Microsoft Office 365 OneNote 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 OneNote Web-Addin 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

This case study is a brief introduction about the core concept of OneNote Add-in, its fundamentals, building architecture & its development tracks.

INTRODUCTION

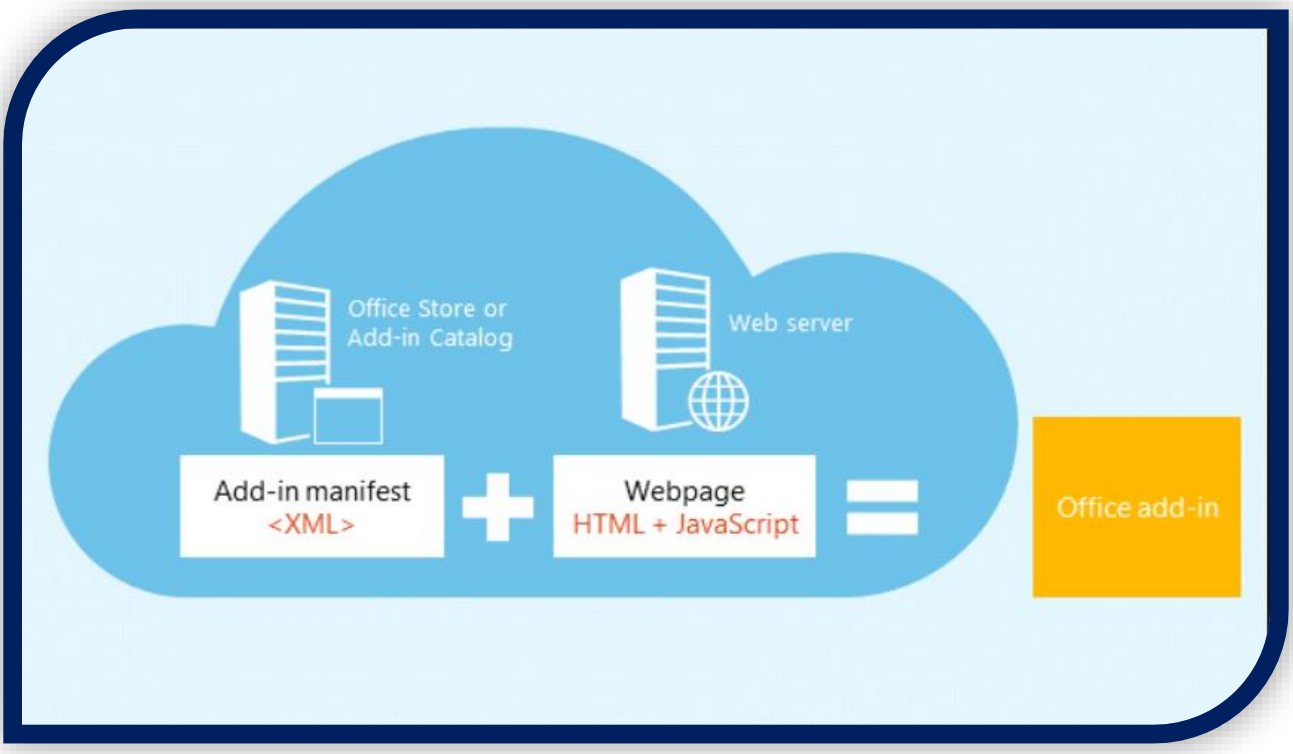
OneNote introduces a JavaScript API for OneNote add-ins on the web. Users can create task pane add-ins, content add-ins, and add-in commands that interact with OneNote objects and connect to web services or other web-based resources.

COMPONENTS OF AN OFFICE ADD-IN

Add-ins consist of two basic components:

- A **web application** consisting of a webpage and any required JavaScript, CSS, or other files. These files are hosted on a web server or web hosting service, such as Microsoft Azure. In OneNote on the web, the web application displays in browser control or iframe.
- An **XML manifest** that specifies the URL of the add-in's webpage and any access requirements, settings, and capabilities for the add-in. This file is stored on the client. OneNote add-ins use the same manifest format as other Office Add-ins.

Office Add-in = Manifest + Webpage



USING THE JAVASCRIPT API

Add-ins use the runtime context of the host application to access the JavaScript API. The API has two layers:

- A **host-specific API** for OneNote-specific operation accessed through the Application object.
- A **Common API** that's shared across Office applications, accessed through the Document object.

ACCESSING THE HOST-SPECIFIC API THROUGH THE APPLICATION OBJECT

Use the Application object to access OneNote objects such as **Notebook**, **Section**, and **Page**. With host-specific APIs, you run batch operations on proxy objects. The basic flow goes something like this:

1. Get the application instance from the context.
2. Create a proxy that represents the OneNote object you want to work with. You interact synchronously with proxy objects by reading and writing their properties and calling their methods.
3. Call load on the proxy to fill it with the property values specified in the parameter. This call is added to the queue of commands.
4. Call context.sync to run all queued commands in the order that they were queued. This synchronizes the state between your running script and the real objects, and by retrieving properties of loaded OneNote objects for use in your script. You can use the returned promise object for chaining additional actions.

