Microsoft Office 365 PowerPoint - Web Add-in - Consulting Practice

Cognitive Convergence is Subject Matter Expert in Office 365, Dynamics 365, SharePoint, Project Server, Power Platform: Power Apps-Power BI-Power Automate-Power Virtual Agents.

Our Microsoft Office 365 Consulting, add-in Development, Customization, Integration services and solutions, can help companies maximize business performance, overcoming market challenges, achieving profitability and providing best customer service.

CONTENTS

OBJECTIVE

In this case study, a brief introduction will be discussed about the Microsoft Office 365 PowerPoint Web Add-in, its basic structure & architecture components.

INTRODUCTION

Users can use PowerPoint add-ins to build engaging solutions for the users of the company for the presentations across platforms including Windows, iPad, Mac, and in a browser. Users can create two types of PowerPoint add-ins:

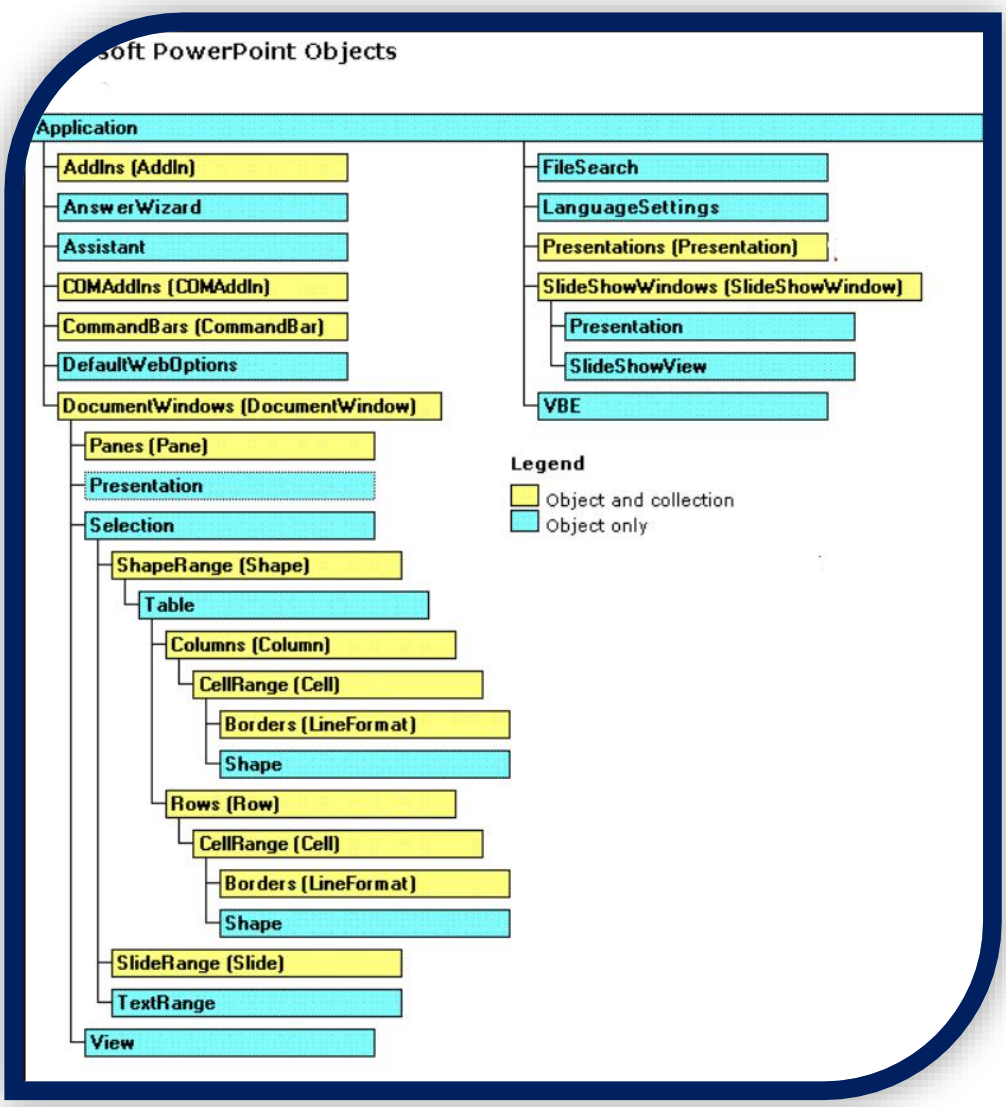
- Use **content add-ins** to add dynamic HTML5 content to your presentations.
- Use **task pane add-ins** to bring in reference information or insert data into the presentation via a service.

POWERPOINT ADD-IN SCENARIOS

The code examples in this article demonstrate some basic tasks for developing add-ins for PowerPoint. Please note the following:

1. To display information, these examples use the app.showNotification function, which is included in the Visual Studio Office Add-ins project templates. If you aren't using Visual Studio to develop your add-in, you'll need replace the showNotification function with your own code.
2. Several of these examples also use a Globals object that is declared beyond the scope of these functions as: var Globals = {activeViewHandler:0, firstSlideId:0};

POWERPOINT ADDIN OBJECT MODEL



DETECT THE PRESENTATION'S ACTIVE VIEW AND HANDLE THE ACTIVEVIEWCHANGED EVENT

In the following code sample:

- The **getActiveFileView** function calls the **Document.getActiveViewAsync** method to return whether the presentation's current view is "edit" (any of the views in which you can edit slides, such as **Normal** or **Outline View**) or "read" (**Slide Show** or **Reading View**).
- The **registerActiveViewChanged** function calls the **addHandlerAsync** method to register a handler for the **Document.ActiveViewChanged** event.

```
eral Office.initialize function. Fires on load of the add-in.
fice.initialize = function(){

    //Gets whether the current view is edit or read.
    var currentView = getActiveFileView();

    //register for the active view changed handler
    registerActiveViewChanged();

    //render the content based off of the currentView
    //....
}

function getActiveFileView()
{
    Office.context.document.getActiveViewAsync(function (asyncResult) {
        if (asyncResult.status == "failed") {
            app.showNotification("Action failed with error: " + asyncResult.error.message);
        }
        else {
            app.showNotification(asyncResult.value);
        }
    });

}

function registerActiveViewChanged() {
    Globals.activeViewHandler = function (args) {
        app.showNotification(JSON.stringify(args));
    }

    Office.context.document.addHandlerAsync(Office.EventType.ActiveViewChanged, Globals.activeViewHandler,
        function (asyncResult) {
            if (asyncResult.status == "failed") {
                app.showNotification("Action failed with error: " + asyncResult.error.message);
            }
            else {
                app.showNotification(asyncResult.status);
            }
        });
}
```

