Contents

[1. Introduction 3](#_Toc497937448)

[2. OneDrive file upload to KM Portal 4](#_Toc497937449)

[2.1. Introduction 4](#_Toc497937450)

[2.2. POC Task 4](#_Toc497937451)

[2.3. Overview Diagram 4](#_Toc497937452)

[2.4. Register a File Handler in Azure AD 6](#_Toc497937454)

[2.4.1. Register File Handler Manifest 6](#_Toc497937455)

[2.5. Endpoint application development 9](#_Toc497937456)

[2.5.1 Project Structure 9](#_Toc497937460)

[2.6. Issues 11](#_Toc497937461)

[2.7. References 11](#_Toc497937462)

[3. Upload attachments to KM Portal using Outlook Add-In 12](#_Toc497937463)

[3.1. Introduction 12](#_Toc497937465)

[3.2. POC Task 12](#_Toc497937466)

[3.3. Overview Diagram 12](#_Toc497937467)

[3.4. Create Manifest file and deploy 13](#_Toc497937468)

[3.5. Create Web Application 16](#_Toc497937469)

[3.5.1. Project Structure 16](#_Toc497937470)

[3.6. Create Outlook Service controller 17](#_Toc497937471)

[3.6.1. Project Structure 17](#_Toc497937472)

[4. Uploading documents to KM Portal in Word Online 18](#_Toc497937473)

[4.1. Introduction 18](#_Toc497937474)

[4.2. POC Task 18](#_Toc497937475)

[4.3. Overview Diagram 18](#_Toc497937476)

[4.4. Create Manifest file and Upload through Word Online Add-In tab 19](#_Toc497937478)

[4.5. Create Web Application 22](#_Toc497937479)

[4.5.1. Project Structure 22](#_Toc497937480)

[4.6. Create Service Application 23](#_Toc497937481)

[4.6.1. Project Structure 23](#_Toc497937482)

[5. Custom Tab for Microsoft Teams 24](#_Toc497937483)

[5.1. Introduction 24](#_Toc497937486)

[5.2. POC Task 24](#_Toc497937487)

[5.3. Overview Diagram 24](#_Toc497937488)

[5.4. Create App package 25](#_Toc497937490)

[5.5. Create Web Application 26](#_Toc497937491)

[5.5.1. Project Structure 26](#_Toc497937492)

[5.6. Load package into a Team 27](#_Toc497937493)

[5.7. Access and add a new Tab to your Team 29](#_Toc497937494)

[5.8. References: 31](#_Toc497937495)

[6. Custom Tile for O365 App Launcher 32](#_Toc497937496)

[6.1. Introduction 32](#_Toc497937497)

[6.2. POC Task 32](#_Toc497937498)

[6.3. Add custom Tile 33](#_Toc497937502)

[6.3. Promote the Tile to the Home tab 33](#_Toc497937505)

[6.4. Rearrange custom Tile 34](#_Toc497937506)

[6.5. References: 34](#_Toc497937507)

[7. Appendix 35](#_Toc497937508)

[7.1. OneDrive Sample Manifest file 35](#_Toc497937515)

[7.2. Outlook Sample Manifest file 35](#_Toc497937516)

[7.3. Word Sample Manifest file 35](#_Toc497937517)

[7.4. Teams Sample Page 35](#_Toc497937518)

# Introduction

This documentation contains POC details that have done in O365 arena, which will take the project to next phase and future enhancements are easier when there is a written documentation to go along with the code.

This describes how to develop Add-Ins for OneDrive, Outlook, Word and Tabs for Teams.

Also this explains how to add a custom Tile to O365 App launcher.

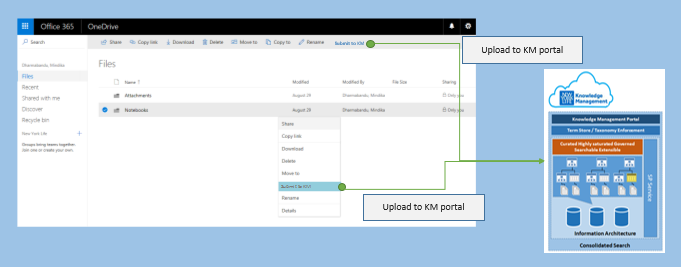
This document uses diagrams, manifest files, sample pages, etc... to explain the POC tasks in more detail.

# OneDrive file upload to KM Portal

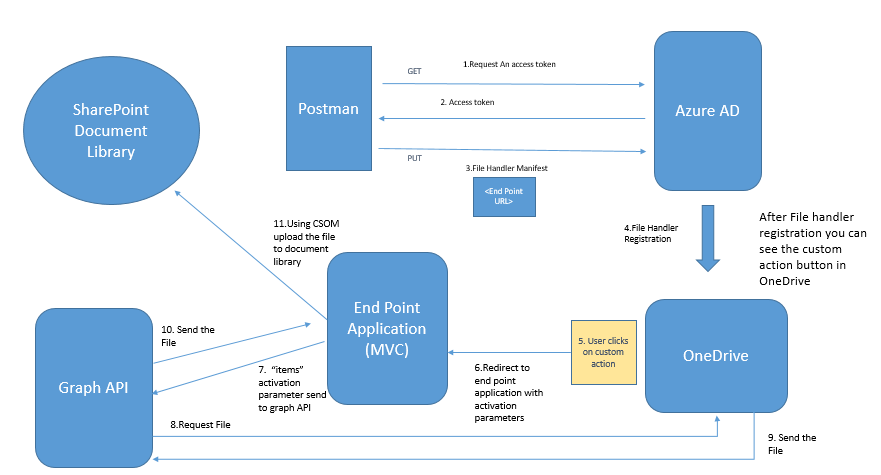
## Introduction

This section describes how to upload OneDrive documents to KM portal. We are developing a custom action for fulfilling this requirement.