For example:

```javascript
function getPagesInSection() {
    OneNote.run(function (context) {

        // Get the pages in the current section.
        var pages = context.application.getActiveSection().pages;

        // Queue a command to load the id and title for each page.
        pages.load('id,title');

        // Run the queued commands, and return a promise to indicate task completion.
        return context.sync()
            .then(function () {

                // Read the id and title of each page.
                $.each(pages.items, function(index, page) {
                    var pageId = page.id;
                    var pageTitle = page.title;
                    console.log(pageTitle + ': ' + pageId);
                });
            })
            .catch(function (error) {
                app.showNotification("Error: " + error);
                console.log("Error: " + error);
                if (error instanceof OfficeExtension.Error) {
                    console.log("Debug info: " + JSON.stringify(error.debugInfo));
                }
            });
    });
}
```

ONENOTE JAVASCRIPT API REQUIREMENT SETS

Requirement sets are named groups of API members. Office Add-ins use requirement sets specified in the manifest or uses a runtime check to determine whether an Office host supports APIs that an add-in needs.

ACCESSING THE COMMON API THROUGH THE DOCUMENT OBJECT

Use the Document object to access the Common API, such as the **getSelectedDataAsync** and **setSelectedDataAsync** methods.

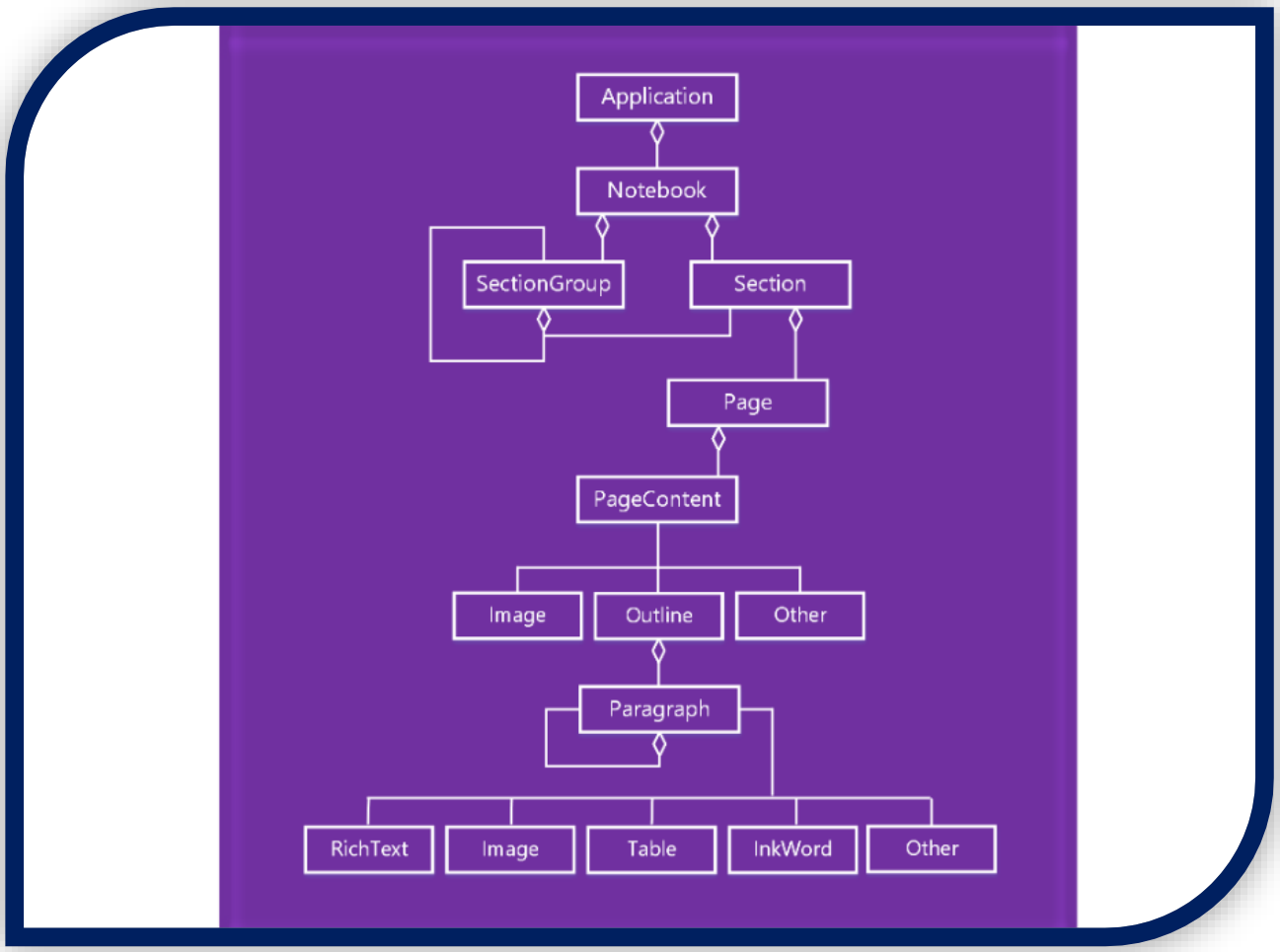
For Example:

```javascript
function getSelectionFromPage() {
    Office.context.document.getSelectedDataAsync(
        Office.CoercionType.Text,
        { valueFormat: "unformatted" },
        function (asyncResult) {
            var error = asyncResult.error;
            if (asyncResult.status === Office.AsyncResultStatus.Failed) {
                console.log(error.message);
            }
            else $('#input').val(asyncResult.value);
        });
}
```