NAVIGATE TO A PARTICULAR SLIDE IN THE PRESENTATION

In the following code sample, the getSelectedRange function calls the **Document.getSelectedDataAsync** method to get the JSON object returned by asyncResult.value, which contains an array named slides. The slides array contains the ids, titles, and indexes of selected range of slides (or of the current slide, if multiple slides are not selected). It also saves the id of the first slide in the selected range to a global variable.

```
function getSelectedRange() {
    // Get the id, title, and index of the current slide (or selected slides) and store the first slide id */
    Globals.firstSlideId = 0;

    Office.context.document.getSelectedDataAsync(Office.CoercionType.SlideRange, function (asyncResult) {
        if (asyncResult.status == "failed") {
            app.showNotification("Action failed with error: " + asyncResult.error.message);
        }
        else {
            Globals.firstSlideId = asyncResult.value.slides[0].id;
            app.showNotification(JSON.stringify(asyncResult.value));
        }
    });
}
```

In the following code sample, the goToFirstSlide function calls the **Document.goToByIdAsync** method to navigate to the first slide that was identified by the getSelectedRange function shown previously.

```
function goToFirstSlide() {
    Office.context.document.goToByIdAsync(Globals.firstSlideId, Office.GoToType.Slide, function (asyncResult) {
        if (asyncResult.status == "failed") {
            app.showNotification("Action failed with error: " + asyncResult.error.message);
        }
        else {
            app.showNotification("Navigation successful");
        }
    });
}
```

NAVIGATE BETWEEN SLIDES IN THE PRESENTATION

In the following code sample, the **goToSlideByIndex** function calls the **Document.goToByIdAsync** method to navigate to the next slide in the presentation.

```
function goToSlideByIndex() {
    var goToFirst = Office.Index.First;
    var goToLast = Office.Index.Last;
    var goToPrevious = Office.Index.Previous;
    var goToNext = Office.Index.Next;

    Office.context.document.goToByIdAsync(goToNext, Office.GoToType.Index, function (asyncResult) {
        if (asyncResult.status == "failed") {
            app.showNotification("Action failed with error: " + asyncResult.error.message);
        }
        else {
            app.showNotification("Navigation successful");
        }
    });
}
```

Get the URL of the presentation

In the following code sample, the getFileUrl function calls the **Document.getFileProperties** method to get the URL of the presentation file.

```
function getFileUrl() {
    //Get the URL of the current file.
    Office.context.document.getFilePropertiesAsync(function (asyncResult) {
        var fileUrl = asyncResult.value.url;
        if (fileUrl == "") {
            app.showNotification("The file hasn't been saved yet. Save the file and try again");
        }
        else {
            app.showNotification(fileUrl);
        }
    });
}
```

CREATE A PRESENTATION

User's add-in can create a new presentation, separate from the PowerPoint instance in which the add-in is currently running. The PowerPoint namespace has the **createPresentation** method for this purpose. When this method is called, the new presentation is immediately opened and displayed in a new instance of PowerPoint. Your add-in remains open and running with the previous presentation.

```javascript
PowerPoint.createPresentation();
```

The createPresentation method can also create a copy of an existing presentation. The method accepts a base64-encoded string representation of an .pptx file as an optional parameter. The resulting presentation will be a copy of that file, assuming the string argument is a valid .pptx file. The **FileReader** class can be used to convert a file into the required base64-encoded string, as demonstrated in the following example.

```javascript
var myFile = document.getElementById("file");
var reader = new FileReader();

reader.onload = function (event) {
    // strip off the metadata before the base64-encoded string
    var startIndex = reader.result.toString().indexOf("base64,");
    var copyBase64 = reader.result.toString().substr(startIndex + 7);

    PowerPoint.createPresentation(copyBase64);
};

// read in the file as a data URL so we can parse the base64-encoded string
reader.readAsDataURL(myFile.files[0]);
```

CREATE THE ADD-IN

Users can create an Office Add-in by using the Yeoman generator for Office Add-ins or Visual Studio. The Yeoman generator creates a Node.js project that can be managed with Visual Studio Code or any other editor, whereas Visual Studio creates a Visual Studio solution.

METHOD 1 - OFFICE 365 POWERPOINT WEB ADDIN - YEOMAN GENERATOR – VISUAL CODE

Prerequisites

1. Node.js (the latest LTS version)
2. The latest version of Yeoman and the Yeoman generator for Office Add-ins. To install these tools globally,

```
npm install -g yo generator-office
```

run the following command via the command prompt:

Create the add-in project

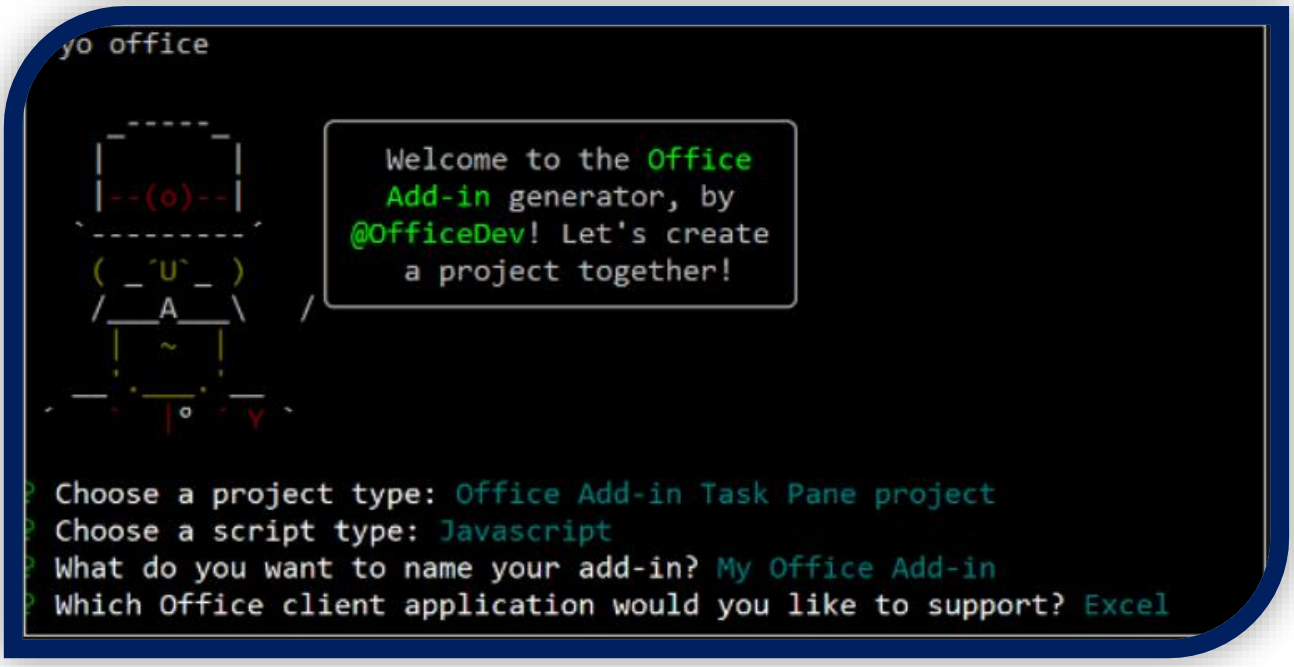
1. Run the following command to create an add-in project using the Yeoman.

```
command line

yo office
```

2. When prompted, provide the following information to create your add-in project:

   - **Choose a project type:** Office Add-in Task Pane project
   - **Choose a script type**: Javascript
   - **What do you want to name your add-in?** My Office Add-in



   - **Which Office client application would you like to support?** Excel

   After completing the wizard, the generator creates the project and installs supporting Node components.

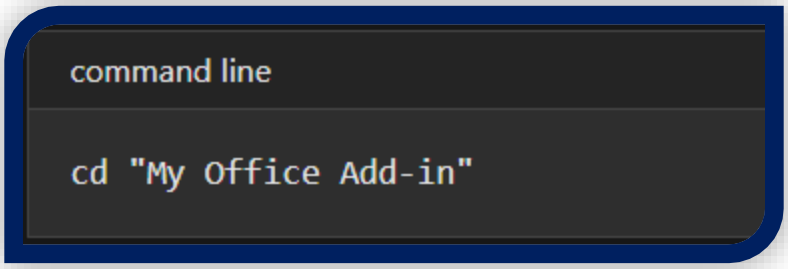
Explore the project

The add-in project that the user has just created with the Yeoman generator contains sample code for a very basic task pane add-in. To explore the components of the add-in project, open the project in the code editor, and review the files listed below.

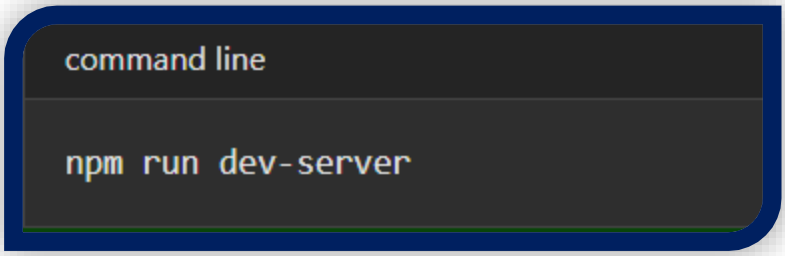
- The ./manifest.xml file in the root directory of the project defines the settings and capabilities of the add-in.
- The ./src/taskpane/taskpane.html file contains the HTML markup for the task pane.
- The ./src/taskpane/taskpane.css file contains the CSS that's applied to content in the task pane.
- The ./src/taskpane/taskpane.js file contains the Office JavaScript API code that facilitates interaction between the task pane and the Office host application.

Try it out

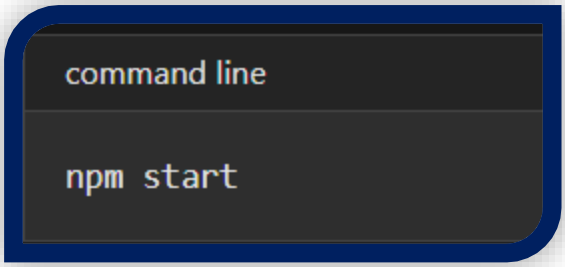
1. Navigate to the root folder of the project.



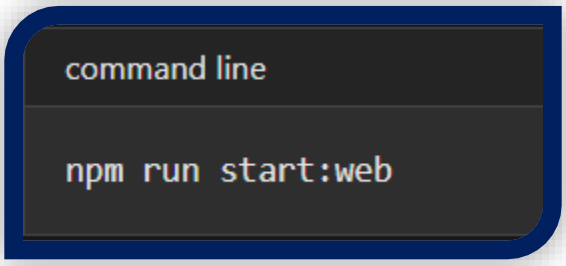
2. Complete the following steps to start the local web server and sideload your add-in.