## POC Task



## Overview Diagram



Tools used

1. Visual studio 2017 – Building Endpoint application
2. Postman (Chrome Extension) - Deployment Purpose

Templates used

1. Downloaded the ASP.NET MVC sample

<https://developer.microsoft.com/en-us/graph/quick-start>

Steps

1. Register a File Handler in Azure AD
2. Endpoint application & service development

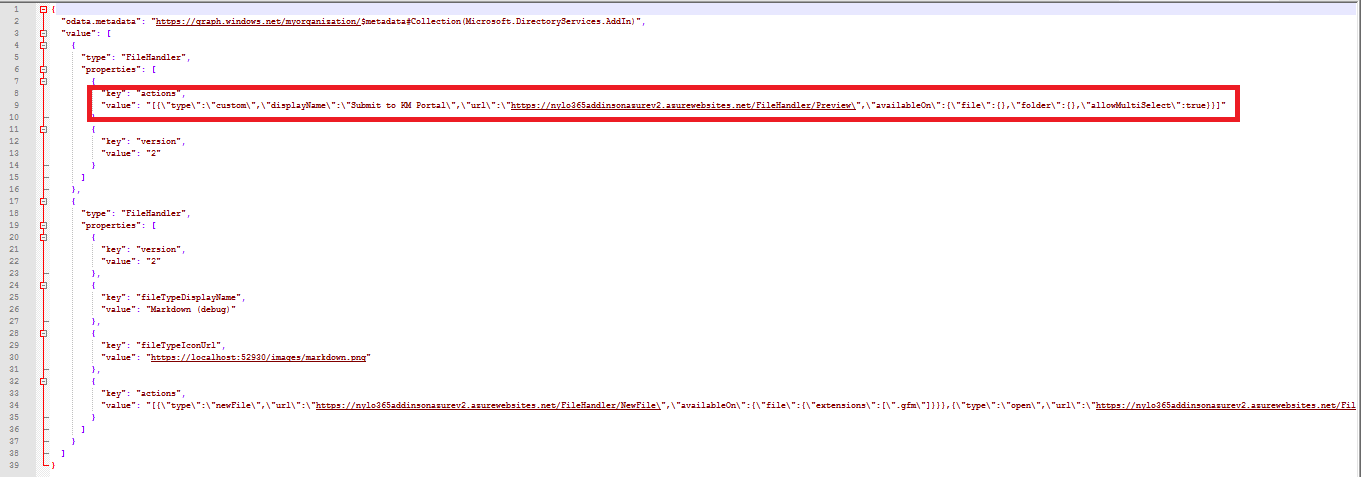
## Register a File Handler in Azure AD

File Handler includes two parts

1. File Handler Manifest
2. File Handler Endpoint

### Register File Handler Manifest

Manifest file defines the interaction between OneDrive and the File handler endpoint. The manifest should be registered with the Azure AD.



In this Solution we are considering “custom” type section only. There you need to change the following parameters according to your requirement.

* type : This should be ‘**custom’**
* displayName : This is the text that will show in the custom action
* url : URL to the end point application
* availableOn : What type of files or folders this custom action should be available. You can provide extensions also.
* allowMultiSelect : Allow on multi select or not

(Refer Appendix 7.1 section for sample Manifest file)

**Required Permissions**

* Tenant Admin user

**Obtain an access token**

To obtain an access token, you can use a tool like Postman.

**Steps to Follow:**

1. Install the Postman chrome extension or desktop application and launch the app.
2. On the Authorization tab, change Type to OAuth 2.0.
3. Click Get New Access Token. A dialog will appear asking for a number of fields to be completed.
4. In your browser, navigate to the Azure Portal and sign-in.
5. Select Azure Active Directory, then click on App registrations.
6. Click Add to create a new application, which will be used to manage file handler manifests
7. Name the application File handler registration tool (FileHandler Display Name) and set the sign-on URL to the value Postman tells you to use: https://www.getpostman.com/oauth2/callback.
8. Scroll to the end of the application list and click on File handler registration tool.
9. Copy the application ID value (guid) and paste it into Postman next to Client ID.
10. Click on Required permissions, then Windows Azure Active Directory.
11. Click on Required permissions, then Windows Azure Active Directory.
12. Under Delegated permissions find and check the following permissions:

* Sign in and read user profile
* Read and write directory data
* Access the directory as the signed in user

1. Click Save.
2. Click on Keys, then create a new key by changing the Expires drop down to Never expires and clicking Save.
3. Copy the value of the new key, and paste it into Postman next to Client Secret.
4. In Postman, complete the access token form, by providing the following values:

* Token name: File handler registration tool
* Auth URL: https://login.microsoftonline.com/common/oauth2/authorize?resource=https://graph.windows.net
* Access Token URL: https://login.microsoftonline.com/common/oauth2/token
* Grant Type: Authorization code

1. Click Request Token and sign in as a **tenant administrator**.
2. After consenting to the permissions, Postman will show you a token for File handler registration tool. Select that entry, and change Add token to **Header** and then click **Use** **Token** to append the new access token.
3. Make a request to find your file handler application manifest, by entering the URL: https://graph.windows.net/myorganization/applications/?api-version=1.6.
4. Change the URL to https://graph.windows.net/myorganization/applications/{applicationObjectId}/addIns?api-version=1.6. Replace {applicationObjectId} (In AAD) with the value obtained previously.
5. Change the request type to PUT.
6. Click on the Body tab, and then select Raw and change the Text drop down to JSON.
7. Copy and paste the file handler manifest into the body window. The body request needs to be wrapped inside a JSON object, with a value array containing the file handler manifest: {"value": [ { /\* manifest here \*/} ] }. Important: this request will overwrite any existing file handlers or addIns registered for this application. If there are other addIns registered, you need to submit all of the addIns in the same request. (Please find the attached Sample Manifest)
8. Click **Send** to update the application.
9. If you see the **response status as 204 No Content** then the update was **successful**.

Now that your File handler manifest has been registered with your application in Azure AD.

**Note: Changes to the add-in manifest may not be immediately applied. The file handler’s manifest is cached for performance. Changes to the file handler manifest can take up to 24 hours to reflect.**

**Force the cache to be cleared for Development purposes**:

**https://docs.microsoft.com/en-us/onedrive/developer/file-handlers/reset-cache**

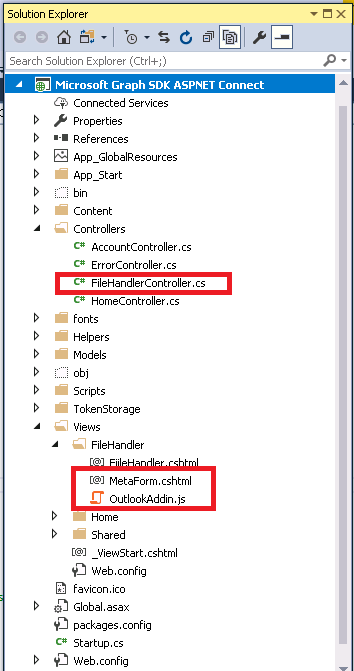
## Endpoint application development

Endpoint application includes the logic of the File handler action. In File handler manifest, we are registering an endpoint Url in Azure AD. This URL should point to this application. When the user clicks on OneDrive Custom action, it sends the **POST** Request to End point Application. This POST Request includes,

* cultureName : The locale identifier for the user's current display language.
* client: The Office 365 application from which the file handler was invoked; for example "SharePoint" or "OneDrive".
* userId: The UPN/login email for the user who invoked the file handler.
* domainHint: A domain hint string that indicates either organizations or consumers.
* items: A collection of Microsoft Graph URLs to the selected item(s).



### Project Structure



For this application, we are using “items” Activation parameter. We can retrieve them in C# using:

var itemsJson = Request.Form["items"];

var itemUrls = JsonConvert.DeserializeObject<string[]>(itemsJson);

itemUrls are Graph API urls. We need to develop graph authenticated application.

To do that we can make use of “**Microsoft Application Registration portal**” (<https://developer.microsoft.com/en-us/graph/quick-start>). After filling required fields, you can download a graph authenticated sample application.

This is a basically C# MVC application. You can pass the **itemurl** to graph api and retrieve the selected file as a memory stream using following code.

GraphServiceClient graphClient = SDKHelper.GetAuthenticatedClient();

var request = graphClient.Shares[itemUrls];

var foundFile = await request.Request().GetAsync();

MemoryStream stream = (MemoryStream)await graphClient.Me.Drive.Items[foundFile.Id].Content.Request().GetAsync();

After retrieving the memory stream, we can upload the document to SharePoint Document library using CSOM code.

## Issues

1. In current solution, activation parameters getting lost in first attempt.

Reason to this is the application redirects to O365 sing-in URL. Within this request, parameters are got lost. Second time onwards it’s working correctly.