OneNote add-ins support only the following Common APIs:

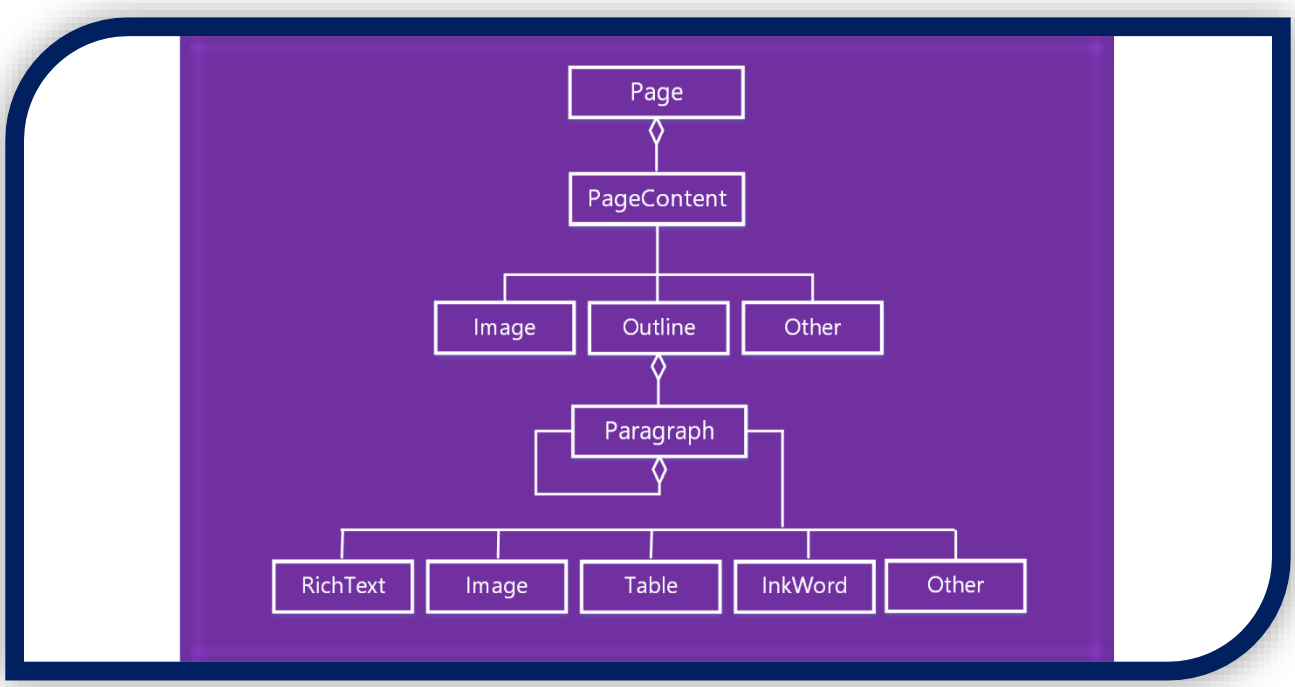
API	Notes
Office.context.document.getSelectedDataAsync	Office.CoercionType.Text and Office.CoercionType.Matrix only
Office.context.document.setSelectedDataAsync	Office.CoercionType.Text, Office.CoercionType.Image, and Office.CoercionType.Html only
var mySetting = Office.context.document.settings.get(name);	Settings are supported by content add-ins only
Office.context.document.settings.set(name, value);	Settings are supported by content add-ins only

ONENOTE OBJECT MODEL DIAGRAM



ONENOTE PAGE CONTENT

In the OneNote add-ins JavaScript API, page content is represented by the following object model.



- A Page object contains a collection of PageContent objects.
- A PageContent object contains a content type of Outline, Image, or Other.
- An Outline object contains a collection of Paragraph objects.
- A Paragraph object contains a content type of RichText, Image, Table, or Other.

The content and structure of a OneNote page are represented by HTML. Only a subset of HTML is supported for creating or updating page content, as described below.