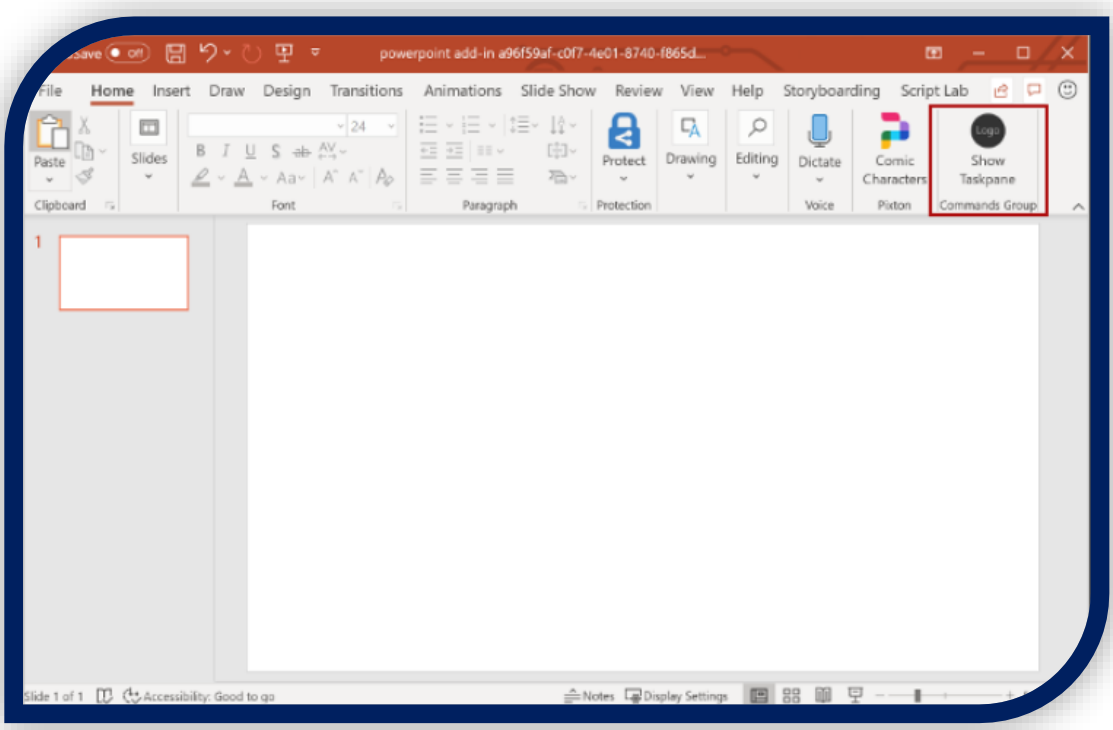
   a. To test the add-in in Excel, run the following command in the root directory of the project. This starts the local webserver (if it's not already running) and opens PowerPoint with the add-in loaded.



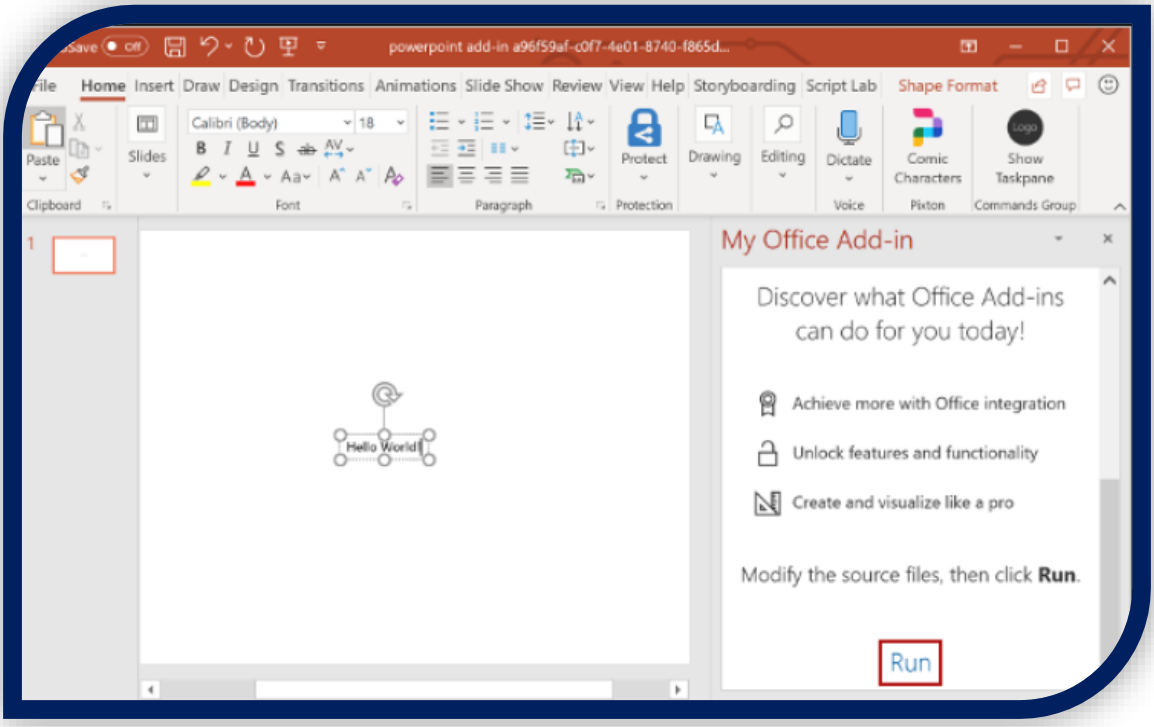
   b. To test the add-in in Excel on a browser, run the following command in the root directory of the project. When this command runs, the local web server will start (if it's not already running).



3. In PowerPoint, insert a new blank slide, choose the Home tab, and then choose the Show Taskpane button in the ribbon to open the add-in task pane.



4. At the bottom of the task pane, choose the Run link to insert the text "Hello World" into the current slide.



METHOD 2 - OFFICE 365 POWERPOINT WEB ADDIN - VISUAL STUDIO

Prerequisites

a. Visual Studio 2019 with the Office/SharePoint development workload installed
b. Office 2016 or later

Create the add-in project

1. In Visual Studio, choose Create a new project.
2. Using the search box, enter add-in. Choose PowerPoint Web Add-in, then select Next.
3. Name your project and select Create.
4. In the Create Office Add-in dialog window, choose Add new functionalities to PowerPoint, and then choose Finish to create the project.
5. Visual Studio creates a solution and its two projects appear in Solution Explorer. The Home.html file opens in Visual Studio.

Explore the Visual Studio solution

When you've completed the wizard, Visual Studio creates a solution that contains two projects.

Project	Description
Add-in project	Contains only an XML manifest file, which contains all the settings that describe your add-in. These settings help the Office host determine when your add-in should be activated and where the add-in should appear. Visual Studio generates the contents of this file for you so that you can run the project and use your add-in immediately. You change these settings any time by modifying the XML file.
Web application project	Contains the content pages of your add-in, including all the files and file references that you need to develop Office-aware HTML and JavaScript pages. While you develop your add-in, Visual Studio hosts the web application on your local IIS server. When you're ready to publish the add-in, you'll need to deploy this web application project to a web server.