1. Custom action displayed only for File handler registered users.
2. File handler Manifest registration takes 24 hours in Azure AD, but o365 gives by-force alternative for development :

See Refresh file handler cache:

<https://docs.microsoft.com/en-us/onedrive/developer/file-handlers/reset-cache>

## References

**File handler tutorial**

<https://docs.microsoft.com/en-us/onedrive/developer/file-handlers/>

**Microsoft graph explorer**

By using this you can test “items” sent from OneDrive with activation parameter

<https://developer.microsoft.com/en-us/graph/graph-explorer>

**Microsoft Application Registration portal**

You can manage the registered application

<https://apps.dev.microsoft.com/>

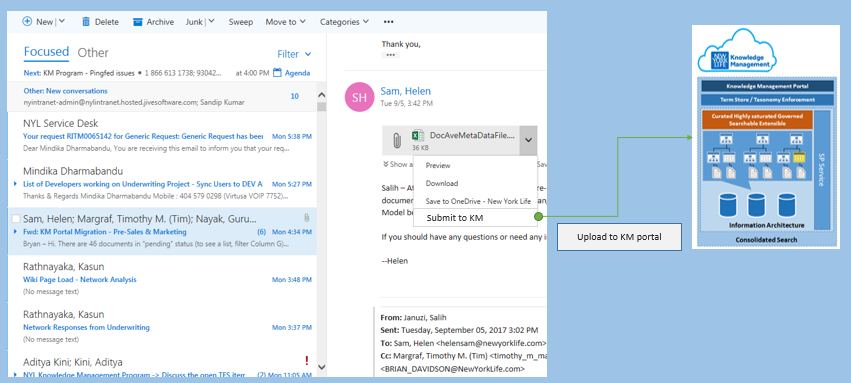
# Upload attachments to KM Portal using Outlook Add-In



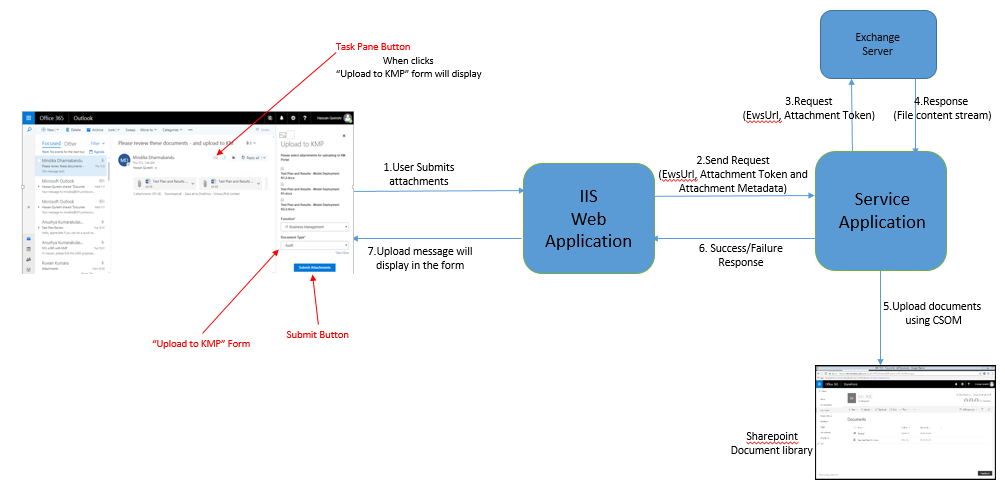
## Introduction

Using an Outlook add-in, user can upload attachments directly to KM portal. User can upload all attachments or one attachment at a time. Uploaded document(s) will be updated with the metadata information provided. We can deploy add-In to Outlook from Admin center.

## POC Task



## Overview Diagram



Tools used

1. Visual Studio 2017

Steps

1. Create Manifest file and deployed through Visual Studio
2. Create Web Application
3. Create Service Application

## Create Manifest file and deploy

You can create Outlook Add-In using Visual Studio.

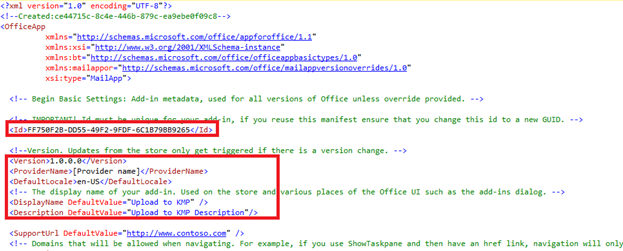
**File** -> **New** -> **Project** -> **Office/Sharepoint** -> **Outlook Web Add-In**

(Refer Appendix 7.2 section for sample Manifest file)

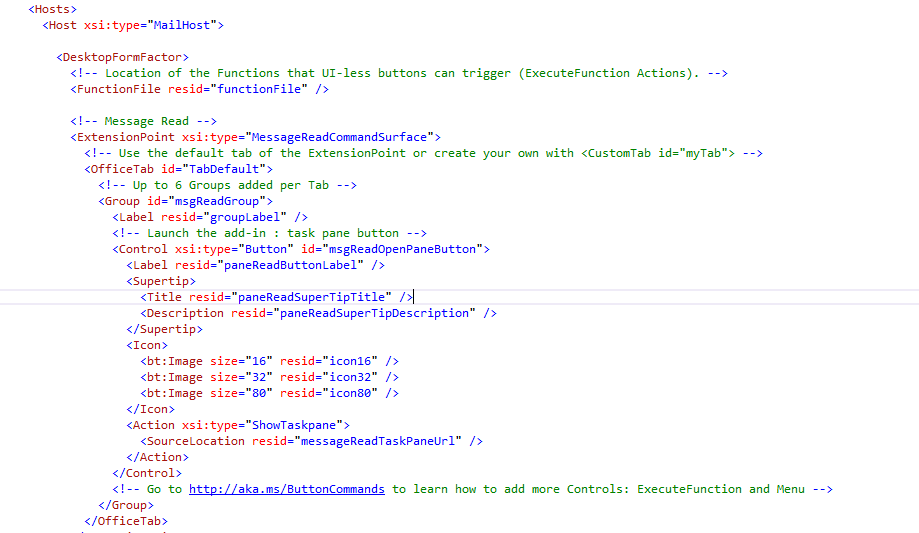
Inside the manifest file, you have to change following properties.

1. ID**, D**isplay name, Description and Image Icon URLs of the add-In
2. SourceLocation - Here you will set the Url of the Page, that needs to be displayed in outlook add-in pane

See the following image, which highlights all the properties we have changed.





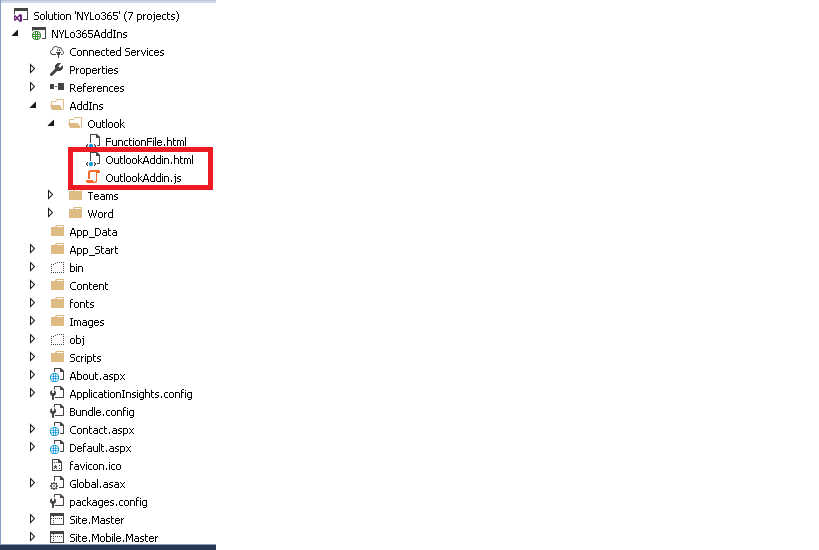




## Create Web Application

Web application will be used to host the display Form, which will capture Metadata information about the attachments. User can send attachments one by one or all at a time.

