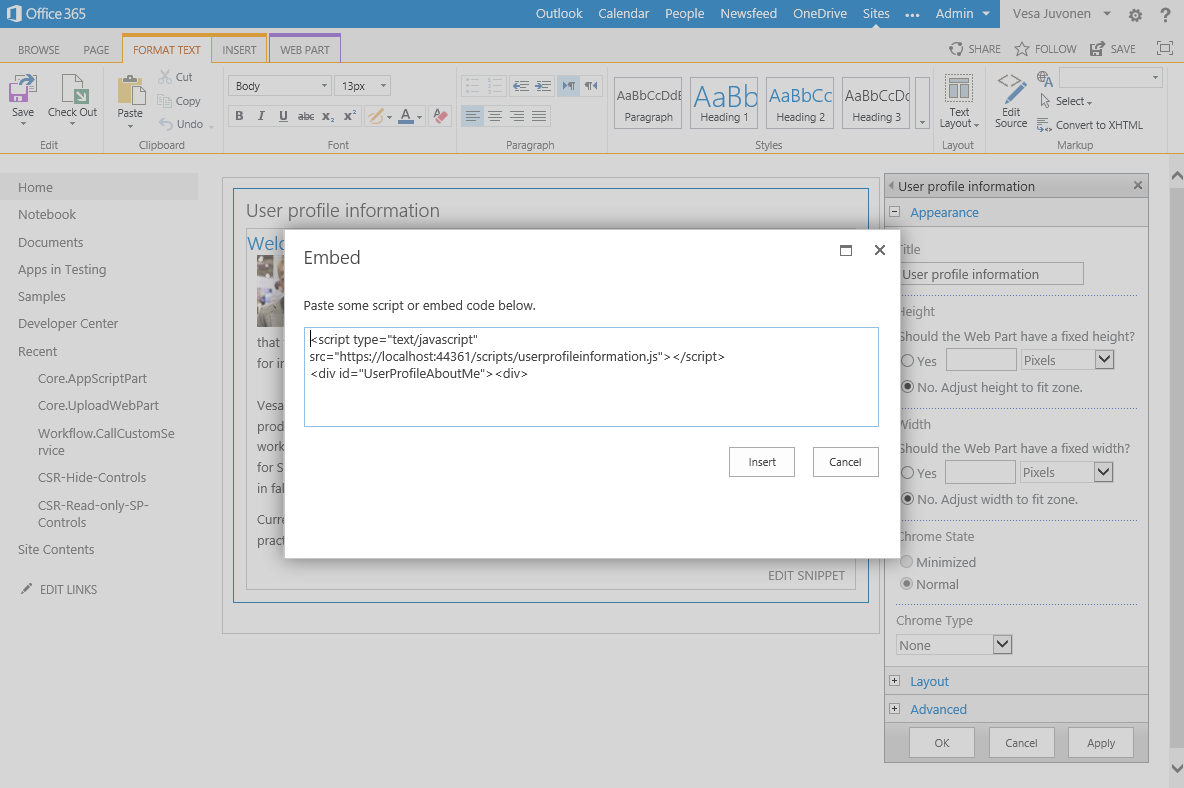
After adding the app script part to the page, we are able to see some user profile information from the particular user like in following picture.



Notice that since we are actually executing the JavaScript from provider hosted app in context of the page, we have full control of the page layout. This means that output can be used with responsive sites or it scales in general based on the layout it is used. This is obviously completely up to the JavaScript which is responsible of rendering the output.



If we take the page in edit mode and have a closer look on what is that app script part, we are able to see that it’s actually a predefined script web part, which has simply a reference to the JavaScript located in the provider hosted app side.



Liked noted already, usage of the local host is not obviously something which would work in the production. You could deploy your JavaScript file to accessible location similarly as how Yammer or Bing maps works.

## Adding web part to the host web

Adding of the web part to the host web is simply implemented by uploading the web part to web part gallery using the *FileCreationInformation* object. In this sample implementation this is done on request when button is pressed, but we could automate this as part of the app installation or simply push the web part to web part gallery from remotely location using similar CSOM logic for example during site collection provisioning. In the code we also set the group attribute properly for the item in the web part gallery, so that web parts are grouped under *App Script Part* group.

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

using (var clientContext = spContext.CreateUserClientContextForSPHost())

{

var folder = clientContext.Web.Lists.GetByTitle("Web Part Gallery").RootFolder;

clientContext.Load(folder);

clientContext.ExecuteQuery();

//upload the "OneDrive for Business Usage Guidelines.docx"

using (var stream = System.IO.File.OpenRead(Server.MapPath("~/userprofileinformation.webpart")))

{

FileCreationInformation fileInfo = new FileCreationInformation();

fileInfo.ContentStream = stream;

fileInfo.Overwrite = true;

fileInfo.Url = "userprofileinformation.webpart";

File file = folder.Files.Add(fileInfo);

clientContext.ExecuteQuery();

}

// Let's update the group for just uplaoded web part

var list = clientContext.Web.Lists.GetByTitle("Web Part Gallery");

CamlQuery camlQuery = CamlQuery.CreateAllItemsQuery(100);

Microsoft.SharePoint.Client.ListItemCollection items = list.GetItems(camlQuery);

clientContext.Load(items);

clientContext.ExecuteQuery();

foreach (var item in items)

{

// Just random group name to diffentiate it from the rest

if (item["FileLeafRef"].ToString().ToLowerInvariant() == "userprofileinformation.webpart")

{

item["Group"] = "App Script Part";

item.Update();

clientContext.ExecuteQuery();

}

}

lblStatus.Text = string.Format("App script part has been added to web part gallery. You can find 'User Profile Information' script part under 'App Script Part' group in the <a href='{0}'>host web</a>.", spContext.SPHostUrl.ToString());

}

## Accessing user profiLE from the JavaScript

Actual accessing and rendering of the information is happening using JavaScript. We are using user profile CSOM to access current user’s user profile for getting the needed information from specific user profile properties. Sample implementation does not have any caching, but in production usage we would recommend to use either HTML local store or cookie based caching for avoiding to JavaScript accessing the user profile on each request. This would also result much faster rendering times for the browser. Typically we could for example cache the information for one hour and refresh after that.