Update the code

1. **Home.html** specifies the HTML that will be rendered in the add-in's task pane. In **Home.html**, replace the <body> element with the following markup and save the file.

```
<body class="ms-font-m ms-welcome">
    <div id="content-header">
        <div class="padding">
            <h1>Welcome</h1>
        </div>
    </div>
    <div id="content-main">
        <div class="padding">
            <p>Select a slide and then choose the buttons to below to add content to it.</p>
            <br />
            <h3>Try it out</h3>
            <button class="ms-Button" id="insert-image">Insert Image</button>
            <br/><br/>
            <button class="ms-Button" id="insert-text">Insert Text</button>
        </div>
    </div>
</body>
```

2. Open the file **Home.js** in the root of the web application project. This file specifies the script for the add-in.

```
'strict';

(function () {

    Office.onReady(function() {
        // Office is ready
        $(document).ready(function () {
            // The document is ready
            $('#insert-image').click(insertImage);
            $('#insert-text').click(insertText);
        });
    });

    function insertImage() {
        Office.context.document.setSelectedDataAsync(getImageAsBase64String(), {
            coercionType: Office.CoercionType.Image,
            imageLeft: 50,
            imageTop: 50,
            imageWidth: 400
        },
            function (asyncResult) {
                if (asyncResult.status === Office.AsyncResultStatus.Failed) {
                    console.log(asyncResult.error.message);
                }
            });
    }

    function insertText() {
        Office.context.document.setSelectedDataAsync("Hello World!",
            function (asyncResult) {
                if (asyncResult.status === Office.AsyncResultStatus.Failed) {
                    console.log(asyncResult.error.message);
                }
            });
    }

})();
```

Replace the entire contents with the following code and save the file.

3. Open the file **Home.css** in the root of the web application project. This file specifies the custom styles for the add-in. Replace the entire contents with the following code and save the file.

```css
#content-header {
    background: #2a8dd4;
    color: #fff;
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 80px;
    overflow: hidden;
}

#content-main {
    background: #fff;
    position: fixed;
    top: 80px;
    left: 0;
    right: 0;
    bottom: 0;
    overflow: auto;
}

.padding {
    padding: 15px;
}
```

Update the manifest

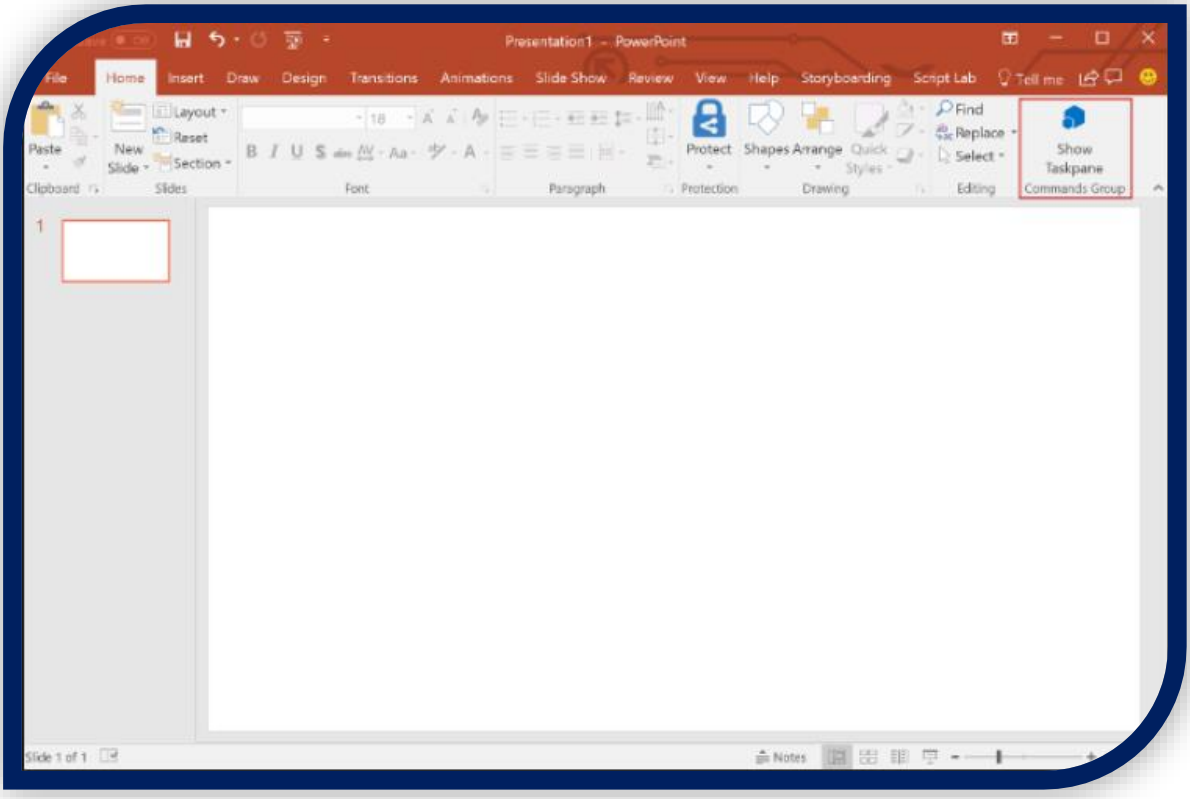
1. Open the XML manifest file in the add-in project. This file defines the add-in's settings and capabilities.
2. The ProviderName element has a placeholder value. Replace it with your name.
3. The DefaultValue attribute of the DisplayName element has a placeholder. Replace it with **My Office Add-in**.
4. The DefaultValue attribute of the Description element has a placeholder. Replace it with **A task pane add-in for PowerPoint**.
5. Save the file

```
...
<ProviderName>John Doe</ProviderName>
<DefaultLocale>en-US</DefaultLocale>
<!-- The display name of your add-in. Used on the store and various places of the (
<DisplayName DefaultValue="My Office Add-in" />
<Description DefaultValue="A task pane add-in for PowerPoint"/>
...
```