Supported HTML

The OneNote add-in JavaScript API supports the following HTML for creating and updating page content:

- <html>, <body>, , ,
- <p>
- <img>
- <a>
- , <ol>,
- <table>, <tr>, <td>
- <h1> ... <h6>
- <b>, , , <i>, <u>, <del>, <sup>, <sub>, <cite>

OneNote does its best to translate HTML into page content while ensuring security for users. HTML and CSS standards do not exactly match OneNote's content model, so there will be differences in appearances, particularly with CSS stylings. We recommend using the JavaScript objects if specific formatting is needed.

Accessing page contents

You are only able to access *Page Content* via Page#load for the currently active page. To change the active page, invoke navigateToPage($page).

Metadata such as title can still be queried for any page.

ONENOTE PACKAGE

Classes

OneNote.Application	Represents the top-level object that contains all globally addressable OneNote objects such as notebooks, the active notebook, and the active section.
OneNote.FloatingInk	Represents a group of ink strokes.
OneNote.Image	Represents an Image. An Image can be a direct child of a PageContent object or a Paragraph object.
OneNote.InkAnalysis	Represents ink analysis data for a given set of ink strokes.
OneNote.InkAnalysisLine	Represents ink analysis data for an identified text line formed by ink strokes.
OneNote.InkAnalysisLineCollection	Represents a collection of InkAnalysisLine objects.
OneNote.InkAnalysisParagraph	Represents ink analysis data for an identified paragraph formed by ink strokes.
OneNote.InkAnalysisParagraphCollection	Represents a collection of InkAnalysisParagraph objects.
OneNote.InkAnalysisWord	Represents ink analysis data for an identified word formed by ink strokes.
OneNote.InkAnalysisWordCollection	Represents a collection of InkAnalysisWord objects.
OneNote.InkStroke	Represents a single stroke of ink.
OneNote.InkStrokeCollection	Represents a collection of InkStroke objects.
OneNote.InkWord	A container for the ink in a word in a paragraph.
OneNote.InkWordCollection	Represents a collection of InkWord objects.
OneNote.Notebook	Represents a OneNote notebook. Notebooks contain section groups and sections.

OneNote.NotebookCollection	Represents a collection of notebooks.
OneNote.NoteTag	A container for the NoteTag in a paragraph.
OneNote.Outline	Represents a container for Paragraph objects.
OneNote.Page	Represents a OneNote page.
OneNote.PageCollection	Represents a collection of pages.
OneNote.PageContent	Represents a region on a page that contains top-level content types such as Outline or Image. A PageContent object can be assigned an XY position.
OneNote.PageContentCollection	Represents the contents of a page, as a collection of PageContent objects.
OneNote.Paragraph	A container for the visible content on a page. A Paragraph can contain any one ParagraphType type of content.
OneNote.ParagraphCollection	Represents a collection of Paragraph objects.
OneNote.RequestContext	Represents the top-level object that contains all globally addressable OneNote objects such as notebooks, the active notebook, and the active section.
OneNote.RichText	Represents a RichText object in a Paragraph.
OneNote.Section	Represents a OneNote section. Sections can contain pages.
OneNote.SectionCollection	Represents a collection of sections.
OneNote.SectionGroup	Represents a OneNote section group. Section groups can contain sections and other section groups.
OneNote.SectionGroupCollection	Represents a collection of section groups.
OneNote.Table	Represents a table on a OneNote page.
OneNote.TableCell	Represents a cell in a OneNote table.
OneNote.TableCellCollection	Contains a collection of TableCell objects.
OneNote.TableRow	Represents a row in a table.
OneNote.TableRowCollection	Contains a collection of TableRow objects.

Functions

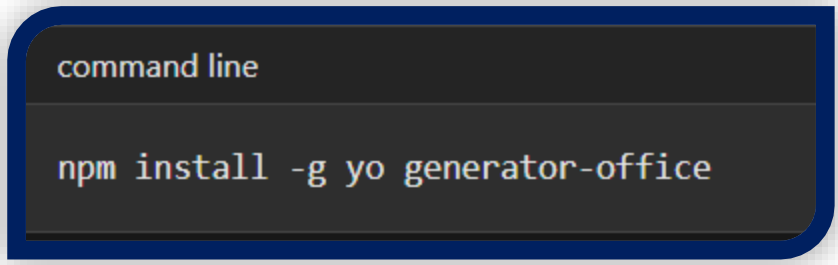
OneNote.run(batch)	Executes a batch script that performs actions on the OneNote object model, using a new request context. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.

OneNote.run(object, batch)	Executes a batch script that performs actions on the OneNote object model, using the request context of a previously-created API object.
OneNote.run(objects, batch)	Executes a batch script that performs actions on the OneNote object model, using the request context of previously-created API objects.