### Project Structure

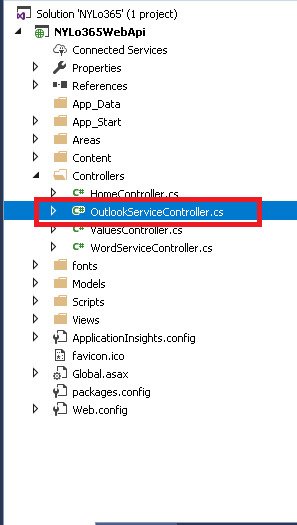


This web application contains OutlookAddIn.html and OutlookAddIn.js files which contains task pane View and its logic. Web application passes the ewsUrl and attachmentToken along with other document related properties to the Service application.

## Create Outlook Service controller

Web service application is used to receive data from web application and send them to SharePoint Online library.

### Project Structure



You can create visual studio Web API template project for this Service application.

This Service application will pass the ewsUrl, attchmentToken and attachementId to Exchange Server and retrieve file content stream. Then MemoryStream object will be saved as a file to Sharepoint online document library using CSOM.

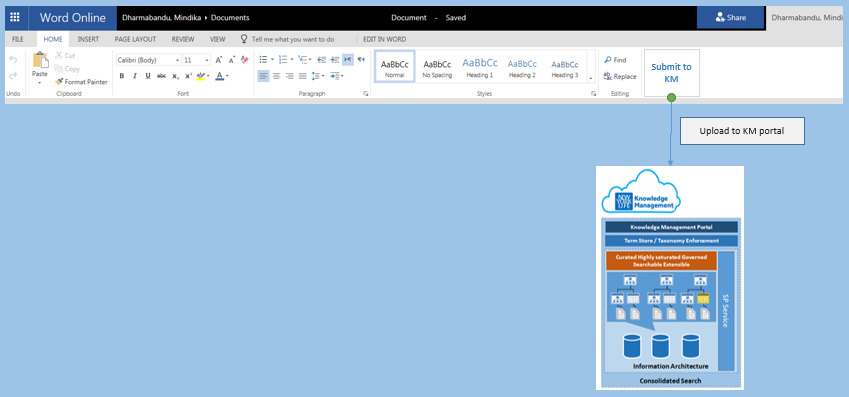
# Uploading documents to KM Portal in Word Online

## Introduction

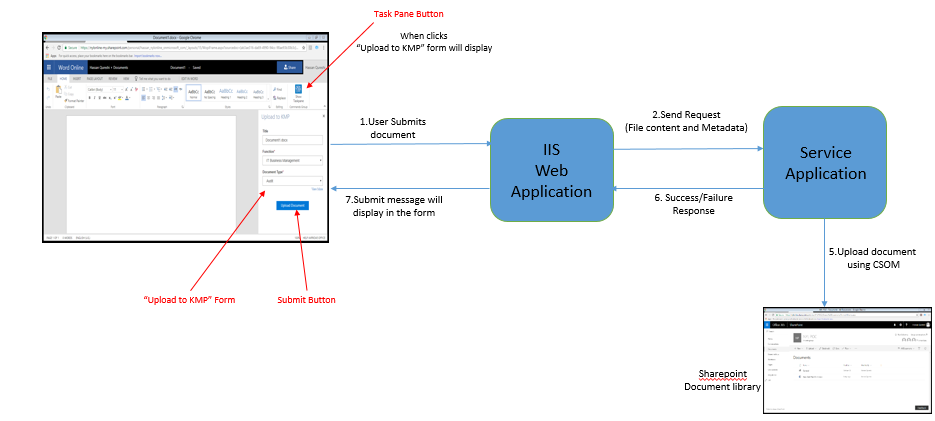
Using a Word add-in, user can upload documents directly to KM portal. For that, user needs to add the app manifest file to Word online.

User can go to the **Insert** >> **Office-add-ins** >> **Upload My add-ins** and upload the app manifest file.

## POC Task



## Overview Diagram



Tools used

1. Visual Studio 2017

Steps

1. Create Manifest file and Upload through Word Online Add-In tab
2. Create Web Application
3. Create Service Application

## Create Manifest file and Upload through Word Online Add-In tab

You can create Word Add-In using Visual Studio.

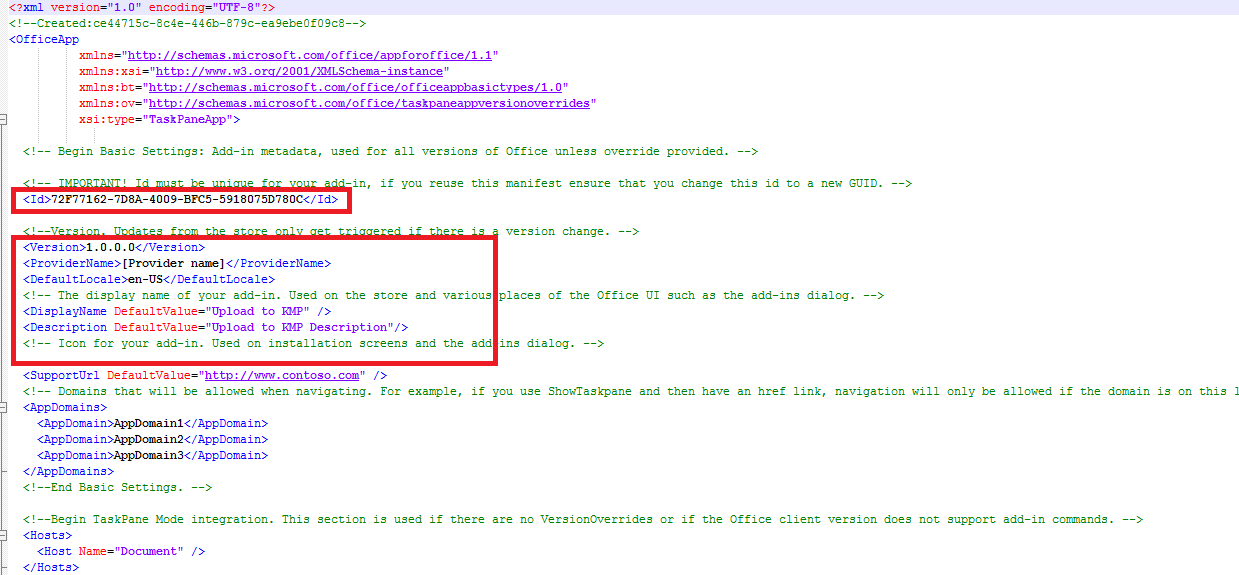
**File** -> **New** -> **Project** -> **Office/Sharepoint** -> **Word Web Add-In**