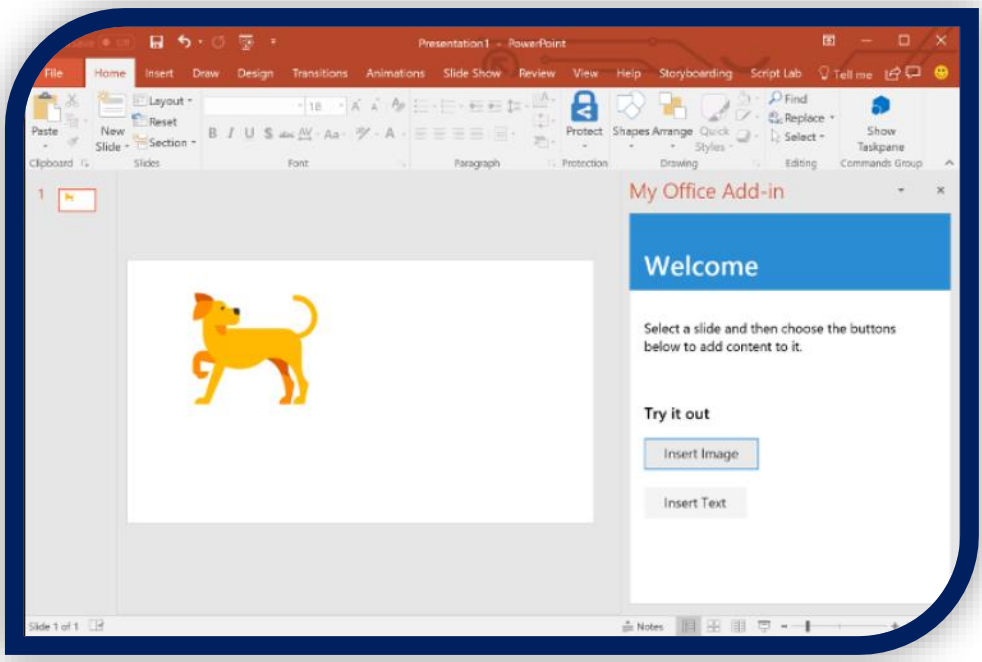
Try it out

1. Using Visual Studio, test the newly created PowerPoint add-in by pressing **F5** or choosing the **Start** button to launch PowerPoint with the **Show Taskpane** add-in button displayed in the ribbon. The add-in will be hosted locally on IIS.

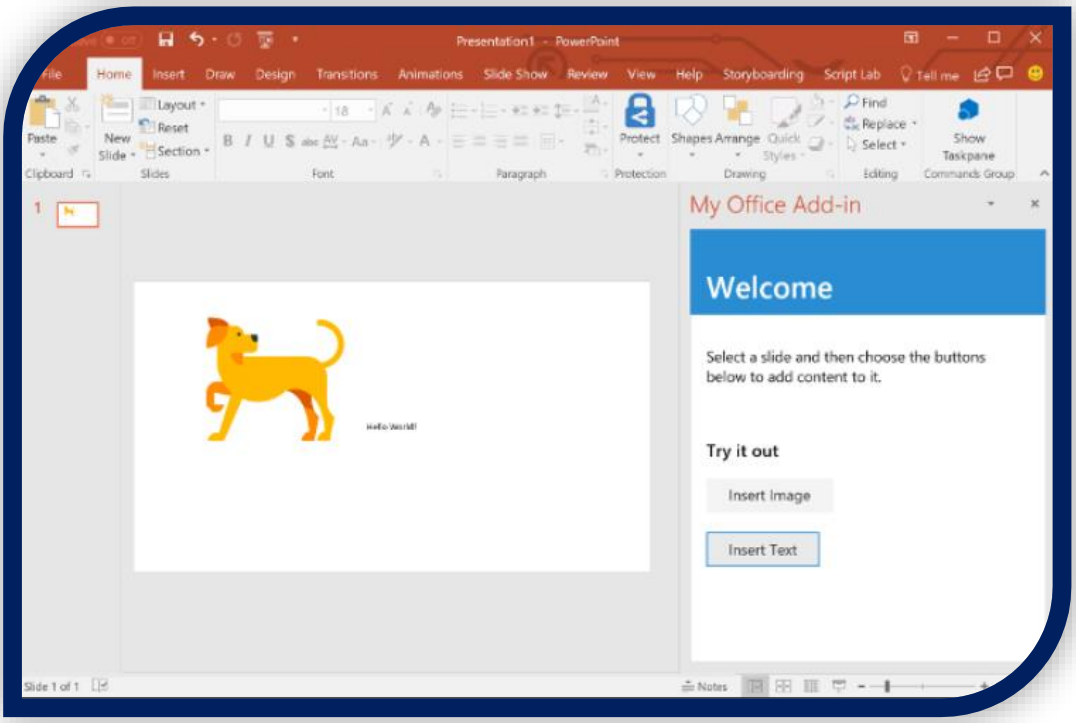
2. In PowerPoint, insert a new blank slide, choose the **Home** tab, and then choose the **Show Taskpane** button in the ribbon to open the add-in task pane.



3. In the task pane, choose the Insert Image button to add an image to the selected slide.



4. In the task pane, choose the **Insert Text** button to add text to the selected slide.



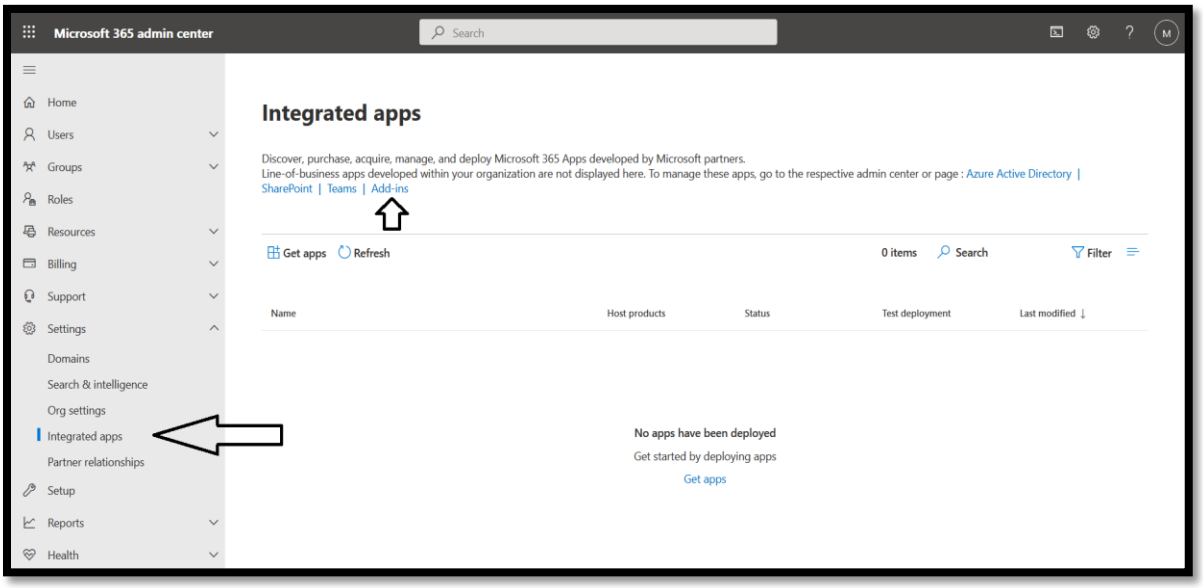
DEPLOYMENT OF ADD-IN

There are multiple methods of deployment but we have followed the Centralized Deployment.