OFFICE 365 ONENOTE WEB ADDIN YEOMAN GENERATOR - VISUAL CODE

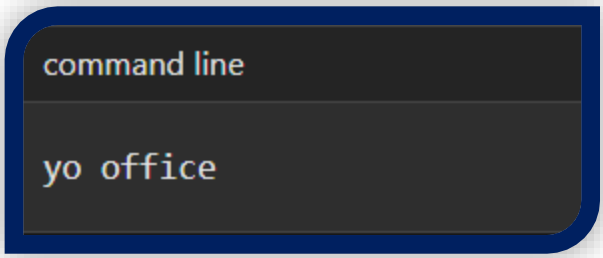
Prerequisites

- Node.js (the latest LTS version)
- The latest version of Yeoman and the Yeoman generator for Office Add-ins. To install these tools globally, run the following command via the command prompt:



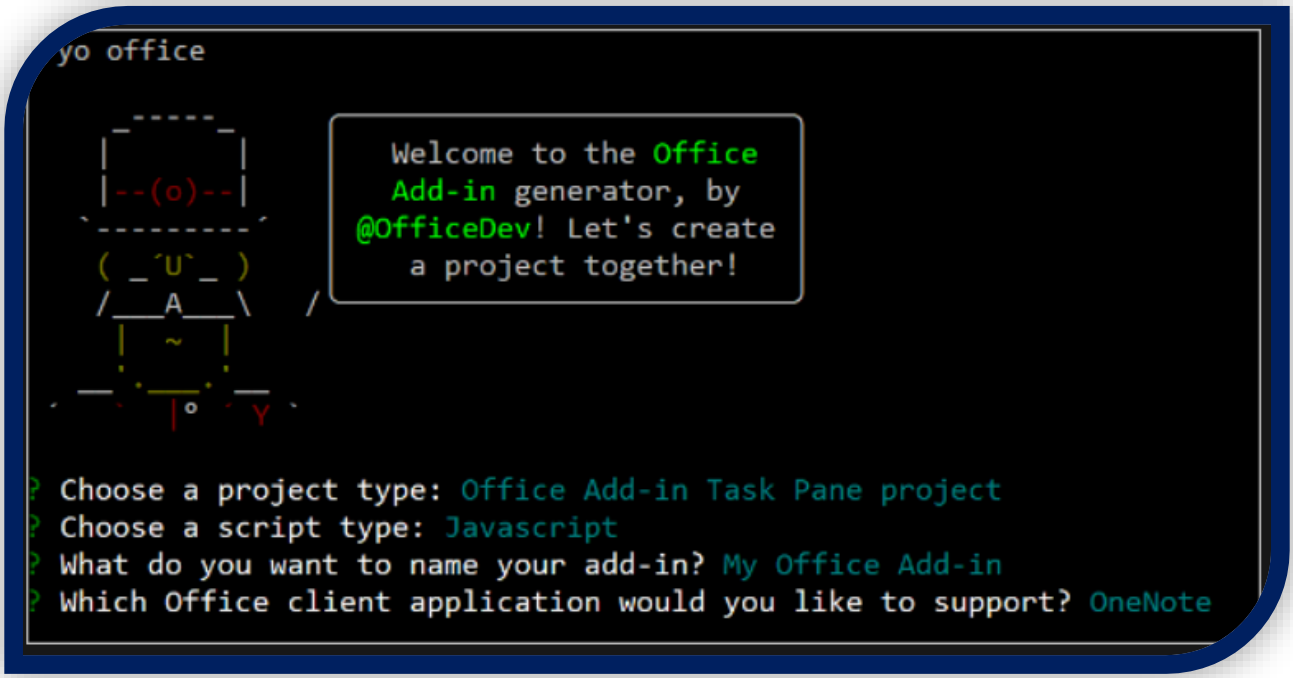
Create the add-in project

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



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?** OneNote



3. After the user completes the wizard, the generator creates the project and installs supporting Node components.

Explore the project

The add-in project that you've created with the Yeoman generator contains sample code for a very basic task pane add-in.

- 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.

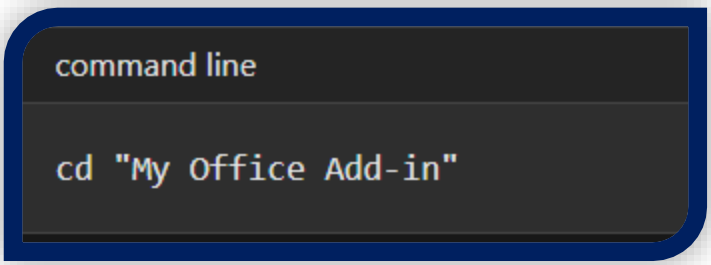
Update the code

In the code editor, open the file ./src/taskpane/taskpane.js and add the following code within the run function. This code uses the OneNote JavaScript API to set the page title and add an outline to the body of the page.