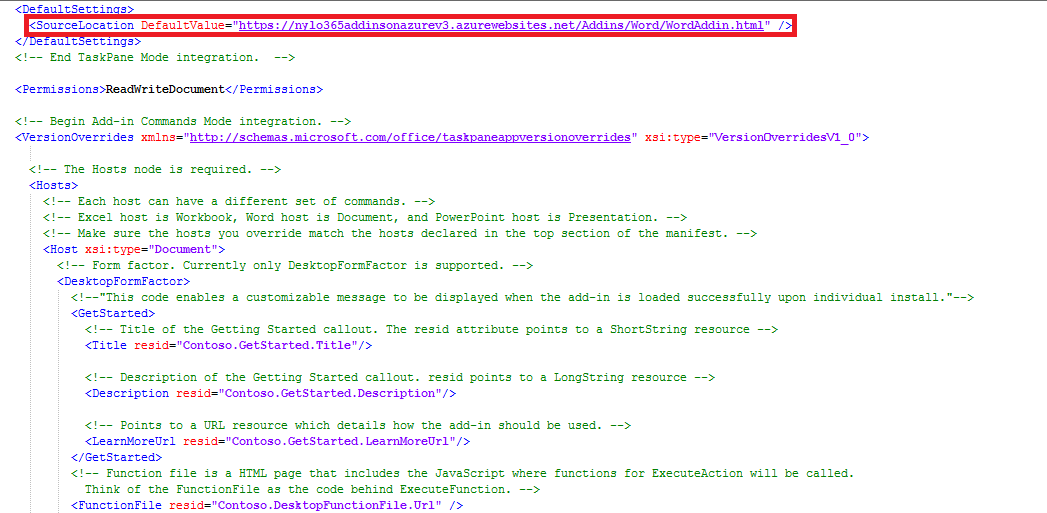
(Refer Appendix 7.3 section for sample Manifest file)

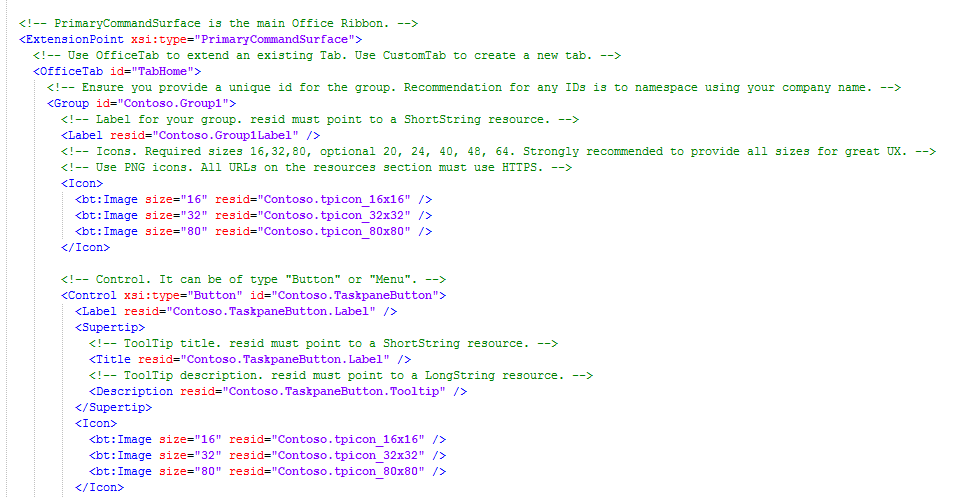
Inside the manifest file, you have to change following properties.

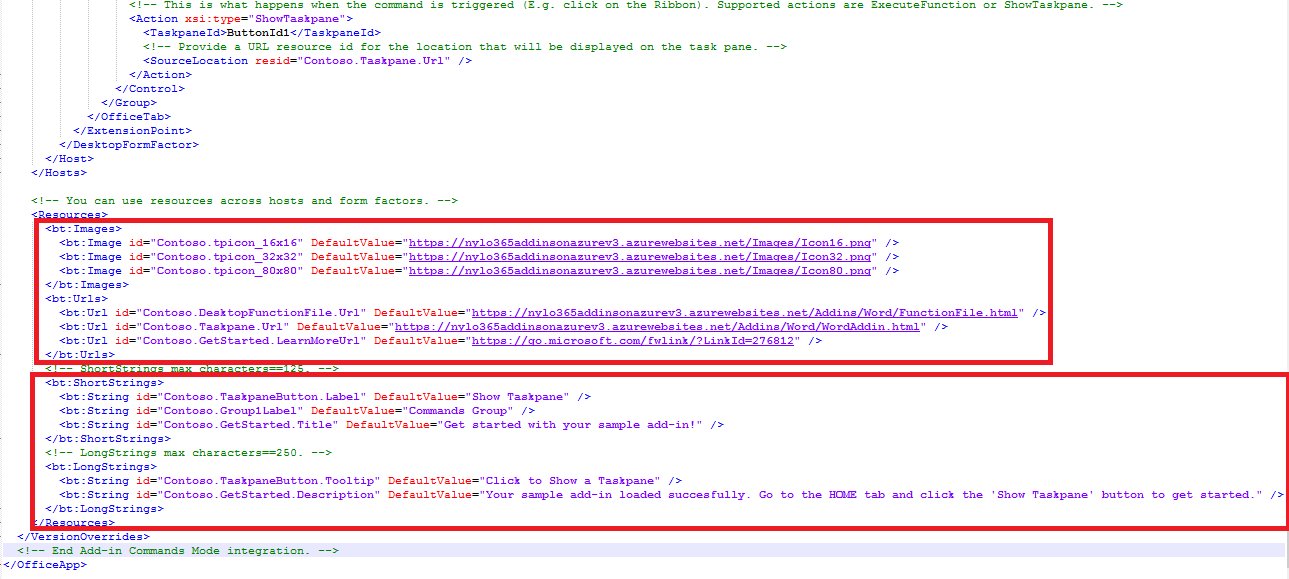
1. ID, Provider Name**, D**isplay Name, Description and Icon URLs of the add-In
2. SourceLocation - Here you will set the Url of the Page which needs to be displayed in word add-in pane

See the following image, which highlights all the properties we have changed.





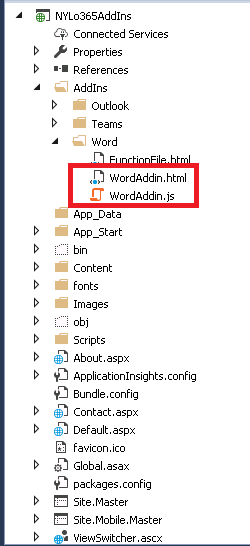




## Create Web Application

Web application will be used to host the Display form, which will capture metadata information about the document. In the manifest file, you will set this display Form URL of the web application.

### Project Structure

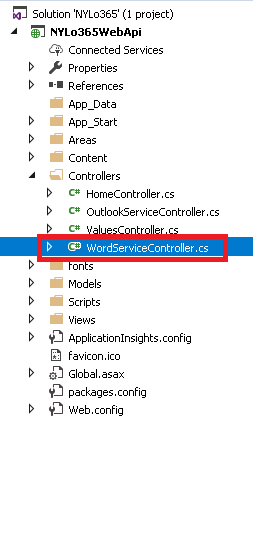


This web application contains WordAddIn.html and WordAddIn.js files, which contains task pane View and its logic. Web application passes chunks of the file content along with its metadata properties to the Service application.

## Create Service Application

Web service application is used to receive data from web application and send them to SharePoint Online library.

### Project Structure



You can create visual studio Web API template project for this Service application.

This Service application will receive the file content stream, converts to a MemoryStream object and save as a file to SharePoint online document library using CSOM.