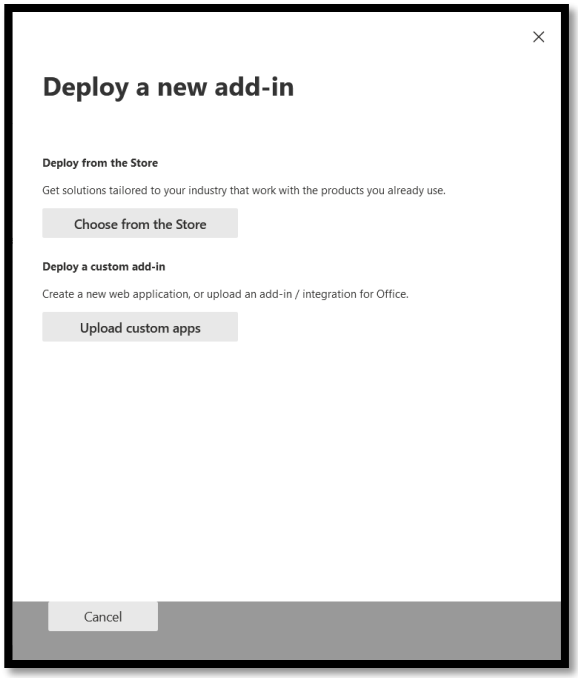
CENTRALIZED DEPLOYMENT

Follow steps below to publish an Office Add-in via Centralized Deployment:

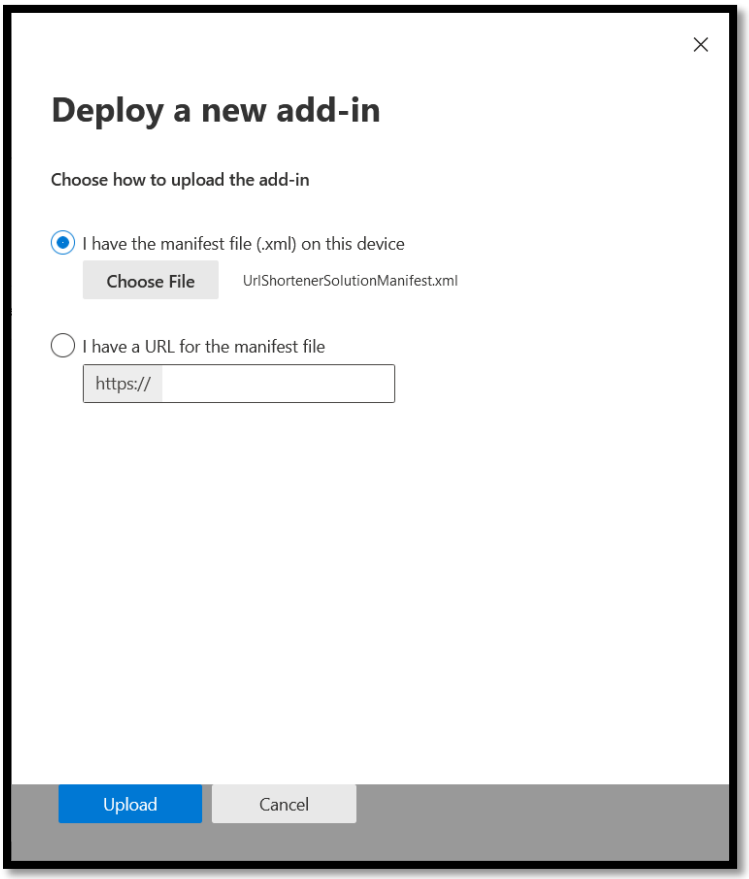
1. Sign in to Microsoft 365 with your work or education account.
2. Select the app launcher icon in the upper-left and choose **Admin**.
3. In the navigation menu, press **Show more**, then choose **Settings** > **Integrated apps**.
4. Choose **Add-ins** at the top of the page



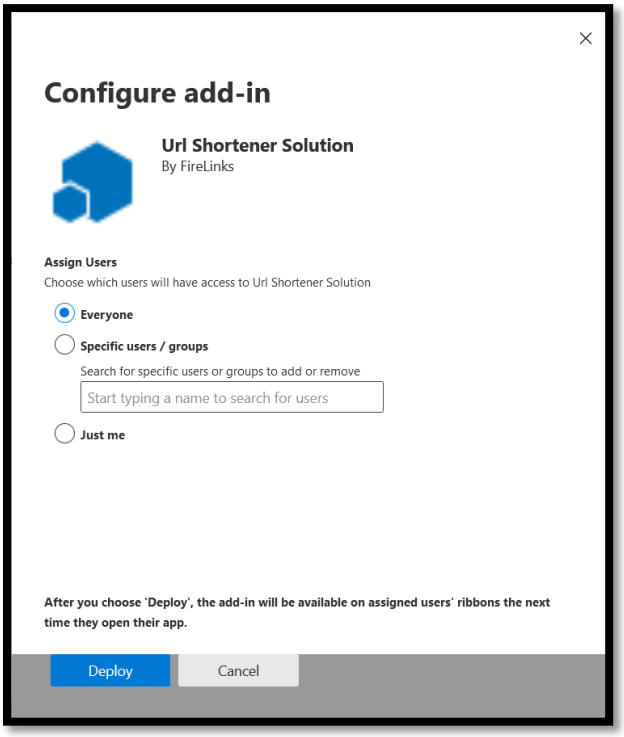
5. Choose **Deploy Add-In** at the top of the page.
6. Choose **Next** after reviewing the requirements.
7. Choose **Upload Custom apps** in the following page



8. Click on **Choose file** and select Add-in manifest file (.xml)
9. Choose **Upload** at the end of the task pane



10. On the **Assign Users** page, choose **Everyone**, **Specific Users/Groups**, or **Only me**. Use the search box to find the users and groups to whom you want to deploy the add-in.