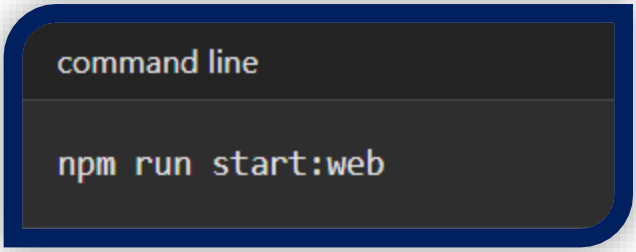
```javascript
try {
    await OneNote.run(async context => {

        // Get the current page.
        var page = context.application.getActivePage();

        // Queue a command to set the page title.
        page.title = "Hello World";

        // Queue a command to add an outline to the page.
        var html = "<p><ol><li>Item #1</li><li>Item #2</li></ol></p>";
        page.addOutline(40, 90, html);

        // Run the queued commands, and return a promise to indicate task completion.
        return context.sync();
    });
} catch (error) {
    console.log("Error: " + error);
}
```

Try it out

1. Navigate to the root folder of the project.



2. Start the local web server and sideload the add-in. Run the following command in the root directory of the project. When the user runs this command, the local web server will start (if it's not already running).
3. In OneNote on the web, open a notebook and create a new page.

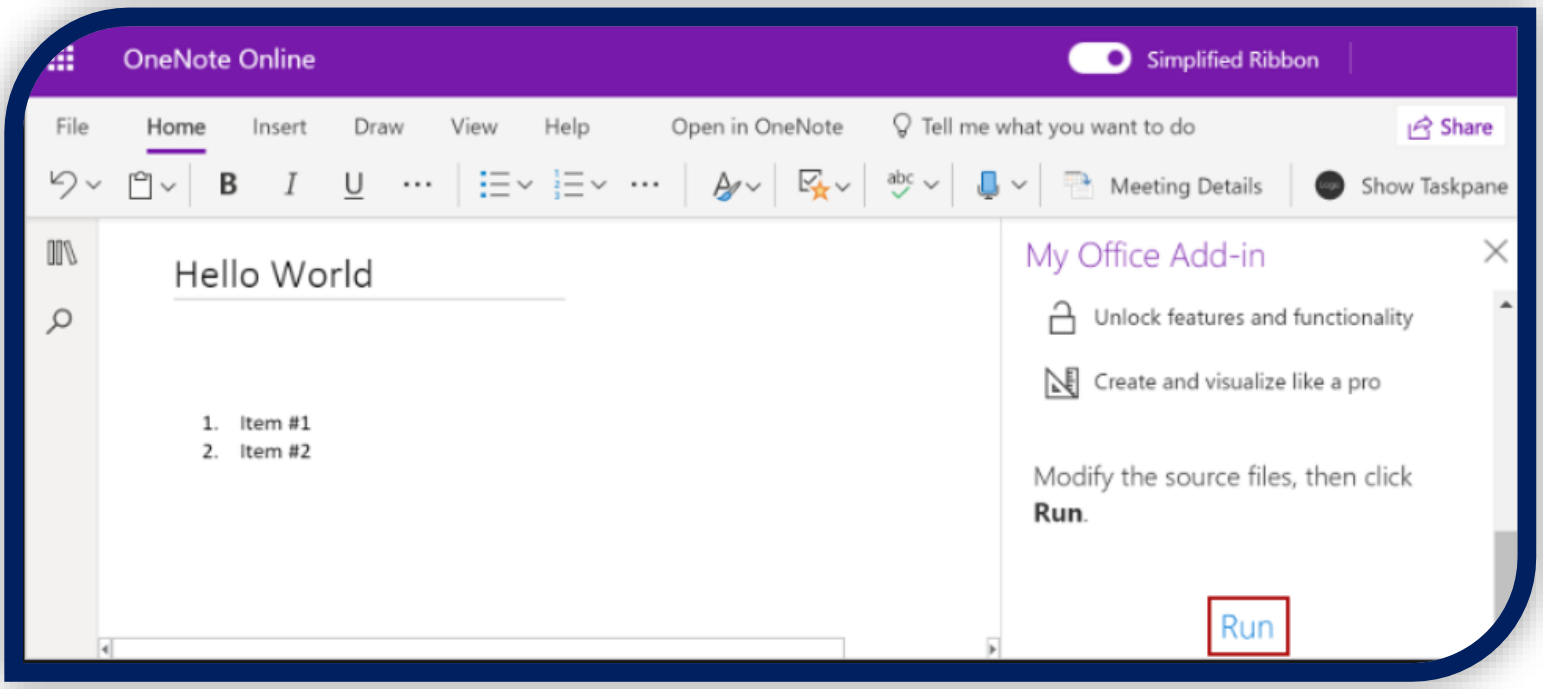


4. Choose Insert > Office Add-ins to open the Office Add-ins dialog.
    I. If the user is signed in with the consumer account, select the MY ADD-INS tab, and then choose to Upload My Add-in.
    II. If the user is signed in with the work or education account, select the MY ORGANIZATION tab, and then select Upload My Add-in.

The following image shows the MY ADD-INS tab for consumer notebooks.