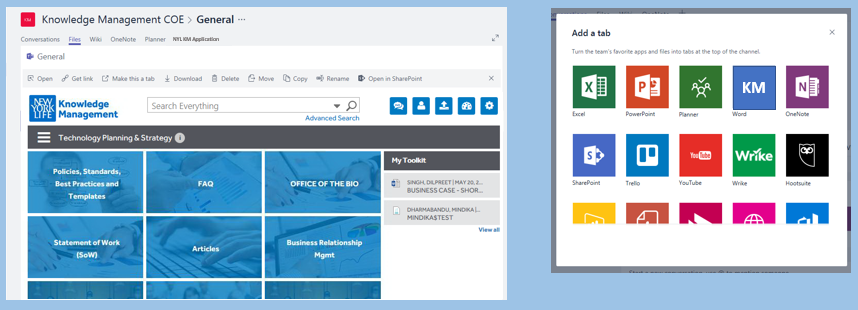
# Custom Tab for Microsoft Teams



## Introduction

This section contains guide lines that followed POC works for adding Custom Tab in Microsoft Teams. App experiences in Teams are defined by their app manifest, and bundled in an app package (.zip) for use in side loading or Office Store submission.

## POC Task



## Overview Diagram

****

Tools used

1. Visual Studio 2017

Steps

1. Create App package (.zip package)
2. Create Web Application
3. Load package into a Team
4. Access and add a new Tab to your Team

## Create App package

A Teams app package is a .zip file containing:

* A manifest file named "manifest.json" - Specifies attributes of your app and points to required resources for your experience, such the location of its tab configuration page
* A transparent "outline" icon and a full "color" icon

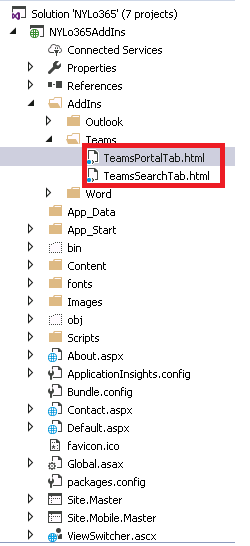
You can download Full App package from below location and customize as required.

<https://github.com/OfficeDev/Microsoft-teams-docs/blob/master/teams/FullAppPackage.zip>

## Create Web Application

Web application will be used to host the page, which will capture user information while Tab adding process. In the manifest file you will set this Page url.

### **Project Structure**



**This page should include Microsoft Teams JavaScript library (**<https://statics.teams.microsoft.com/sdk/v1.0/js/MicrosoftTeams.min.js>) **and call microsoftTeams.initialize**();

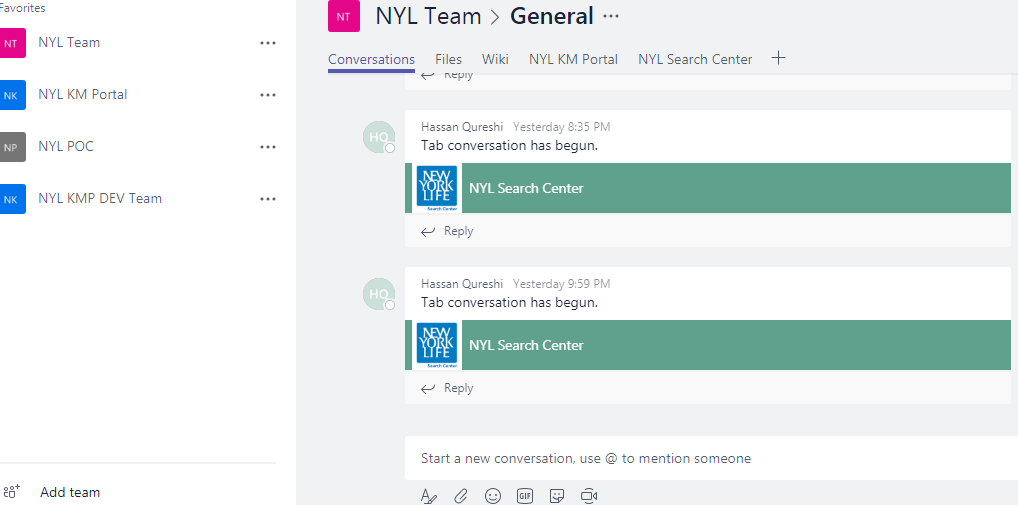
**You can set different properties like** contentUrl, suggestedDisplayName **in** microsoftTeams.settings.setSettings() **function and finally call** notifySuccess().

**Note : You must host your page on a secure HTTPS endpoint, ensure that your page permits itself to be iframed.**

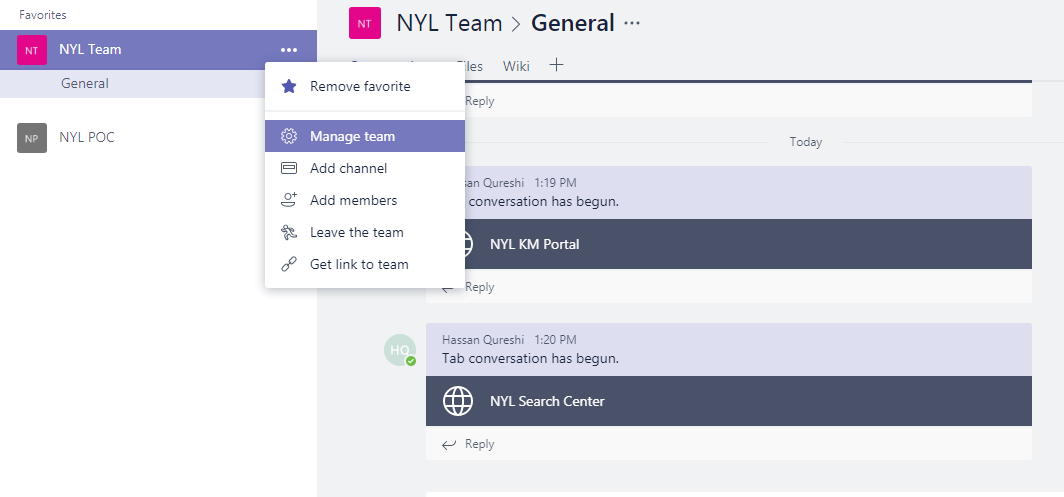
**(Refer Appendix 7.4 section for a sample Page developed)**