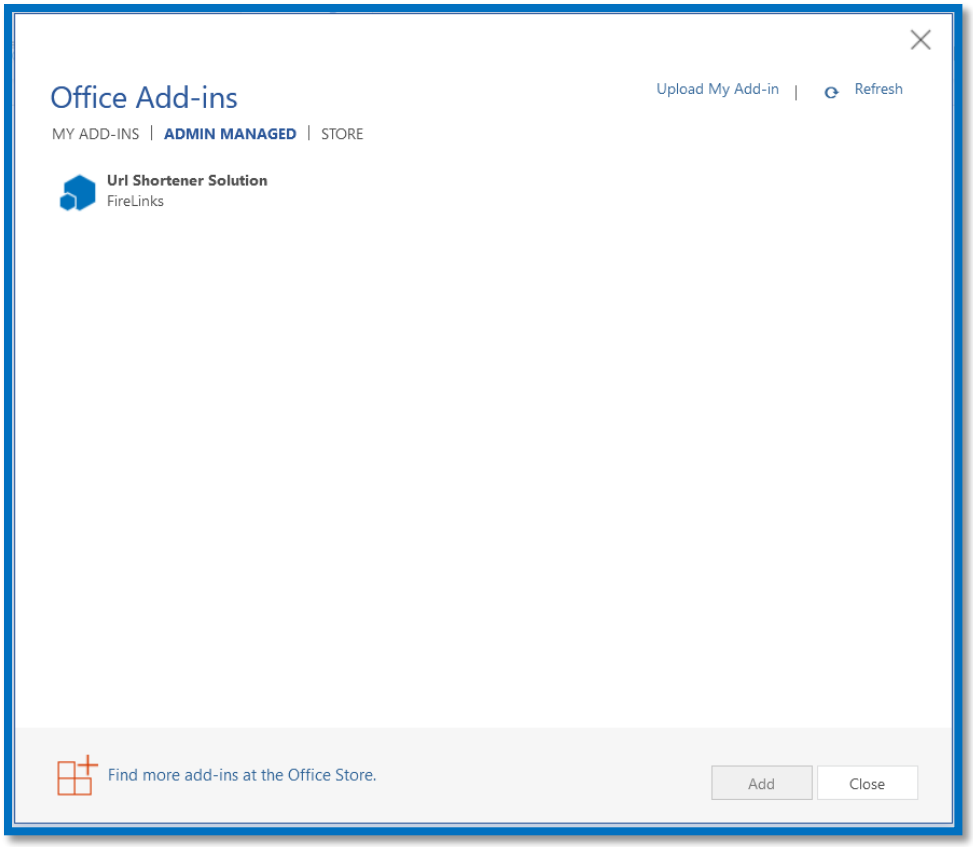


11. When finished, choose **Deploy**. This process may take up to three minutes. Then, finish the walkthrough by pressing **Next**.

DEPLOY ON WEB

Now the add-in has been deployed in your tenant and you can test it on PowerPoint web application following steps below:

1) Open a **new blank document** in PowerPoint web application
2) Go to **Insert** tab
3) Choose **Add-ins** in top navigation
4) Choose **Admin managed** in the top of Office Add-ins modal
5) Select your deployed add-in from available list
6) Choose **Add** at the end of the modal



DEPLOY ON DESKTOP

You can test the solution on desktop PowerPoint application by following the steps below:

1) Open a **new blank document** in desktop PowerPoint application
2) Select **File** > **Account**
3) If you're not already signed in, click **Sign In**, else choose **Switch account**
4) Choose **Sign-in with a different account**
5) Provide **login** credentials of Office account on which add-in has been deployed in Office 365
6) Follow the steps 2 to 6 from heading "Test on Web"

EXAPMLES OF POWERPOINT ADDIN

Lucidchart Diagrams for PowerPoint

Company: Lucid Software Inc

App Source: URL

Description: Lucidchart is a visual workspace that combines diagramming, data visualization, and collaboration to accelerate understanding and drive innovation. With this intuitive, cloud-based solution, everyone can work visually and collaborate in real-time while building flowcharts, mockups, UML diagrams, and more.

Global compatibility:

1. Integrates with Microsoft Office, Microsoft Teams, Salesforce, Slack, Confluence, Jira, and more
2. Imports Visio, OmniGraffle, Gliffy, and draw.io files
3. Runs on all major browsers
4. Shape libraries for every scenario:
5. Flowcharts and process maps
6. AWS, Azure, and GCP shapes
7. Mockups and wireframes
8. UML, ER, and network diagrams
9. Mind maps and Venn diagrams
10. Org charts and BPMN diagrams

Pexels - Free Stock Photos

Company: OfficeConsult AS

App Source: URL

Description: Best free stock photos in one place. Create beautiful documents and presentations with free, professional stock photos.

1. Search for images
2. View popular images
3. Save images as favorites for easy access

When this add-in is used, it

1. Can read and make changes to your document
2. Can send data over the Internet

BigMarker PowerPoint Uploader

Company: BigMarker.com

App Source: URL

Description: The BigMarker PowerPoint Uploader Add-In allows users to upload PowerPoint presentations from their PC or PowerPoint Online directly to their events on BigMarker. Through the add-in, users can log into their BigMarker account, view their upcoming webinars and online events, and upload their PowerPoint presentation directly to a specific event.

BigMarker is a place to host live online talks, classes, presentations, webinars and webcasts, from your computer to the world. It provides all the tools you need to make your event successful, from scheduling and promoting your

event, to audience registration and live hosting. BigMarker handles it all, including landing pages, ticket sales, lead generation, event reminders, audience communication, recording, building your audience, and hosting up to 1,000 attendees.

CONCLUSION

In this case study, a brief introduction about the Microsoft Office 365 PowerPoint Web add-in, its architecture, development & deployment ways & a demo of the introductory level is discussed in detail.

Our Microsoft Office 365 PowerPoint Consulting, add-in Development, Customization, Integration services, and solutions, can help companies maximize business performance, overcoming market challenges, achieving profitability, and providing the best customer care service.

CONTACT US

Shahzad Sarwar

Cognitive Convergence Team