5. In the Upload Add-in dialog, browse to manifest.xml in your project folder, and then choose Upload.
6. From the Home tab, choose the Show Taskpane button in the ribbon. The add-in task pane opens in an iFrame next to the OneNote page.

7. At the bottom of the task pane, choose the Run link to set the page title and add an outline to the body of the page.



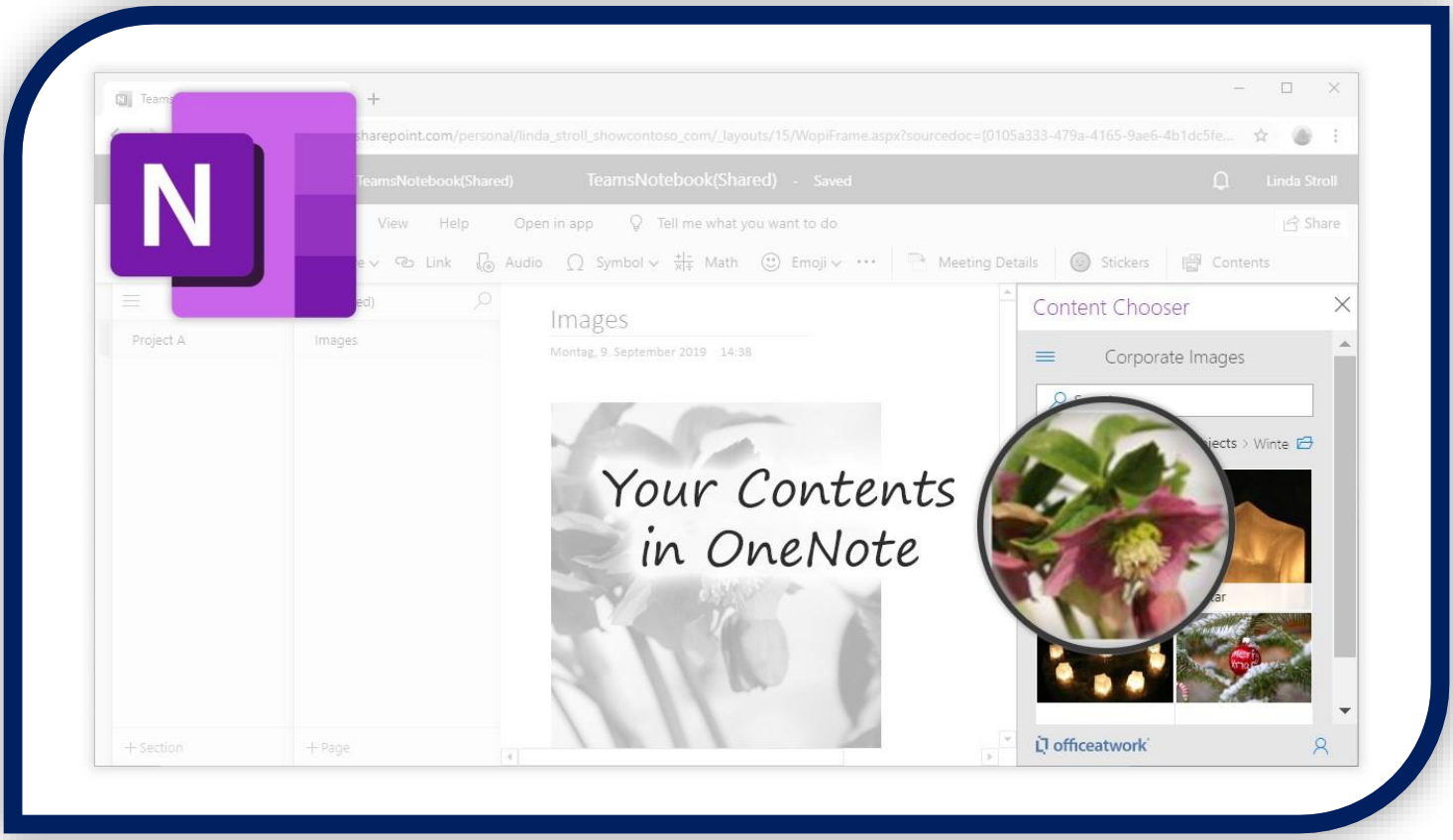
SAMPLE

officeatwork | Content Chooser for Office

Company: officeatwork

Appsource URL: link

Description: Once launched it will present a list of available contents. Inserting a content is as simple as clicking on one of the listed contents. Contents can be organized in multiple libraries to best reflect the organizational needs. Publishing contents for your entire organization is as easy as uploading a content file to a SharePoint Online library. SharePoint will also allow the users to easily manage access rights to confidential contents where needed. Optionally, personal contents can be stored in OneDrive or OneDrive for Business. The Content Chooser is simply the easiest and most efficient way to publish and manage the Office contents. It is available as a service globally and runs in Word, Excel, PowerPoint, OneNote and Outlook.