## Load package into a Team

1. Open Microsoft Teams and create/edit a team

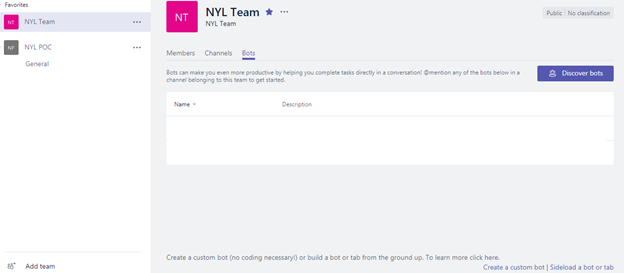


1. Choose More options (⋯) and choose **Manage team**.



1. Manage Team Option will navigate to the page Team dashboard.

There selects the tab named **Bots.**



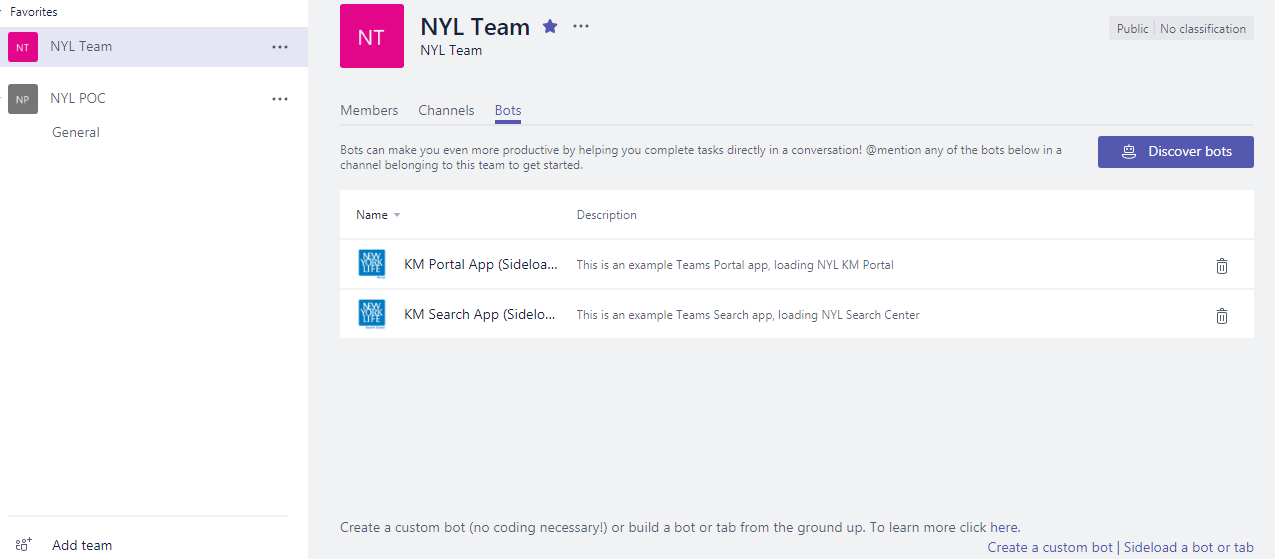
1. Then select the **Side load a bot or tab**

**Note: For sideloading to work, your tenant admin must first enable sideloading of apps.**

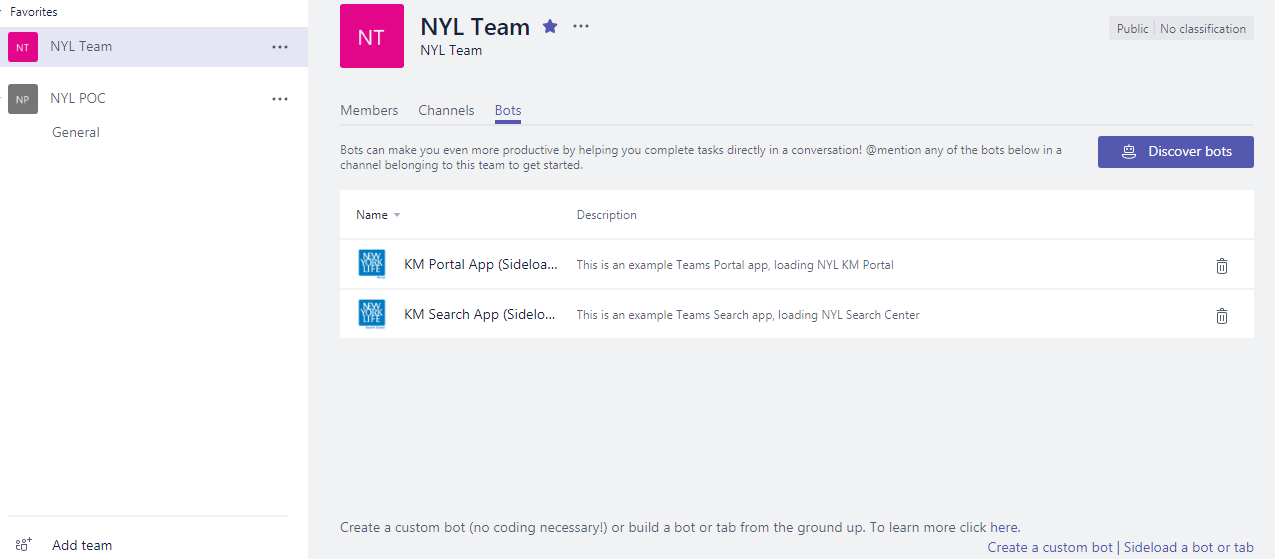
Refer following article for enabling sideloading of apps for Microsoft Teams:

<https://msdn.microsoft.com/en-us/microsoft-teams/setup#3-enable-sideloading-of-apps-for-microsoft-teams>