Emoji Keyboard

Company: Patrick Bürgin

Appsource URL: link

Description: Emoji Keyboard brings emoji to Microsoft Word, PowerPoint and OneNote.

It's simple, fun, and surprisingly useful.

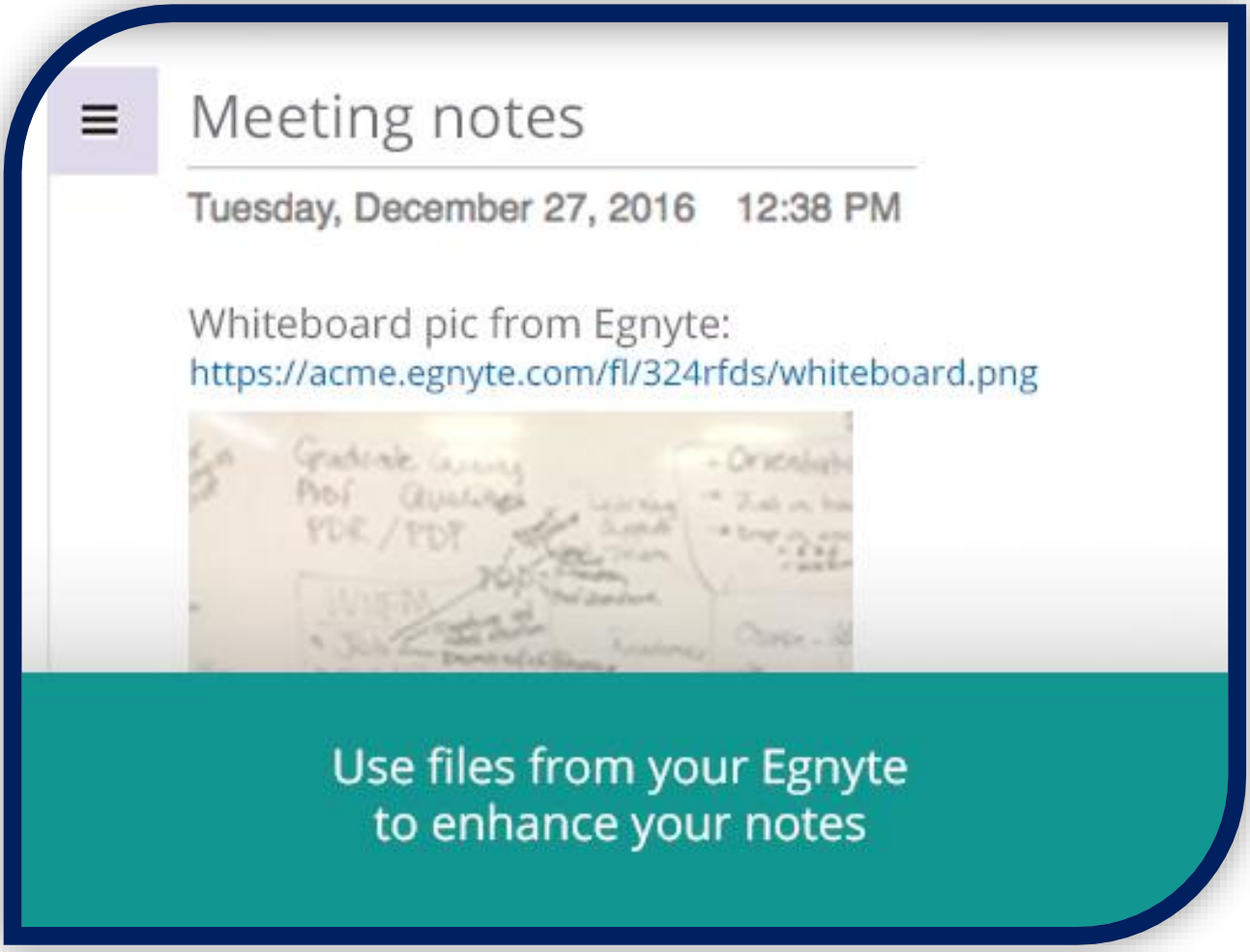
- Provides more than 1300 emoji (Unicode 9.0 standard).
- Emoji can be inserted as text or as images in different sizes.
- Offers keyword-based emoji search, and skin tone modifiers.
- Includes a comprehensive user manual to help you get started.
- Requires minimal access permissions, and does not track usage data.

Egnyte Connect - Share Files from the Cloud

Company: Egnyte, Inc.

Appsource URL: link

Description: Microsoft OneNote integrated with an Egnyte Connect account helps the users to easily embed hyperlinks in your notes to content stored in the cloud. Access and then add rich content to the notes without ever leaving OneNote. These links can even render an image preview, making for quick confirmation that collaborators are always working with the right files. Security features in Egnyte Connect allow the users to set a Link Expiration and assign Accessibility Rules to specific files right from OneNote.



CONCLUSION

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

Our Microsoft Office 365 OneNote Web-Addin 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.