1. Select the folder path which has the .zip package.
2. Upload the package to the Team.

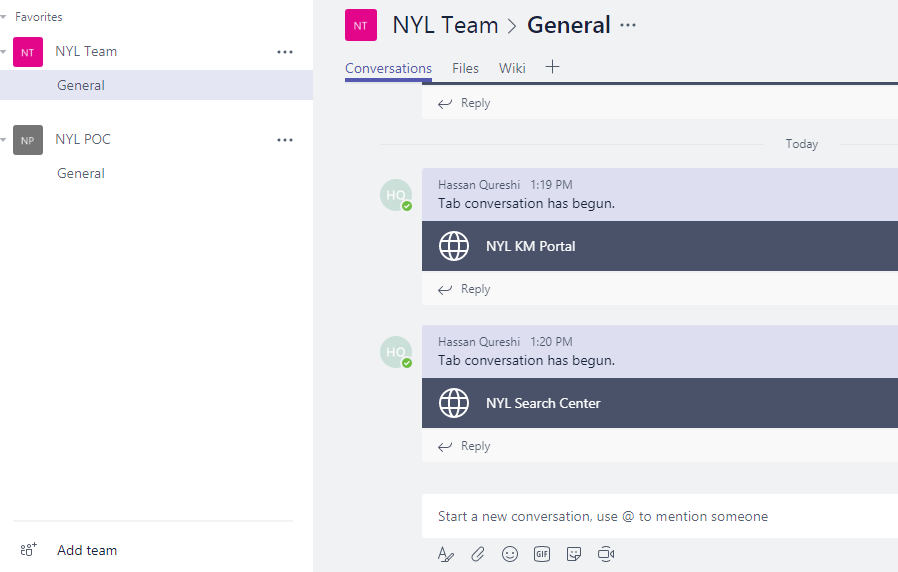


1. Uploaded packages are available on the Microsoft Teams dashboard.

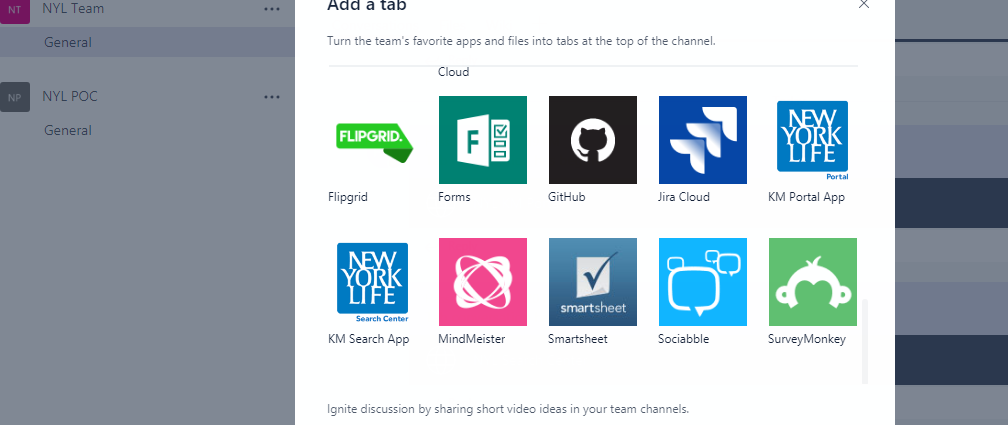


## Access and add a new Tab to your Team

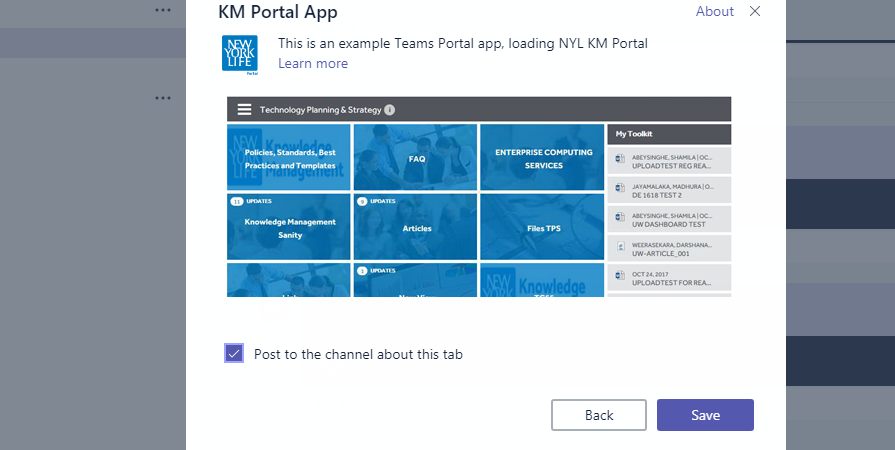
1. Click the (+) sign



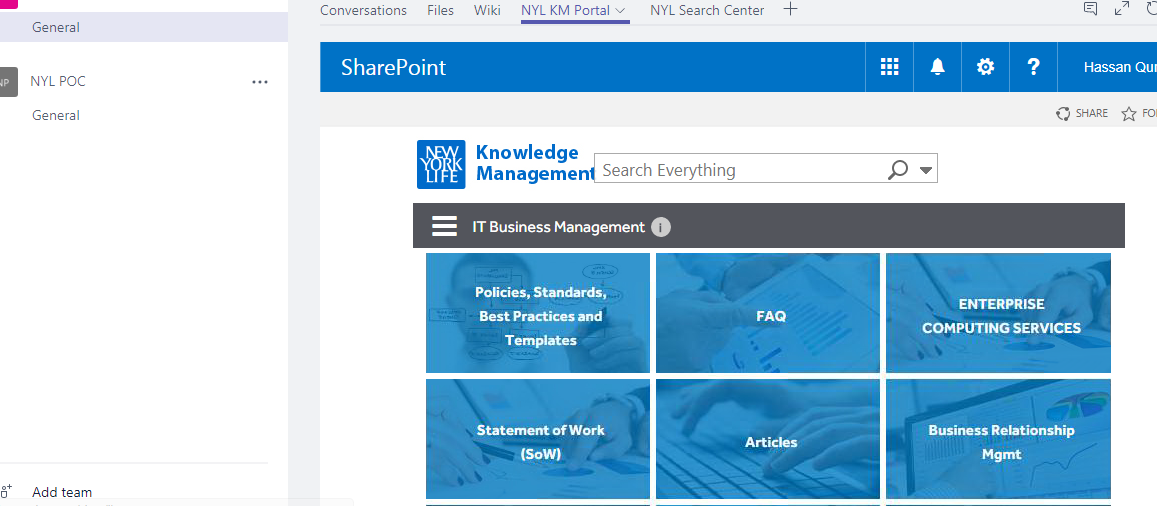
1. Custom Tabs are displayed in the Microsoft Teams.



1. Select the app and save it.



1. Tabs has been deployed into the Microsoft Teams.



## References:

<https://msdn.microsoft.com/en-us/microsoft-teams/sideload>

<https://msdn.microsoft.com/en-us/microsoft-teams/createpackage>

<https://msdn.microsoft.com/en-us/microsoft-teams/prerequisites>

<https://msdn.microsoft.com/en-us/Microsoft-teams/createconfigpage>

# Custom Tile for O365 App Launcher

## Introduction

You can add your own custom Tiles to the app launcher that point to SharePoint sites, external sites, legacy apps, and more. The custom Tile appears under the app launcher's All apps, but you can pin it to the Home apps.

**Note :  Still O365 doesn’t support Admin user to pin custom Tiles to all users, but it’s in the O365 roadmap.**

<https://products.office.com/en-us/business/office-365-roadmap>

## POC Task

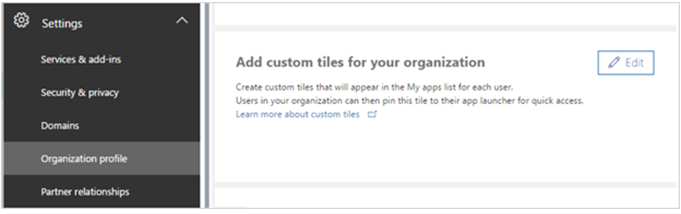


Steps

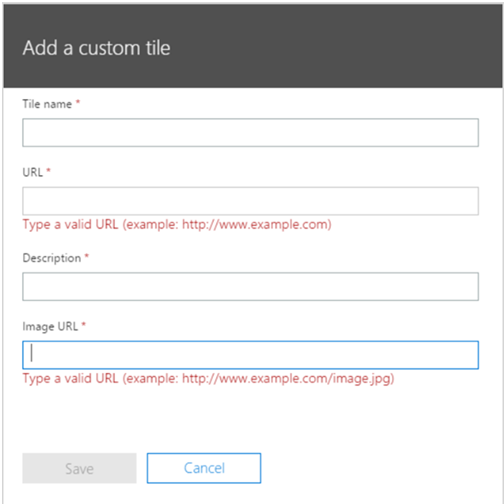
1. Add custom Tile
2. Promote the Tile to the Home tab
3. Rearrange custom Tile

## Add custom Tile

1. Go to O365 **Admin** center.
2. Choose **Settings** > **Organization profile** > **Add custom tiles for your organization**.



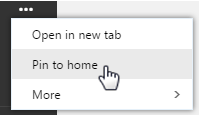
1. Choose + **Add a custom tile**.



1. Enter a **Tile name** for the new tile. The name will appear in the tile.
2. Enter a **URL** for the tile. This is the location where you want your users to go when they click the tile on the app launcher.
3. Enter a **Description** for the tile. You see this when you select the tile on the My apps page and choose App details.
4. Enter an **Image URL** for the tile. The image appears on the apps page and app launcher.
5. Choose **Save** to create the custom tile.

## Promote the Tile to the Home tab

1. Select the app launcher icon and select the **Home** tab.
2. Locate the new tile for your app, select the ellipsis, and choose **Pin to home.**



## Rearrange custom Tile

1. Choose the Tile that you want to move.
2. Drag it to where you want it to appear on the app launcher.

## References:

<https://support.office.com/en-us/article/Add-custom-tiles-to-the-app-launcher-1136115a-75af-4497-b693-640c4ce70bc6>

<https://support.office.com/en-us/article/Change-the-order-of-your-tiles-on-the-Office-365-app-launcher-07a83bef-9710-4e3d-851d-419dd04201fd?ui=en-US&rs=en-US&ad=US>

# Appendix



## OneDrive Sample Manifest file



## Outlook Sample Manifest file



## Word Sample Manifest file



## Teams Sample Page

